Una aproximación evolutiva a la planificación en entornos HPC basada en la incorporación de criterios subjetivos

Autor: Juan Monroy Camafreita

Tese de doutoramento UDC / 2015

Directores:

Richard J. Duro Fernández

José Antonio Becerra Permuy

Programa de Doutoramento en Computación

UNIVERSIDADE DA CORUÑA

Departamento de Computación

Tesis Doctoral

Una aproximación evolutiva a la planificación en entornos HPC basada en la incorporación de criterios subjetivos

Juan Monroy Camafreita

Directores:

Richard J. Duro Fernández
José Antonio Becerra Permuy

2015

UNIVERSIDADE DA CORUÑA

D. Richard J. Duro Fernández, Catedrático de Universidad del Departamento de Computación de la Universidade da Coruña,

D. José Antonio Becerra Permuy, Contratado Doctor del Departamento de Computación de la Universidade da Coruña,

CERTIFICAN:

Que la memoria titulada:

"Una aproximación evolutiva a la planificación en entornos HPC basada en la incorporación de criterios subjetivos"

ha sido realizada por **D. Juan Monroy Camafreita** bajo nuestra dirección en el Departamento de Computación de la Universidade da Coruña, y constituye la Tesis que presenta para optar al grado de Doctor.

Fdo. Richard J. Duro Fernández
Codirector de la Tesis Doctoral

Fdo. José A. Becerra Permuy
Codirector de la Tesis Doctoral

A mi abuela

Agradecimientos

Durante el tiempo de desarrollo de esta tesis doctoral han sucedido muchas cosas, algunas muy buenas y otras no tanto, pero mucha gente ha estado involucrada de manera muy positiva en todo ello. Tanto la tesis como esas personas han formado parte de mi vida durante los últimos años por lo que, sin duda, ambas cosas están muy relacionadas entre sí y, de alguna manera, se han influido y afectado la una a la otra. Precisamente por eso esta tesis también ha sido posible gracias a ellos...

Quiero dar las gracias a Richard y a José Antonio por su tiempo, su paciencia, sus consejos, su dedicación y su guía para que esta tesis pudiera llegar a buen puerto, sin olvidarme de las oportunidades que me han brindado durante los últimos años.

Quiero dar las gracias a todo el GII por formar el fantástico marco para el desarrollo de esta tesis y por el buen ambiente que siempre ha reinado en el laboratorio.

Quiero dar las gracias al Centro de Supercomputación de Galicia por su colaboración y por proporcionar los datos utilizados en los desarrollos de este trabajo.

Quiero dar las gracias a Álex por compartir las penurias y alegrías de todo aquello relacionado con el desarrollo de nuestras tesis. Sin olvidarme de Rosa y Dani, por vernos todos en la misma tesitura.

Quiero dar las gracias a Juan Carlos por la genial portada realizada así como por su ayuda en la edición de algunas imágenes.

Quiero dar las gracias a mi padre y a Rosa por el apoyo proporcionado, que ha sido mucho, y por seguir manteniendo un hogar para mí.

Quiero dar las gracias a mi madre y a Walter por el otro apoyo proporcionado y por seguir manteniendo otro hogar para mí.

Quiero dar las gracias a mis hermanos. Fran, ya no tendrás que decirme más eso de “¡haber estudiao!”...

Quiero dar las gracias por todo a mi abuela, porque ella sí que es incondicional.

Quiero dar las gracias a Mónica, Manu y Javi por haberme enseñando tanto, ¡tanto!, de la vida estos últimos años.

Quiero dar las gracias a Lucía y a David por todos sus viernes, por todos nuestros viernes.

Quiero dar las gracias a Pincho por todas sus ideas y su punto de vista alternativo para todo.

Quiero dar las gracias a Esther y a Sergi por su hospitalidad y por tener siempre una llamada para mí en cada visita.

Quiero dar las gracias a Rubén, Óscar, Silvia, Borja y Cristina por haber sabido mantener de alguna manera mi mundo más antiguo.

Quiero dar las gracias a SiVE por el hecho de haber podido crear todos juntos algo tan duradero, a pesar de las grietas y de la lentitud.

Quiero dar las gracias a Alma, Pilar, Lu, Adriana y Cristina por su ayuda, sus ánimos, sus palabras, sus historias, sus risas...

Quiero dar las gracias a Blanca y a Antílope por hacer mis incursiones al rural en los últimos meses mucho más agradables.

Quiero dar las gracias a Bea por su compañerismo, el tiempo compartido en nuestro pequeño despacho, su apoyo, sus ánimos y la amistad que estamos forjando.

Quiero dar las gracias a mi otra abuela por haber sido capaz de no abandonar jamás mis pensamientos y el haberse mantenido de manera constante en mis recuerdos, aunque ya no esté aquí. Sí que estás aquí.

Quiero dar las gracias a todos los que no aparecéis aquí mencionados, familia, amigos y compañeros, pero que sentís que deberíais estar. Hay miles de motivos.

Si no fuera por vosotros...

Juan.-
23 de septiembre, 2015

Resumen

En el contexto de un centro de supercomputación, por muy elevados que sean los recursos, la demanda será siempre superior. Por ello, los usuarios deben realizar solicitudes para la ejecución de sus trabajos, que se ponen en espera hasta que el planificador del sistema decide pasarlos a ejecución. Pero, por desconocimiento o temor a que los trabajos sean abortados, estas solicitudes son normalmente muy imprecisas, dificultando la labor del planificador. Además, los planificadores son difíciles de configurar y en todo momento asumen que una planificación dada va a satisfacer de igual manera a todos los usuarios.

En este trabajo se propone un sistema de planificación que utiliza técnicas de computación evolutiva para permitir la definición de políticas de planificación de manera más natural y estimar las necesidades reales de recursos para lograr planificaciones más precisas. Adicionalmente, se considera el concepto de calidad de servicio percibida, posibilitando la incorporación de criterios subjetivos en el proceso de planificación para mantener un alto nivel de satisfacción en el conjunto de usuarios y en el propio centro de supercomputación. Finalmente, se modelan diversos aspectos de los propios recursos computacionales mejorando aún más la precisión en la planificación, especialmente en sistemas heterogéneos.

Palabras clave: planificación, HPC, satisfacción, recursos, estimación de recursos, modelado de usuarios, modelado de comportamiento, modelado de satisfacción, modelado de recursos.

Summary

In the context of a supercomputing center, no matter what its computational resources are, the demand will always be higher. Therefore, users must send their jobs to a queue, where they are put on hold until the scheduler decides to execute them. But, through ignorance or fear that jobs are aborted, these requests are usually very imprecise, hindering the performance of the scheduler. In addition, schedulers are difficult to configure and they assume that a given scheduling will satisfy equally to all users at all times.

This thesis proposes a scheduler for high performance computing systems based on evolutionary computation techniques to allow the definition of scheduling policies more naturally and to estimate the real needs of resources in order to achieve more accurate schedules. Additionally, the concept of perceived quality of service is considered, enabling the incorporation of subjective criteria in the scheduling process to maintain a high level of satisfaction in the set of users and in the supercomputing center itself. Finally, various aspects of the computational resources are modeled to further improving accuracy in scheduling, especially in heterogeneous systems.

Keywords: scheduling, HPC, satisfaction, resources, resource estimation, user modelling, behavior modelling, satisfaction modelling, resource modelling.

Resumo

No contexto dun centro de supercomputación, por moi elevados que sexan os recursos, a demanda será sempre superior. Por elo, os usuarios deben realizar solicitudes para a execución dos seus traballos, que se poñen en espera ata que o planificador do sistema decide pasalos a execución. Pero, por descoñecemento ou temor a que os traballos sexan abortados, estas solicitudes son normalmente moi imprecisas, dificultando o labor do planificador. Ademais, os planificadores son difíciles de configurar e en todo momento asumen que unha planificación dada vai satisfacer de igual maneira a todos os usuarios.

Neste traballo proponse un sistema de planificación que utiliza técnicas de computación evolutiva para permitir a definición de políticas de planificación de maneira máis natural e estimar as necesidades reais de recursos para lograr planificacións máis precisas. Adicionalmente, considérase o concepto de calidade de servizo percibida, posibilitando a incorporación de criterios subxectivos no proceso de planificación para manter un alto nivel de satisfacción no conxunto de usuarios e no propio centro de supercomputación. Finalmente, se modelan diversos aspectos dos propios recursos computacionáis mellorando aínda máis a precisión na planificación, especialmente en sistemas heteroxéneos.

Palabras crave: planificación, HPC, satisfacción, recursos, estimación de recursos, modelado de usuarios, modelado de comportamento, modelado de satisfacción, modelado de recursos.

Publicaciones

El trabajo desarrollado en esta tesis y otras investigaciones relacionadas han dado lugar a las siguientes publicaciones:

Bruno Fernández, Juan Monroy, Francisco Bellas y Richard J. Duro, *Visual Behavior Definition for 3D Crowd Animation through Neuro-evolution*, Hybrid Artificial Intelligence Systems, pp. 354 - 364, 2014

J. Monroy, F. Bellas, R.J. Duro, R. Lopez, A. Puentes, J. Isaac, *Automatic Speech-Lip Synchronization System for 3D Animation*, Lecture Notes in Computer Science, pp. 122 - 129, 2009

J. Monroy, J. A. Becerra, F. Bellas, R. J. Duro, *Improving Performance in HPC Centers by Modeling Users Through an Evolutionary Virtual Interface*, IEEE Transactions on Instrumentation and Measurement, pp. 1885 - 1893, 2008

A. Paz-Lopez, G. Varela, J.Monroy, S.Vazquez-Rodriguez, R.J. Duro, *HI3 Project: Software Architecture System for Elderly Care in a Retirement Home*, Advances in Soft Computing, pp. 11 - 20, 2008

A. Paz-Lopez, G. Varela, J. Monroy, S. Vazquez-Rodriguez, R. J. Duro, *HI3 Project: General Purpose Ambient Intelligence Architecture*, Proceedings of the 3rd Workshop on Artificial Intelligence Techniques for AmI (AITAmI'08), pp. 77 - 81, 2008

J. Monroy, J. A. Becerra, F. Bellas, R. J. Duro, F. López Peña, *Automatic Profiling and Behavior Prediction of Computer system Users*, Proceedings of the 2006 Workshop on Measurement Systems for Homeland Security, Contraband Detection and Personal Safety, pp. 62 - 66, 2006

J. Monroy, J. A. Becerra, F. Bellas, R. J. Duro, *Intelligent Virtual Interface for Improving Performance in HPC Centers by Modelling Users and their Satisfaction*, Proceedings of 2006 IEEE Int. Conf. on Virtual Environments, Human-Computer Interfaces, and Measurement Systems, pp. 69 - 74, 2006

J. Monroy, J. A. Becerra, F. Bellas, R. J. Duro, *Parallel Job Scheduling through Evolutionary Based Cognitive Strategies*, Proceedings of 2006 IEEE Congress on Evolutionary Computation, pp. 11043 - 11049, 2006

J. Monroy, José A. Becerra, Francisco Bellas, Richard J. Duro, and Fernando López-Peña, *A Profiling Based Intelligent Resource Allocation System*, Lecture Notes in Artificial Intelligence, pp. 840 - 846, 2005

Índice general

Capítulo 1

Introducción

El primer paso es, casi siempre, el más difícil,
pero se da con ilusión.

Los centros de computación de alto rendimiento (o *High Performance Computing, HPC*) actuales poseen supercomputadores con unos recursos computacionales que exceden en órdenes de magnitud los disponibles en los equipos de propósito general, y que los convierten en imprescindibles para resolver tareas que de otra forma serían inabordables. Miles de usuarios optan al uso de estas máquinas que, al ser escasas, les obligan a competir por sus recursos, asignados a los trabajos en función de determinadas políticas. Estas políticas generales, que establecen los administradores del centro de manera más o menos manual y homogénea sobre la población de sus usuarios, normalmente conllevan dos partes. Por un lado los objetivos comerciales, operativos o de servicio del centro, que constituyen propiamente las políticas y, por otro, una serie de reglas de planificación que pretenden implementar de forma práctica dichas políticas. Con dichas reglas los administradores buscan generalmente la optimización de medidas claramente definidas como el tiempo de ejecución de los trabajos, la tasa de trabajos servidos o la ocupación de los recursos, asumiendo que son medidas que proporcionan una alta calidad de servicio, de manera general, al conjunto de usuarios y que llevarán a la consecución de los objetivos globales del centro. Sin embargo, actualmente el nivel de complejidad de dichos centros y de los recursos implicados y la gran variabilidad de la población que hace uso de los mismos llevan a que el establecimiento explícito por parte de los administradores de reglas que implementen políticas que cumplan los objetivos comerciales de dichas infraestructuras resulte muy complejo e incluso, a veces, inabordable, especialmente cuando se buscan criterios de satisfacción de usuarios o de calidad percibida del servicio. Es por ello que esta tesis aborda el problema de la optimización automática de la planificación en dichos centros en base a objetivos o políticas generales introduciendo el concepto de calidad de servicio percibida, también a menudo llamada satisfacción de los usuarios, como un nuevo eje subjetivo. Se libera de esta forma al administrador de la definición de los conjuntos de reglas

que han de implementarse para obtener una planificación que permita alcanzar los objetivos marcados y se limita su tarea al establecimiento de dichos objetivos y sus restricciones.

Tratando de formalizar, se puede hablar de calidad percibida en contraste con calidad mensurable. La calidad mensurable es la que suele buscar el fabricante, a través de una serie de medidas estandarizadas, en el proceso de desarrollo al intentar satisfacer las necesidades identificadas. Dicha calidad está perfecta y unívocamente definida a través de la medida de los parámetros que dan lugar a la misma. Por otra parte, la calidad percibida es un valor subjetivo y cambiante y que proviene de la valoración por parte de usuarios de distintos aspectos y que, además de depender de diversos parámetros y percepciones de diferentes maneras para cada individuo, se ve afectada por el contexto, estado emocional y condiciones del mismo, con lo cual, para el mismo individuo y proceso, esta calidad percibida puede cambiar con el tiempo. En este trabajo vamos a considerar como premisa de partida que existe una relación directa entre la calidad percibida y la satisfacción del individuo y, por lo tanto, usaremos ambos términos indistintamente.

La calidad percibida general de un producto o servicio viene dada por la del conjunto de usuarios e implica la agregación de calidades percibidas individuales que no se calculan de la misma forma, pues, como ya se ha indicado, cada usuario valora diferentes aspectos. Por ello, cuando lo que se busca es la mayor aceptación posible para un producto o servicio, el objetivo debería ser maximizar la calidad percibida general, definida como la suma, o cualquier otra agregación que resulte oportuna, de las calidades percibidas individuales de los usuarios. Una manera de hacerlo es mediante el establecimiento de procedimientos que permitan hacer uso de políticas que puedan contemplar las calidades percibidas individuales, teniendo en cuenta que estas pueden cambiar con el tiempo y con el uso y que la calidad percibida general también depende de la muestra de usuarios.

Hay que destacar un tipo de sistema en particular en el que la cuestión de la calidad de servicio percibida adquiere especial relevancia: los sistemas informáticos con recursos limitados y compartidos entre múltiples usuarios que concurren y compiten por su uso. Como ya se ha indicado, tal es el caso de los centros HPC. Esto es así porque, normalmente, la citada limitación en los recursos hace inevitables los periodos de espera por parte de los usuarios para poder ser servidos, cosa que, en general, decrementa su satisfacción, pero de diferente manera en función del usuario, el momento, etc. Será a través de la adaptación a las características y preferencias de cada usuario como se tratará de minimizar este efecto en esta tesis. Como se ha mencionado, el tipo de sistema sobre el que se abordará este problema son los centros HPC, aunque, como se podrá observar, los conceptos, ideas y trabajo aquí presentados son fácilmente exportables a cualquier otro sistema con similares características.

Actualmente, los sistemas de computación de alto rendimiento, como los supercomputadores, los *clusters*, o las estructuras tipo *grid*, son herramientas indispensables para ciertas tareas que no pueden llevarse a cabo de otra manera, ya sea por motivos económicos o restricciones temporales. Suelen ser máquinas de altas prestaciones con un gran número de procesadores y potencia muy elevada mantenidas

por centros de computación de alto rendimiento. Estas máquinas permiten la ejecución de programas informáticos en paralelo logrando la consecución de resultados y objetivos en un intervalo de tiempo significativamente menor que aquel utilizado por un equipo de escritorio o de propósito general actual. Los sistemas HPC más potentes cuentan, actualmente, con cientos de miles de núcleos de procesamiento y alrededor de medio millar de estos sistemas tienen varios miles de procesadores. Esto hace fácil imaginar la gran cantidad de usuarios que utilizan y aspiran a utilizar dichos sistemas.

Resulta obvio pensar que máquinas de estas características no son sencillas de configurar, puesto que la cantidad de parámetros a manejar es muy elevada. En particular, esto es cierto en el *software* que permite el acceso a los usuarios para el envío de trabajos y la gestión de los mismos. Desde la aparición de los supercomputadores, y debido al partido y rendimiento que se les puede sacar, son cada vez más los colectivos que las utilizan y no solo provenientes de la profesión informática, como se podría pensar, sino que físicos, químicos, biólogos, médicos, ingenieros, arquitectos y meteorólogos, por citar algunos, se suman al uso de programas que requieren gran capacidad de cómputo. Además, la naturaleza de las aplicaciones es tan variada que difícilmente coinciden en lo que a necesidades se refiere, tanto de *hardware* (almacenamiento, memoria o número de procesadores, por ejemplo) como temporales (tiempo de CPU y comunicación). Es decir, por un lado existen muchos colectivos de usuarios con perfiles diferentes, con muchos usuarios a su vez. Por otro lado, y por potentes que sean los supercomputadores, siempre resultarán un recurso escaso y limitado, problema que se ve agravado por la cantidad de usuarios. Todo esto provoca que los usuarios tengan que esperar para hacer uso de los supercomputadores, espera que puede alargarse hasta días o semanas. Y de aquí la necesidad del mencionado *software* para la gestión de los trabajos y el acceso a los recursos, cuyas principal finalidad es el almacenamiento en colas de dichos trabajos y la planificación para su puesta en ejecución. Este *software* recibe el nombre de gestor de colas de procesos o *job management system* (*JMS*) y la parte encargada de desarrollar la función más importante se denomina planificador o *scheduler*.

El principal cometido de un planificador en un centro de computación de alto rendimiento es el de asignar los recursos a los procesos que los necesitan siguiendo un orden y una política determinados. Obviamente, esta labor no sería necesaria si los recursos disponibles superaran en todo momento a los recursos solicitados por los trabajos enviados al sistema, o si todos los trabajos tuviesen la misma importancia y necesitasen los mismos recursos. Sin embargo, como ya se ha dicho, esto no es así. Algunos de los gestores de colas de trabajos más utilizados son LSF (*Load Sharing Facility*) [1], PBS (*Portable Batch System*) [2] o Grid Engine [3]. Todos ellos poseen planificadores complejos y, si no los tienen, se pueden combinar con planificadores todavía más complejos como Maui [4]. Estos planificadores permiten definir políticas de asignación de recursos, aplicando distintas prioridades a los trabajos y variando estas prioridades en el tiempo para asegurar que ningún trabajo se queda permanentemente en espera, rellenando huecos en la utilización de recursos con trabajos cortos (*backfilling*) y, en ocasiones, pudiendo incluso parar trabajos para lanzar inmediatamente otros más urgentes (*preemption*). Sin embargo, y como se ha dicho, cuanto más flexibles y potentes son los planificadores para tratar así

de poder adaptarse a cualquier circunstancia, más complicada se hace la configuración y ajuste de sus parámetros. Y no solo eso, sino que el comportamiento del sistema puede cambiar inducido por los usuarios, cuyo número también cambia con el tiempo, obligando a reconfigurar y reajustar el sistema de manera periódica, con los problemas que ello conlleva (tiempo, dinero y paradas en el sistema). Además, a pesar de que uno de los objetivos primordiales, como se ha dicho, es maximizar la satisfacción de los usuarios, como en cualquier otro servicio, se intenta cuantificar a través de medidas estándar sin entrar más a fondo en lo que realmente implica la satisfacción, que es individual. Se cae en la generalización para grandes grupos de usuarios, con el agravante de que es el centro el que determina qué es lo que satisface al usuario y no el propio usuario, estableciendo políticas de optimización de tiempo de ejecución, del uso de recursos o de la tasa de ejecución de trabajos, por ejemplo.

De toda la problemática de un centro de computación de altas prestaciones existen cuatro aspectos, que en esta tesis se consideran clave, sobre los que trabajar para lograr un buen funcionamiento. Tres de estos aspectos son (el cuarto se describirá posteriormente):

1. La configuración de los JMS y su principal componente, el planificador, que es compleja, poco adaptativa y requiere frecuentes reconfiguraciones debido al cambio en las tendencias de uso así como en el conjunto de usuarios.

2. Las solicitudes de recursos, o estimaciones de recursos necesarios en caso de no existir las solicitudes, que suelen ser poco precisas pero resultan ser información fundamental para poder realizar buenas planificaciones.

3. Las políticas habituales de planificación y asignación de recursos, que asumen criterios de satisfacción generales para todos los usuarios o grandes grupos de ellos, sin tener en cuenta que una diferenciación más individual podría mejorar las planificaciones y elevar la satisfacción global.

En estos aspectos se ve reflejada la problemática principal derivada del proceso de diseño y mejora de productos, que describe Ulrich en [5]. Así, y debido a una etapa fundamental de identificación de necesidades clásica, el propio proceso de diseño del sistema adolece de estos problemas:

1. Se basa en una generalización del modo de uso y de las características deseables del sistema por parte de los usuarios. Es decir, para contentar a la mayoría se desarrolla un sistema que cumple con las necesidades comunes descartando aquellas menos frecuentes o consideradas menores.

2. Se persiguen características claramente cuantificables y mensurables ignorando, normalmente, otras que resultan más difíciles de evaluar, especialmente las más subjetivas. En la descripción de un sistema suelen aparecer múltiples cualidades o conceptos que son fáciles de definir, cuantificar, medir y manejar provenientes de la citada fase de identificación de necesidades del proceso de diseño del sistema. Tanto en el diseño como en la evolución de un producto

se suele buscar la mejora de una o varias de estas cualidades pues será fácil de evaluar y comprobar si existe tal evolución.

3. El hecho de que las tendencias de uso cambien con el tiempo implica la mejora o rediseño de un producto para adaptarse de nuevo al mercado, es decir, a las nuevas necesidades de los usuarios. Pero, en general, este proceso se lleva a cabo tras largos periodos de tiempo, haciendo poco dinámica dicha adaptación.

Estos problemas se trasladan directamente al desarrollo de planificadores y su configuración por parte de los administradores de un centro de HPC. En primer lugar, se tienen en cuenta valores fácilmente mensurables, como puede ser maximizar la ocupación de los recursos o el servir la mayor cantidad de trabajos posibles por unidad de tiempo ignorando otras medidas que podrían resultar más esclarecedoras. Por otra parte, y consecuencia de lo anterior, se generaliza lo que los usuarios quieren y se asume que todos van a dar el mismo uso al sistema. Finalmente, una nueva reconfiguración del sistema o un cambio en la política de planificación o asignación de recursos desechan configuraciones anteriores que podrían resultar idóneas para algunos usuarios, haciendo que la supuesta mejora no tenga tanta validez como se pretende.

El cuarto aspecto clave al que se hacía mención con anterioridad está directamente relacionado con el uso de los recursos. En los planificadores tradicionales no se suele tener en cuenta el comportamiento real en cuanto a disponibilidad y comportamiento general de los recursos, asumiendo que se comportan según indica el fabricante y que están siempre disponibles. Esto no es siempre así, especialmente en entornos complejos tipo *grid* o *cloud* y, por lo tanto, si se requiere una planificación precisa es necesario poder modelar el comportamiento de los recursos.

Es en torno a los tres problemas del proceso de diseño de un sistema, y centrando la atención en los cuatro aspectos clave descritos, donde se establecen los objetivos concretos de este trabajo, recogidos en el capítulo 2. Dichos objetivos se pueden resumir como el estudio y desarrollo de una aproximación evolutiva a la planificación en un gestor de colas de procesos para la maximización de la calidad de servicio percibida por los usuarios de un sistema de computación de alto rendimiento. Para ello se considerarán no solo parámetros fácilmente mensurables, sino también valores subjetivos. Se busca, de esta manera, que el administrador del sistema pueda establecer sus políticas en términos generales sin tener que entrar en el detalle de las reglas de operación.

Desde un punto de vista metodológico, se aborda el trabajo en esta tesis en cuatro fases de forma incremental, realizando pruebas parciales a medida que se van añadiendo elementos al sistema propuesto, así como pruebas finales ya con el sistema completo y todos sus elementos integrados.

Primeramente, se afronta el problema de aliviar la necesidad de reconfiguración del planificador y facilitar la definición de políticas de planificación. En esta línea, Streit propone en [6, 7] un sistema adaptativo que selecciona dinámicamente entre tres políticas distintas en función del estado de la cola de trabajos, utilizando en cada momento la mejor política. No obstante, el número de políticas es muy bajo

y está prefijado. Además, es necesario ajustarlas de manera individual y manual. Para evitar esto, Min y Cheng presentan en [8] un algoritmo genético para llevar a cabo la planificación buscando la minimización del *makespan* (tiempo total de ejecución de toda la cola de trabajos). Sus resultados son interesantes, pero limitados a sistemas de 30 trabajos y 10 nodos, alejándose bastante de un ejemplo real. En [9] y [10] también se estudian técnicas evolutivas para la minimización del *makespan*, no obstante, y como se demuestra en [11], la minimización de esta medida no siempre conduce a planificaciones adecuadas. Este problema se trata en esta tesis mediante el desarrollo de un planificador evolutivo que, además de realizar el trabajo de planificación, posibilita la definición de políticas de planificación de manera natural, facilitando su configuración y evitando su continua reconfiguración debido a cambios en el comportamiento de los usuarios. Asimismo, permite la integración de diversas medidas, como las de satisfacción, en las políticas de planificación.

En una segunda fase, se aborda el problema de la escasez de precisión en las solicitudes de recursos de los usuarios. Para hacer frente a este problema, algunos autores se centran en la predicción del momento de inicio de los trabajos para realizar una asignación adecuada de recursos y un balanceo de carga, como el de Smith *et al.* [12]. El principal problema de esa aproximación es que se siguen utilizando los valores de solicitud proporcionados por los usuarios con todas sus imprecisiones. Li *et al.* realizan en [13] esto mismo pero prediciendo previamente el tiempo de ejecución de los trabajos. Esto implica la creación y verificación previas de una serie de plantillas basadas en las características de trabajos. Cada trabajo a predecir debe encajar en una de estas plantillas o categorías predefinidas, a partir de las cuales, y mediante técnicas estadísticas, predicen la duración. En trabajos como el de Piro *et al.* [14], se predicen otro tipo de recursos como la memoria RAM o el almacenamiento utilizado, pero siguen basándose en plantillas o categorías. En [15] y [16] se presentan sistemas en los que se siguen categorizando trabajos, pero de manera más pormenorizada, teniendo en cuenta otros factores como el propietario del trabajo y otros datos asociados, tales como la procedencia. La solución propuesta en esta tesis se concreta en forma de un sistema de optimización de solicitudes que “aprende”, mediante la utilización de técnicas de neuroevolución, la conducta del usuario en cuanto a la solicitud de los recursos. Esto se consigue basándose no solo en datos históricos, sino también de manera *on–line*. Con este aprendizaje se pretende detectar las desviaciones entre las solicitudes de los usuarios y los recursos realmente utilizados con el fin de predecir el uso real de dichos recursos. Se elimina así la necesidad de la categorización indicada en las aproximaciones encontradas en la bibliografía y se particulariza el aprendizaje de solicitud de recursos a los usuarios individuales. Por otro lado, se propone un sistema que se actualiza de manera dinámica en el momento en que se disponga de nueva información, es decir, con cada nuevo trabajo que termina. Así, este sistema permite proporcionar al planificador estimaciones del uso de recursos en los trabajos enviados a un HPC por parte de los usuarios, para tenerlas en cuenta en sus decisiones de planificación.

Como ya se ha indicado, la introducción de medidas de satisfacción o de calidad percibida en las políticas de planificación es otro de los grandes retos de esta tesis. En la bibliografía, la inmensa mayoría de trabajos que tratan la satisfacción lo hacen al respecto de sistemas de información clásicos, que difieren en gran medida

del ámbito de supercomputación aquí abordado, para el cual no se han encontrado trabajos que manejen los conceptos de satisfacción o calidad de servicio percibida. Así pues, como tercera fase del trabajo presentado en esta tesis, se aborda la creación e integración con el resultado de las dos fases previas, de un sistema novedoso de estimación de la satisfacción de los usuarios. Su objetivo es modelar dicha satisfacción de los usuarios con el propósito primordial de poder realizar predicciones respecto a la misma que ayuden a escoger entre posibles planificaciones.

Finalmente, el cuarto punto que se lleva a cabo es la obtención de modelos o perfiles de los recursos, que permitan predecir su estado y disponibilidad, con el fin de mejorar las planificaciones evitando grandes desviaciones debido a la disparidad entre la disponibilidad teórica y la real. Esto posibilita optimizar la asignación de recursos con el fin último de apoyar la optimización de la calidad de servicio percibida. Este subsistema gana relevancia en entornos *grid*, cada vez más numerosos, donde los recursos son heterogéneos y su disponibilidad muy variable.

En resumen, se pueden indicar como principales aportaciones del trabajo que aquí se presenta las siguientes:

- Estudio y desarrollo de un planificador basado en algoritmos evolutivos, que facilita la tarea de configuración y permite la obtención automática y adaptativa de planificaciones efectivas, así como la incorporación de criterios subjetivos en las políticas de planificación.
- Un algoritmo de modelado del comportamiento de los usuarios, ideado para funcionar *on–line* y adaptarse gradualmente y de forma continua a los cambios de los usuarios a medida que estos se producen. Este algoritmo permite la creación y actualización de modelos capaces de predecir el uso de recursos de los trabajos enviados por los usuarios al sistema.
- Una aproximación para obtener medidas indicativas de la satisfacción de los usuarios huyendo de grandes intrusiones. Esto, conjugado con el planificador evolutivo, permite generar planificaciones que optimizan la satisfacción del conjunto de usuarios sin hacer explícito ningún tipo de conocimiento.
- Un sistema de modelado de recursos que permite dos beneficios adicionales. Por un lado, capturar el comportamiento de los recursos para lograr planificaciones adaptadas a ellos, y por otro, permitir a los administradores del centro incluir en las políticas de planificación aspectos que influyan directamente en la satisfacción del centro y que tengan que ver con el uso de recursos. Además, este sistema extiende la utilidad del resto hacia entornos de recursos heterogéneos o de tipo *grid*.

En lo que respecta a la estructura de esta memoria, en el capítulo 2 se especifican concisamente los objetivos perseguidos. En el capítulo 3 se enmarca el trabajo y se hace una revisión bibliográfica del área que abarca. A continuación, en los capítulos 5, 4, 6 y 7 se estudian, detallan y prueban los cuatro subsistemas mencionados de los que se compone este trabajo: el sistema de optimización de solicitudes, el sistema de planificación evolutiva, el sistema de optimización de la satisfacción y

el sistema de optimización de la asignación de recursos. En el capítulo 8 se presenta el sistema completo con los cuatro subsistemas integrados, incluyendo una prueba global de funcionamiento. El 9 revisa las principales conclusiones y aportaciones fundamentales de esta tesis, mientras que en el capítulo 10 se presentan las principales líneas de trabajo futuro.

Capítulo 2

Objetivos

Antes de salir hay que tener muy claro el
destino. O tener muy claro que
no hay destino fijo.

La hipótesis principal de este trabajo es que la consideración de medidas subjetivas indicadoras del nivel de satisfacción al respecto del funcionamiento de un sistema, conjuntamente con la utilización de algoritmos de optimización que posibiliten la predicción de valores para estas medidas ante determinadas actuaciones, permite incrementar la calidad percibida del sistema a todos los niveles y, por tanto, optimizar su funcionamiento. Se intuye, además, que este enfoque es particularmente acertado en sistemas con múltiples usuarios accediendo a recursos limitados y compartidos.

Bajo esta hipótesis, y estableciendo como ámbito concreto de trabajo los entornos de computación de altas prestaciones, el objetivo de este trabajo es el de estudiar y desarrollar un método que permita implementar un planificador de un gestor de colas que, de manera dinámica y autónoma, mantenga un alto grado de satisfacción, tanto en el conjunto de usuarios como en el centro. Esto es, que maximice la calidad percibida individual y la general del centro.

A continuación se describen una serie de subobjetivos planteados como pasos necesarios de una metodología orientada a la consecución de dicho objetivo global. Lo propuesto para alcanzar cada uno de estos subobjetivos representaría individualmente por sí mismo una mejora notable en caso de integración en un sistema de gestión de trabajos de un centro de supercomputación, no obstante, el mayor beneficio aparece con su aplicación conjunta.

Como primer paso, se plantea abordar la minimización del número de parámetros de configuración en el planificador, de forma que se permita de manera sencilla el cambio de políticas de planificación y de asignación de recursos, y que se posibilite, además, hacerlo de manera declarativa explicitando qué buscar y no cómo

buscarlo, y que las planificaciones puedan obtenerse automáticamente. Para todo ello se plantea la utilización de un algoritmo evolutivo, ya que modificando la función de calidad de este se pueden obtener automáticamente planificaciones acordes a la política deseada.

El segundo subobjetivo consiste en el desarrollo de un mecanismo que permita modelar el comportamiento de cada usuario de forma individualizada en términos de recursos solicitados a la hora de enviar trabajos a un sistema HPC. Así, con este mecanismo se trata de estimar las características reales de los trabajos de los usuarios para poder obtener una planificación lo más adecuada posible, que no dependa de la exactitud de lo indicado por el usuario, que suele sobrestimar las necesidades reales de sus trabajos o directamente indica siempre los mismos valores, lo cual representa uno de los mayores problemas de base de un planificador para entornos HPC. Este mecanismo debe operar *on–line* y debe ser adaptativo, es decir, debe "aprender" continuamente de la interacción del usuario con el sistema.

Ya con un mecanismo que permita estimar con precisión las características de los trabajos de los usuarios, y con un algoritmo de optimización que posibilite obtener automáticamente planificaciones a partir de una especificación de alto nivel y dichas estimaciones, el siguiente paso consiste en abordar el tema de la satisfacción o calidad percibida del servicio por parte de los usuarios, de manera que esta información se pueda incorporar al planificador. Para ello se plantea un sistema que, de forma análoga a lo realizado en el primer paso con las solicitudes de recursos, permite predecir la satisfacción de los usuarios ante una planificación concreta y considerar individualmente la calidad percibida por cada usuario en el proceso global de planificación de trabajos.

Dado que el trabajo aquí presentado pretende incrementar la calidad percibida del funcionamiento del sistema tanto de los usuarios del centro HPC como de sus gestores, el cuarto subobjetivo consiste en desarrollar un mecanismo que modele los recursos computacionales, lo que abre la puerta a la inclusión en la política de planificación de atributos o características de estos. Es obvio que a mayor satisfacción de los usuarios del centro HPC, mayor satisfacción de sus gestores, pero estos últimos pueden tener que considerar otros aspectos como costes, fiabilidad, etc. Además, para aquellos entornos de ejecución no homogéneos donde existen recursos de diferentes características y con distintos requisitos de uso y disponibilidad, este mecanismo de modelado de recursos permite a un JMS tomar mejores decisiones en la etapa de planificación. Para ello el sistema debe monitorizar y perfilar los recursos continuamente actualizando los modelos correspondientes para predecir su estado en el futuro, así como determinar los tiempos de ejecución de los trabajos que hagan uso de ellos.

Finalmente, se plantea también como subobjetivo estudiar la viabilidad de la implantación de un sistema que siga las directrices de este trabajo en un entorno real. Para ello se llevarán a cabo todas las pruebas de integración y funcionamiento que sean necesarias para comprobar la validez de la hipótesis inicial y de la solución propuesta.

Capítulo 3

Marco de trabajo y estado del arte

Si no nos salimos del camino nunca llegaremos
a nuevos destinos. Si lo hacemos, es mejor
asegurarse con un mapa.

Los entornos de supercomputación son un caso concreto y paradigmático de sistemas de recursos limitados y compartidos. En este tipo de sistemas, entre los que destacan tres arquitecturas principales, los supercomputadores clásicos, los *clusters* y los *grids*, existe un componente principal, el planificador, que tiene dos responsabilidades principales: asignar recursos y decidir qué trabajos y en qué momentos entran a ejecución con los recursos asignados, es decir, la propia planificación.

Uno de los grandes problemas de los planificadores radica en su complejidad, lo que hace que el proceso de configuración sea complicado y tedioso y que no siempre dé buenos resultados. Sumado a esto, cambios en el conjunto de usuarios y su comportamiento, en el conjunto de recursos o en las políticas administrativas de los centros de HPC obligan a reconfigurar los planificadores con excesiva frecuencia.

El segundo gran problema de los planificadores es que no cuentan con información de solicitudes de recursos precisas, realizadas por los usuarios, para poder proporcionar planificaciones precisas. Las causas principales de la imprecisión en las solicitudes son que los usuarios desconocen la cantidad exacta de recursos que sus trabajos necesitan, que no realizan ninguna solicitud explícita por pereza o desconocimiento o que, simplemente, hacen solicitudes por exceso para evitar que el sistema detenga sus trabajos por un uso excesivo de recursos. Por otro lado, y en parte debido a la complejidad del proceso de configuración de los planificadores, los administradores de los centros hacen asunciones y generalizaciones acerca de lo que los usuarios buscan en el servicio y establecen políticas de planificación partiendo de parámetros medibles como el *makespan* o la tasa de trabajos servidos, generando un servicio que no siempre mantiene el contento general de los usua-

rios y el centro. Todos estos problemas ganan magnitud en entornos *grid*, donde la heterogeneidad de los recursos así como su disponibilidad dificultan todavía más la configuración, el establecimiento de políticas, la asignación de recursos y, en definitiva, la consecución de un servicio eficiente y que mantenga un alto nivel de satisfacción.

Dado que este trabajo se centra en la optimización de un planificador que aborde estos problemas se ha considerado necesario realizar revisiones del marco de trabajo y del estado actual de las investigaciones de manera separada para cada uno de ellos. A continuación, y tras una introducción al marco central de esta tesis, se recogen algunos de los trabajos más significativos en el ámbito de la planificación en centros de HPC, teniendo en cuenta el campo del modelado de usuarios y comportamientos para la predicción de recursos, el de medida de satisfacción y el modelado de recursos.

3.1. Computación paralela de alto rendimiento

Como ya se ha comentado, los entornos de computación de alto rendimiento son un buen ejemplo de sistemas de recursos limitados y compartidos por lo que es en ellos donde se centra la atención de este trabajo. Las principales arquitecturas de este tipo de sistemas son los supercomputadores, los *clusters* y los *grids*.

Los supercomputadores nacen en la década de los 60 de la mano de Seymour Cray como máquinas que exceden en mucho la capacidad computacional de un ordenador de propósito general por medio del uso de técnicas y tecnologías punteras no disponibles o inexistentes para otro tipo de ordenadores, pasando por arquitecturas innovadoras y múltiples procesadores. El *Atlas*, desarrollado en una colaboración entre la Universidad de Manchester y la compañía *Ferranti*, junto con algunos otras máquinas desarrolladas por Cray en *CDC* (Control Data Corporation), fue uno de los primeros supercomputadores, que contaba con unos pocos procesadores.

Los procesadores de un supercomputador están altamente conectados entre sí mediante redes de muy alta velocidad de forma que el paso de datos entre ellos se facilita y resulta muy rápido. Apoyado esto mediante arquitecturas de acceso a memoria común, el intercambio de datos se aligera todavía más. Estos tres pilares forman la base de un supercomputador clásico: múltiples y potentes procesadores, alto acoplamiento y arquitecturas de memoria compartida.

Grandes son las inversiones de capital para la investigación, el diseño y la construcción de supercomputadores cada vez más potentes, convirtiéndose, a veces, es una competición entre las distintas compañías involucradas en estos desarrollos. Una referencia básica en el mundo de la supercomputación es la lista TOP 500 [17] de los 500 supercomputadores de propósito general más potentes del mundo ordenados por rendimiento. La lista, coordinada por Hans Meuer, Erich Strohmaier, Jack Dongarra y Horst Simon, se actualiza dos veces al año con la ayuda de expertos de la comunidad basándose en resultados de rendimiento sobre el *benchmark LINPACK*, un denso conjunto de ecuaciones lineales. En el momento de escribir es-

te trabajo, el puesto número 1 de la lista[1] lo ocupa el sistema *Tianhe–2*, también llamado *MilkyWay–2*, del *NUDT* (*National University of Defense Technology*), en China, con un total 3.120.000 *cores* y un rendimiento máximo, obtenido con el citado *benchmark*, de 33.862,7 TFlop/s. Estos números, que muestran el potencial de estos sistemas, dan una idea de lo solicitados que llegan a ser y del elevado número de usuarios que hacen uso de ellos, apoyando las principales motivaciones de esta tesis. Por ejemplo, y sin ir más lejos, en el Centro de Supercomputación de Galicia, el CESGA [18], existen dos máquinas principales, el *cluster SVG* [19] y el *FinisTerrae* [20], cada una con su conjunto de usuarios. El primero de ellos tiene una media de trabajos diarios en ejecución entre 312 y 374, mientras que el número de trabajos en espera se halla entre 240 y 401. En el caso del segundo, estos números son de 180–237 y 82–257 respectivamente[2]. Actualmente, solo dos de los sistemas de la lista se encuentran en España. Uno es el *MareNostrum* del BSC (*Barcelona Supercomputing Center*) [21] que se sitúa actualmente en el puesto 77 de la lista[3]. El segundo y último es el *TEIDE-HPC* del *ITER* (Instituto Tecnológico de Energías Renovables) [22] en Tenerife, que ocupa el lugar 260.

Los *clusters* aportan una visión diferente. La idea básica es la interconexión de diversos nodos de procesado proporcionando, además, la infraestructura suficiente como para poder ejecutar una aplicación en varios de estos nodos de manera simultánea y repartiéndose el trabajo, de manera similar a cómo se hace en un supercomputador tradicional, donde dichos nodos son ordenadores comerciales o de propósito general. Pueden verse como un caso particular de superordenador con arquitectura de memoria distribuida. Esta idea queda recogida por primera vez de la mano de Gene Amdahl en 1967, que por entonces pertenecía a *IBM Corporation*. Amdahl describe en [23] la ganancia y las limitaciones de ejecutar partes de una aplicación en paralelo.

En términos definitorios se puede decir que un *cluster* es un sistema de procesado paralelo o distribuido que consiste en una colección de ordenadores, que individualmente son completos y funcionales, interconectados entre sí mediante una red de comunicaciones de baja latencia. Un sistema así construido puede ser mejorado de diversas formas:

- Los nodos u ordenadores individuales pueden ser mejorados añadiéndoles recursos adicionales, como más memoria RAM o más capacidad de almacenamiento sin un coste excesivo.
- Se pueden añadir nuevos nodos al sistema.
- Pueden configurarse *clusters* de *clusters*.

Así, las diferencias principales entre los supercomputadores y los *clusters* son que, de manera general, los primeros poseen nodos más potentes y mejor interconectados pero más costosos de construir y que tienen una arquitectura de memoria

[1] Los datos han sido extraídos de la lista actual en el momento de escribir esta memoria, que es la de junio de 2015.

[2] Estos datos corresponden a la tercera semana de julio de 2015.

[3] En noviembre de 2004, este sistema ocupaba el puesto número 4.

compartida, frente al esquema distribuido de los segundos. Estas diferentes características hacen que los trabajos destinados a ejecutarse en un tipo de sistema u otros difieran entre sí, hecho que se ve reflejado en las colas de espera de ambos tipos de sistemas.

A finales de los 90, Foster y Kesselman, considerados los padres de la tecnología *grid*, sientan sus bases en [24], trabajo que es ampliado y actualizado en 2003 en [25] y ampliamente referido. El término proviene de una metáfora con la red eléctrica (*power grid* en inglés) donde se pretende que el acceso a los recursos sea tan sencillo como el acceso a la electricidad.

En términos de computación, se llama *grid* a un sistema que interconecta a través de redes de área extensa (*Wide Area Networks*, WAN) un conjunto de recursos de computación de diversos tipos como pueden ser nodos de procesado (PCs, estaciones de trabajo, *clusters* etc.), almacenamiento o aplicaciones, por ejemplo. La idea principal es similar a la de los *clusters*, la resolución de un problema complejo mediante la interconexión de múltiples recursos sencillos. En general, un *grid* es un tipo de sistema de computación en paralelo basado en ordenadores completos conectados a una red mediante interfaces de red estándar. La ventaja frente a los *clusters* es que se pueden configurar sistemas con muchos nodos de cómputo a mucho menor coste que el de diseñar y construir un *cluster* completo. Por tanto, suele suceder que un *grid* conforma un sistema mucho más heterogéneo que un *cluster*, además de estar más disperso geográficamente. Por otro lado, la principal desventaja radica en que, en general, las conexiones de redes no son de alta capacidad repercutiendo directamente en la calidad de las comunicaciones entre los distintos nodos. Así, los *grids* son adecuados a aquellas situaciones en las que los trabajos paralelos pueden ejecutarse sin una elevada necesidad de intercomunicación entre ellos o sus subtareas.

Un caso particular dentro de la tecnología *grid* es la computación voluntaria. Este término, acuñado por Sarmenta *et al.* [26], describe una forma de computación en la que los usuarios de ordenadores personales donan parte de sus recursos libres (CPU, RAM, espacio en disco o conexión a internet, por ejemplo) para la resolución de tareas muy complejas que requieren gran cantidad de cómputo, lo que permite reutilizar recursos en lugar de construir unos nuevos y válidos para este tipo de tareas. La primera gran aplicación de la computación voluntaria fue GIMPS (*Great Internet Mersenne Prime Search*) [27], que es un proyecto colaborativo, ideado por Woltman, que busca, desde 1996, los números primos de Mersenne[4]. Uno de las aplicaciones con mayor repercusión y un buen ejemplo de computación voluntaria es el proyecto *SETI@home* (*Search for Extraterrestrial Intelligence*) [28], iniciando en mayo de 1999, y todavía vigente, por David Anderson *et al.* [29, 30] en la Universidad de Berkeley. Basándose en el uso de radio–telescopios para la escucha de señales de radio de banda estrecha, el proyecto tiene como objetivo la búsqueda de inteligencia extraterrestre, para lo cual ejecuta un software en millones de PC's que analizan los datos de las escuchas.

[4]Los números primos de Mersenne son aquellos números primos de la forma $M_n = 2^n - 1$. Los cuatro primeros números primos de Marsenne son 3, 7, 31 y 127. Actualmente, el último número oficial es el 44º, es decir, el $M_{232,582,657} = 2^{232,582,657} - 1$

Es habitual que los centros de supercomputación administren diversos sistemas de distintas características. En general buscan proporcionar un amplio abanico de recursos con el fin de poder acoger trabajos con diferentes requisitos. Esto complica más la tarea de gestión del centro, que debe administrar diversas máquinas por separado, cosa que hace deseable unificar la configuración de las máquinas para facilitar la tarea e, incluso, poder tratarlas como un sistema heterogéneo único y completo. Así, en la problemática a tratar, como ya se ha comentado, la configuración de los recursos, la gestión de las peticiones de uso o solicitudes de ejecución de trabajos, la asignación de recursos y mantener altos niveles de satisfacción en los usuarios son aspectos que deben ser resueltos. Los sistemas de gestión de trabajos, que se presentan a continuación, son el motor principal para los sistemas de supercomputación y son los encargados de lidiar con todos estos aspectos.

3.2. Gestores de colas y planificadores

Como ya se ha indicado, los gestores de colas o sistemas de gestión de trabajos (JMS) son el *software* encargado de gestionar las máquinas, colas y trabajos de los entornos de supercomputación. Una de sus funciones principales es la de asignar los recursos a los trabajos y decidir cuáles de ellos pasan a ejecución, es decir, planificar.

Haciendo una revisión de los gestores de colas existentes, también llamados gestores de recursos, un buen punto de partida es *Network Queueing System* (NQS), que puede ser considerado el padre de todos ellos y fue desarrollado por la NASA (*National Aeronautics and Space Administration*) dando lugar a la semilla del estándar POSIX 1003.2d [31], que define el interfaz de usuario con un sistema distribuido de colas. Basándose en este estándar surgieron nuevos JMS que mejoraban las capacidades del NQS, como *Network Queueing Environment* (NQE) [32], *Portable Batch System* (PBS) [2] o *Condor* [33], el cual a su vez influyó en dos JMS posteriores: *LoadLeveler* [34] y CODINE (*Computing in Distributed Network Environment*) [35]. Los JMS más importantes en la actualidad y los más utilizados en muchos de los sistemas de la lista TOP 500 son, aunque se puede encontrar cualquiera de los mencionados e incluso otros, el propio PBS, en alguna de sus versiones, LSF (*Load Sharing Facility*) [1] y Sun Grid Engine (SGE) [36], nacido de la compra de *Gridware*, la cual a su vez era resultado de la fusión de los creadores de CODINE (Genias Software) con Chord Systems, por parte de Sun. En 2010, con la compra de Sun por parte de Oracle, el código del SGE, que pasó a llamarse *Oracle Grid Engine* (OGE), [3] dejó de distribuirse con los binarios, por lo que la comunidad *Grid Engine* se hizo cargo de la versión *open source* de lo que pasó a llamarse *Open Grid Scheduler* (OGS) [37]. Oracle siguió manteniendo su versión comercial, pero finalmente, en diciembre de 2010, cedió plenamente los derechos a la comunidad *Grid Engine* para mantener la versión libre de manera oficial.

Aunque cada gestor de colas pueda poseer diferentes características que lo distinga de sus competidores, todos ellos cuentan con una serie de funcionalidades principales comunes:

Gestión de recursos Permiten añadir o eliminar recursos al sistema, denomina-

dos normalmente nodos, ya sean equipos de escritorio, supercomputadores o *clusters* completos o nuevas unidades de proceso, así como su configuración para poder ser utilizados.

Visión unificada y punto de acceso único Proporcionan un punto acceso único y global por parte de los usuarios, al cual se conectan para interactuar con el sistema. Además, este punto de acceso unifica la visión de todos los nodos del sistema por heterogéneos que sean haciendo que todos ellos se puedan utilizar de la misma manera.

Control de trabajos Tanto para los usuarios como para los administradores, un gestor de colas proporciona funcionalidades de control de trabajos, es decir, debe permitir el envío de trabajos junto con sus solicitudes a los usuarios, la eliminación de los trabajos de las colas, la cancelación o interrupción de trabajos en ejecución, la monitorización y consulta de trabajos, la ejecución y la finalización de los mismos. Todo esto es posible bajo el punto de acceso único y para todo el conjunto de nodos configurado.

Gestión de usuarios Permiten el alta, la modificación y la baja de usuarios, así como su configuración en términos de control de acceso, prioridad, *tickets*[5] de ejecución y restricciones.

Gestión de colas Proporcionan un mecanismo de configuración de diversas colas para trabajos con requisitos comunes. Así, los administradores pueden hacer particiones del conjunto de nodos para ser asignados a colas concretas. De esta manera, por ejemplo, podría configurarse una cola para trabajos monoprocesador muy cortos y otra para trabajos multiprocesador más largos. Los usuarios deben decidir a qué cola concreta envían sus trabajos y asegurarse de que esta satisface los requisitos de sus trabajos.

Planificación Por último, deben proporcionar un mecanismo de planificación para hacer la asignación de recursos y para poner en ejecución los trabajos de las colas. Algunos gestores de colas no incluyen un planificador, teniendo que hacer uso de uno de terceros. Esto es así porque el propio planificador ya es un elemento muy complejo de por sí. No obstante, aunque los gestores de colas sí integren uno, suele ser posible el uso de otro de terceros, como por ejemplo, el Maui [4]. Independientemente del planificador usado, este también se debe configurar para su funcionamiento de manera coherente con el resto del sistema, es decir, los usuarios, los nodos y las colas.

Es en estas funciones donde se aprecian los fuertes requisitos de configuración que tienen estos sistemas, y que constituyen uno de los principales problemas comentados. En lo que respecta a esta tesis, el problema fundamental que se aborda es el de la planificación y su configuración. Para poder resolver este problema, se ha realizado una revisión de investigaciones cuyos trabajos han ido en la dirección de aliviar la necesidad de configuración y reconfiguración continua así como de facilitar dicha tarea.

[5] En los sistemas basados en *tickets*, los administradores asignan una serie de puntos, o crédito, a los usuarios que estos canjean con cada trabajo que envían a ejecución, pudiendo evitar así situaciones en las que un usuario acapara demasiados recursos.

3.3. Planificación

Los planificadores de procesos de los sistemas de colas en centros de computación de alto rendimiento son, generalmente, muy configurables a través de una elevada cantidad de parámetros de ajuste. Es precisamente esa flexibilidad la que los dota de gran complejidad cuando se trata de lograr buenas políticas de planificación. Por ejemplo, en la típica situación de la existencia de múltiples colas para priorizar diversos tipos de trabajos en cada cola. Al hilo de este ejemplo es bastante habitual encontrar varias colas con diferentes restricciones en lo relativo al máximo uso de recursos permitido, o varias colas para trabajos de usuarios con distintos privilegios. También son bastante comunes los sistemas en los que se adjudican *tickets* a los usuarios con el fin de evitar que un mismo usuario envíe un número excesivo de trabajos al sistema. Este tipo de configuraciones es siempre una solución de compromiso que dista de una solución realmente adecuada, guiando al sistema hacia situaciones en las que algunas colas están repletas de trabajos mientras que otras están vacías o casi vacías, generando la necesidad de una reconfiguración del sistema. Es probable, incluso, que el problema vuelva a aparecer una y otra vez tras las reconfiguraciones.

En la revisión realizada se han identificado una serie de líneas de investigación principales. Una primera línea se centra en estudios que se basan en análisis histórico del funcionamiento de los sistemas para detectar anomalías y oportunidades de mejora y poder reajustar el planificador en consecuencia. Una segunda línea más interesante considera los sistemas adaptativos como una alternativa válida para un ajuste automático de los parámetros de configuración. Partiendo de esta última, algunos investigadores se centran en el uso de algoritmos evolutivos para llevar a cabo dicho ajuste. Algunos de los trabajos que proponen sistemas adaptativos parten del uso de datos históricos, al igual que la primera de las líneas. Otros investigadores han orientado su trabajo a la configuración de planificadores para optimizar determinadas medidas de rendimiento, como el *makespan*. Otros intentan compensar las deficiencias en las planificaciones mediante técnicas alternativas. Finalmente, una línea de investigación se centra en la mejora de sistemas teniendo en cuenta que el entorno es dinámico y los recursos son heterogéneos.

En la primera de las líneas identificadas, existen algunos ejemplos de investigaciones en las que se intenta, *a posteriori*, ajustar el funcionamiento de los planificadores analizando el comportamiento del sistema en el pasado y realizando simulaciones sobre cómo hubiera sido este si los parámetros del planificador fuesen distintos. Tal es el caso, por ejemplo, del trabajo realizado por Feitelson *et al.* sobre el IBM SP2 [38, 39, 40], el realizado por Moreira *et al.* sobre el ASCI Blue-Pacific [41, 42] o el estudio del efecto de diversas técnicas sobre la planificación, como el *gang-scheduling* [43, 44] o el *backfilling* [38, 45, 46]. El principal problema de tomar como base datos históricos es la incapacidad de adaptarse con suficiente velocidad a los cambios que puedan darse en estos entornos, que suelen ser muy dinámicos. Además, estos trabajos no resuelven el problema de manera automática, sino que necesitan de intervención humana para el reajuste de los planificadores, que aunque puedan resultar más eficaces, siguen adoleciendo de los problemas ya comentados.

Entrando en la corriente que propone sistemas adaptativos para la configuración de los planificadores, un buen ejemplo es el propuesto por Streit. En [6] desarrolla un sistema que denomina *Planificador dynP* y que cambia dinámicamente entre tres tipos de políticas de reordenación de los trabajos en el JMS en función de dos parámetros (umbrales) ajustables manualmente que hacen referencia a tiempos de ejecución promedio estimados para los trabajos en el JMS. En [7], el autor adapta su algoritmo, denominándolo *Self–Tuning dynP*, para no tener que ajustar de forma manual ningún parámetro. Simplemente, en cada decisión, el sistema calcula un valor de calidad para cada una de las tres posibles políticas (simulando el resultado de aplicar estas sobre la cola actual) y utiliza aquella con un mayor valor de calidad. La calidad viene dada por una fórmula que tiene en cuenta el tiempo medio de respuesta, la duración prevista y los recursos solicitados por cada trabajo. Este trabajo solamente utiliza tres posibles políticas de planificación, lo que hace innecesario el uso de algoritmos de búsqueda a costa de una menor flexibilidad y capacidad en la política de planificación.

Partiendo de las ideas de Streit, y particularizando el caso de los sistemas adaptativos, se han llevado a cabo diversas investigaciones referentes al uso de técnicas evolutivas en planificadores en entornos de computación. Aunque algunos de estos trabajos no están orientados a sistemas de colas sí resultan interesantes por ser extrapolables al ámbito de esta tesis. Uno de los primeros intentos fue el de Feitelson y Naaman [47], que desarrollaron un sistema que se ajusta automáticamente utilizando para ello el tiempo libre de CPU, con lo que el proceso es asíncrono respecto del planificador, para ejecutar evoluciones sobre datos obtenidos de ficheros de registro permitiendo que el sistema ajuste automáticamente los parámetros del planificador. En realidad, el marco ahí presentado no era exclusivo para optimizar el planificador, sino para optimizar cualquier elemento del sistema informático, siempre y cuando se pudiesen tener los datos suficientes como para contar con la posibilidad de realizar la evolución. Como el caso del planificador es un claro ejemplo de cuando eso es posible, lo aplicó sobre él. El problema de este sistema es que la evolución se lleva a cabo usando datos antiguos, con lo que si el grado de dinamismo es alto no será posible adaptarse con la suficiente rapidez. De hecho, los autores realizaron la prueba sobre datos históricos que abarcaban tres meses, con lo que el sistema no puede adaptarse a cambios en intervalos de tiempo inferiores.

Dentro del uso de técnicas evolutivas, muchos investigadores han dirigido sus esfuerzos a maximizar determinados parámetros de rendimiento. Por ejemplo, Min y Cheng presentan en [8] un algoritmo genético para minimizar el *makespan* (tiempo total de ejecución) en el caso de *clusters* homogéneos. Esta solución aporta ventajas frente a un procedimiento heurístico y un algoritmo de enfriamiento o recocido simulado (*simulated annealing*), aunque sus experimentos estuvieron limitados a 30 trabajos y 10 nodos de ejecución, alejándose bastante de un caso real en el que se alcanzan con facilidad los 300 trabajos.

Phan *et al.* [9] presentan una solución donde el envío de trabajos y la replicación de datos en sistemas de área extensa son co–evolucionados mediante el uso de algoritmos genéticos. Se basan en la idea intuitiva de que replicando los datos y eliminando parte la necesidad de las comunicaciones entre procesos el *makespan* será menor. Su sistema logra *makespans* de entre un 20 % y un 45 % de los alcan-

zados mediante planificadores FIFO (*first in, first out*), pero asumiendo que no hay comunicación entre los trabajos, es decir, no hay trabajos paralelos.

Loukopoulos [10] realiza un interesante análisis de varios operadores genéticos especializados en ordenación para mejorar los resultados de un algoritmo genético usado para minimizar el *makespan* teniendo en cuenta una alta transferencia de datos en los trabajos. Un problema común de estas tres últimas propuestas es el considerar el *makespan* como criterio de planificación, ya que no siempre conduce a planificaciones óptimas, tal y como afirman Frachtenberg y Feitelson en [11].

Otros investigadores se han centrado en resolver las carencias de la planificación una vez que los trabajos están en ejecución. En esta línea se puede citar a Wu *et al.* [48], que desarrollan un análisis sobre la tolerancia a fallos en planificación mediante el uso de algoritmos genéticos. El trabajo está enfocado en la tolerancia a fallos por lo que usan un algoritmo genético simple para el estudio, en el que confirman la idea de que establecer puntos de control para continuar la ejecución de trabajos muy largos es preferible frente a los reintentos, que solo resultan más beneficiosos en trabajos cortos. Un problema ligeramente diferente ocurre cuando los nodos de un sistema no son exclusivos para un trabajo concreto, sino que compiten por la CPU. El problema se transforma, por tanto, en un problema de balanceo de carga. En este contexto, Correa y Melo [49] proponen una aproximación en la que un sistema clasificador genera reglas utilizando un algoritmo genético, reglas que se usan para decidir la migración de los trabajos entre los nodos de ejecución tratando de mantener la misma carga de proceso en cada uno de los nodos. Nikravan y Kashani [50] utilizan un algoritmo genético para resolver esto mismo pero intentando, al mismo tiempo, minimizar el *makespan*, la comunicación entre procesos y maximizar la utilización media por procesador. Las propuestas en esta línea atacan los síntomas del problema en lugar de intentar resolverlo. Además, proponen técnicas que no son fáciles de implementar ni son posibles en todos los casos, como pueden ser la migración o la pausa de trabajos para seguir su ejecución en el futuro.

En otra línea de investigación, se abarca el problema desde la óptica de sistemas con recursos heterogéneos y dinámicos. Un ejemplo, con ciertas similitudes con el problema tratado en esta tesis, es aquel en el que las tareas se dividen en subtareas con dependencias entre ellas. Este problema ha sido bien estudiado y se ha abordado en diversos ámbitos utilizando, también, algoritmos evolutivos. Algunos ejemplos de esto son [51] [52] y [53]. Recientemente, este mismo problema se ha extendido al contexto de los *grids*. Un ejemplo se trata en [54] donde se utiliza un algoritmo genético adaptado. Este trabajo en particular resulta muy interesante aunque los experimentos llevados a cabo en él cuentan con pocos trabajos, hasta un máximo de 35 y con 8 nodos de procesado. No obstante se maneja un número de tareas más elevado, del orden de 10000. Los autores no indican el tiempo necesitado para generar la planificación, solo el número de generaciones.

Relacionados con esta misma línea, existen trabajos en los que se utiliza evolución para planificación en computación *cloud* y sistemas voluntarios, con un escenario distinto pero directamente relacionado con el tema tratado en esta tesis. Estrada *et al.* [55] aplicaron un algoritmo genético paralelo para la planificación

adaptativa en entornos de computación voluntaria, entornos que cambian frecuentemente debido a los propios y también frecuentes cambios en la propia comunidad de voluntarios, lo que convierte la planificación en una tarea complicada y muy dinámica.

Jinhua Hu *et al.* [56] aplicaron un algoritmo genético para planificar los recursos de una máquina virtual en un entorno de computación *cloud* para tener en cuenta tanto la carga actual del sistema en cada momento como datos históricos y la carga del sistema tras aplicar las planificaciones, de forma que la carga se balancea mientras se trata de minimizar la migración de trabajos.

En [57], Gkoutioudi *et al.* proponen un algoritmo genético acelerado para la implementación de planificador evolutivo que tiene en cuenta diversos criterios como la ejecución completa de los trabajos y la seguridad, trabajando con conceptos como la tolerancia a fallos, la fiabilidad y la privacidad de la información en entornos *grid*. El uso de los algoritmos propuestos disminuye el tiempo de planificación evitando retrasos en los trabajos causados por algoritmos de planificación más lentos. Estos tres últimos trabajos resultan interesantes porque en esta tesis son un factor clave los frecuentes cambios, no solo en los usuarios, sino en los propios recursos, atendiendo a características como la carga de las máquinas o su disponibilidad.

En general, el trabajo realizado facilita la configuración de los gestores de colas y planificadores y demuestra que las aproximaciones evolutivas a la planificación son una herramienta adecuada para aliviar la configuración y para definir políticas de planificación de manera más natural. No obstante, las principales carencias que se encuentran son el ajuste de un conjunto de parámetros limitado, el intercambio entre conjuntos restringidos de políticas o el intento de optimizar medidas objetivas que no guían a buenas planificaciones en algunos casos o no representan fielmente las necesidades de los usuarios.

En este trabajo se aborda también este problema mediante técnicas evolutivas, lo que permite, además de facilitar la configuración, como ya se ha dicho, la definición de políticas que incorporen criterios subjetivos huyendo de la optimización de parámetros fácilmente mensurables como el *makespan*. Además, la solución propuesta busca ser adaptativa y dinámica de manera *on–line* para adaptarse a los cambios del conjunto de usuarios y de recursos lo más rápidamente posible para minimizar errores en la planificación, teniendo en cuenta la heterogeneidad de los recursos y la posibilidad de ejecución de trabajos paralelos. Finalmente, es importante no limitar el número de nodos de ejecución o trabajos a un número excesivamente bajo, pues como se ha dicho, estos sistemas tienen gran número de usuarios lo que genera importantes colas de espera. Para poder abordar el problema teniendo en cuenta las mencionadas características, hay que solucionar otro de los grandes problemas ya comentados en los sistemas de planificación: la imprecisión en las solicitudes de recursos. Para ello se propone en este trabajo el modelado del comportamiento de los usuarios durante su interacción con el sistema con el fin de predecir los recursos que realmente necesitarán sus trabajos. A continuación se recoge el estudio de las principales investigaciones en el campo del modelado de usuarios y comportamientos para después centrarse en la meta concreta de la predicción del uso de recursos.

3.4. Modelado de usuarios y comportamientos

Uno de los requisitos de la aproximación propuesta en esta tesis es la de adaptarse a cada uno de los usuarios de manera particular. El modelado de usuarios es importante y útil en aquellas aplicaciones y sistemas que necesitan cualquier tipo de adaptación al usuario. Para estudiar la forma de abordar este problema y las posibles alternativas, se ha realizado una revisión de los principales trabajos y sistemas de modelado de usuarios y comportamientos. Dicha revisión se puede agrupar por trabajos que se enfocan en lograr una ventaja concreta: los sistemas de ayuda al usuario, la detección de desvíos de conducta, contenidos personalizados o predicción de comportamiento. Huyendo de aplicaciones concretas del modelado de usuarios, existe una línea de investigación que se basa en el desarrollo de sistemas de modelado genérico de usuarios. A continuación se presentan algunos casos para las citadas aplicaciones y algunos sistemas de modelado genérico.

Un buen punto de partida, y como parte de los casos de sistemas de ayuda al usuario, es acudir a un ejemplo clásico de modelado, uno de los primeros intentos y por tanto pionero, y que ha resultado satisfactorio, para así entender los beneficios del modelado en un ámbito concreto. Se trata del sistema *WEST* presentado por Burton en [58], que es un sistema de entrenamiento para el juego llamado "How the West was Won", al estilo del juego de mesa "serpientes y escaleras"[6], en el que los jugadores dan vuelta a tres ruletas para obtener tres números con los que formar una expresión aritmética en base a los operadores (de uso único) "+", "−", "×" y "/" y los paréntesis adecuados, para así obtener el número que determina las posiciones que deben avanzar en el tablero de juego. El sistema WEST analizaba los movimientos de los estudiantes en términos de la estrategia óptima y los puntuaba en base a esta. El sistema vigilaba si los estudiantes seguían de manera consistente una estrategia no óptima y si detectaba este patrón de comportamiento podía intervenir en un momento adecuado e indicar, por ejemplo, cómo aprovechar mucho mejor los movimientos. Aunque el sistema WEST tiene en cuenta algunos de los conceptos básicos de modelado, lo hace en el contexto de un dominio muy simple, con lo que la aproximación funciona porque se pueden analizar todas las posibilidades de movimientos y los eventos del usuario son fáciles de interpretar. El modelo de usuario se construye incrementalmente por medio del uso de varios eventos que ocurren en el mismo dominio.

Un ejemplo más complejo es *INFOSCOPE*, que Fischer *et al.* presentan en [59], un sistema basado en conocimiento que ayuda a los usuarios a localizar información de interés en grandes fuentes de información mal estructuradas, en concreto en el dominio de los grupos de noticias dentro de la red *Usenet*. Por medio de agentes se capturan datos acerca del uso de esta información para construir modelos que ayudan a encontrar esta información relevante de manera más sencilla.

[6] *Serpientes y escaleras* es un antiguo juego de mesa indio considerado un juego clásico en todo el mundo. El juego consiste en un tablero con casillas numeradas y una serie de escaleras y serpientes que interconectan pares de casillas. El objetivo del juego es avanzar por medio de un dado o una ruleta una serie de casillas en cada turno desde la primera casilla hasta la última. La particularidad del juego reside en que aquellas casillas con escaleras hacen avanzar al jugador hasta una casilla posterior, mientras que las casillas con serpientes los hacen retroceder.

Como afirma Fischer [60], las aplicaciones de alta funcionalidad (AAF), como pueden ser los sistemas operativos, paquetes ofimáticos o clientes de servicios, son complejos porque necesitan cubrir las necesidades de grandes conjuntos de usuarios heterogéneos. Por tanto, el diseño de AAFs debe atender a tres problemas principales:

- Las funcionalidades no utilizadas no deben estar presentes.
- Las funcionalidades no conocidas por el usuario deben estar accesibles y ser mostradas cuando puedan ser necesarias.
- Las funcionalidades usadas comúnmente no deberían ser difíciles de aprender, utilizar y recordar.

Para abordarlos es necesario el estudio de patrones de uso, su estructura y sus mecanismos de ayuda y aprendizaje asociados. Los sistemas de ayuda activa (como *Activist* [61], *UNIX Consultant* [62] y *EUROHELP* [63]) fueron un intento temprano de analizar el comportamiento de los usuarios y de inferir objetivos de alto nivel partiendo de operaciones de bajo nivel. Activist es un sistema de ayuda activa para editores del tipo *EMACS* que ayudaba a los usuarios a aprender gradualmente acerca de las funcionalidades relevantes para sus tareas. De esta manera, Activist modela al usuario monitorizando las acciones que toman los usuarios y, contrariamente al sistema WEST, opera en un entorno abierto, creándose así el desafío de inferir los objetivos del usuario de las interacciones de bajo nivel.

Entre los trabajos orientados a la detección de desvíos de comportamiento, se puede destacar el de Fawcett [64], que describe un sistema de creación de modelos para la detección de conductas fraudulentas y lo aplica al ámbito de la telefonía móvil. En este trabajo se combinan técnicas de minería de datos con otras de aprendizaje máquina. En concreto, se usan técnicas de inducción constructiva junto con algunas otras más estándar para diseñar métodos de detección de este tipo de comportamientos en el uso de teléfonos móviles basándose en el perfilado de los clientes. Mediante la minería de datos se descubren indicadores de fraude mientras que mediante métodos inductivos se generan detectores de perfiles que utilizan estos indicadores. Un sistema de combinación de evidencias determina cómo manejar las señales de los detectores de perfiles para generar alarmas.

Otro ejemplo es el de Pepyne *et al.*, que presentan en [65] un sistema de perfilado de usuarios en el ámbito de la seguridad informática, en el que se hace uso de la teoría de colas y de métodos de regresión para perfilar a los usuarios basándose en aspectos temporales simples de su comportamiento. El objetivo es el de crear perfiles para grupos de usuarios muy especializados en los que por su naturaleza tienden a comportarse de manera muy similar y regular. El fin principal es el de detectar intrusiones y usos ilícitos de las cuentas de usuario de un sistema informático.

Otra aplicación de modelado, y que queda dentro del campo de la inteligencia ambiental, es la presentada por Lo *et al.* en [66], en donde se habla de equipamiento ubicuo de sensores para el cuidado de ancianos en sus hogares. Los cambios en la postura o en la forma de andar de personas con enfermedades crónicas son un posible indicativo de eventos no esperados (tales como caídas) o empeoramiento

de la enfermedad. La idea de este trabajo es modelar a un sujeto basándose en análisis postural extrayendo métricas personales del análisis de señales de vídeo obtenidas de cámaras inalámbricas situadas en el entorno de residencia del sujeto, donde el fin principal es el de detectar los citados cambios.

Alejándose de la idea presentada en estos tres últimos trabajos, y como ejemplo de sistema de generación de contenidos personalizados, Kuflik *et al.* [67] presentan un sistema multimedia de guía para los visitantes de un museo basado en modelado transparente para los mismos. Un sistema de estas características puede aumentar el nivel de satisfacción de los visitantes por medio de la oferta dirigida de contenidos y guías. En el trabajo al que se hace referencia se introduce el sistema *PEACH* por el que se lleva a cabo el modelado, que se usa para la presentación *on–line* de contenidos multimedia personalizados y que permite aportar servicios de valor añadido, como la generación automática de resúmenes e informes de la visita al museo. Dicho modelado, que se basa únicamente en la observación del comportamiento del visitante, tiene en cuenta información de este, tal como la posición espacial y su histórico (para saber si ya ha estado en un lugar determinado y si está en frente de una obra o solo cerca de ella) respecto de las exposiciones en cada momento, qué presentaciones han sido servidas al usuario y si ha mostrado interés en ellas, cosa que se deduce dado el carácter interactivo de las mismas.

El trabajo de Pentland *et al.* en [68] es un ejemplo interesante de sistema de predicción de comportamientos. Los autores afirman que el comportamiento humano puede ser descrito de manera precisa como un conjunto de modelos dinámicos, en este caso representados por medio de filtros de Kalman [69], secuenciados mediante cadenas de Markov. Los autores utilizan esos modelos de Markov para reconocer un comportamiento humano a partir de datos obtenidos de sensores y para predecir dicho comportamiento con una antelación de unos pocos segundos. Para demostrar su hipótesis, realizaron un experimento cuyo objetivo era la predicción de los movimientos subsiguientes de conductores de automóviles a partir de algunos movimientos iniciales de preparación. Para ello tuvieron en cuenta que cada comportamiento o acción puede ser dividido en pequeñas tareas sencillas (girar a la izquierda, frenar, cambiar de carril o comprobar que un carril está despejado, por ejemplo). En sus pruebas lograron un 95 % de precisión en la predicción de comportamientos.

La aplicación de técnicas de modelado de usuarios para la resolución *ad–hoc* de problemas concretos y ante la dificultad de extrapolar los resultados hicieron surgir una nueva línea de investigación en modelado genérico de usuarios. En este campo una referencia básica es [70], donde Kobsa revisa este tipo de sistemas desde veinte años antes de su publicación. Discute las propuestas y los servicios dentro de sistemas adaptativos al usuario y los diferentes requisitos y sistemas desarrollados, además de comentar diversas arquitecturas que han sido estudiadas y varios prototipos y sistemas comerciales. Como Kobsa explica en su trabajo, existen varios desarrollos de sistemas académicos de los que se pueden destacar los siguientes:

- *UMT* [71] permite al desarrollador la definición de estereotipos jerárquicamente ordenados y de reglas para realizar inferencias a partir de los modelos de usuario y detectar contradicciones.

- *BGP-MS* [72] permite asumir determinados estereotipos y hechos sobre el usuario para ser expresados como predicados de lógica de primer orden.
- *DOPPELGÄNGER* [73] es un servidor de modelado de usuarios que recibe información de sensores software y hardware que, mediante técnicas de extrapolación, son puestas a disposición de los desarrolladores de los modelos.
- *um* [74] es un conjunto de herramientas para modelado que representa asunciones acerca del conocimiento del usuario, sus creencias, preferencias, etc., en forma de pares atributo–valor. Cada subconjunto de información es acompañado por una lista de evidencias a favor o en contra.

Independientemente de los problemas concretos abordados en estos trabajos, existen múltiples aproximaciones para realizar perfilado o modelado de usuarios teniendo en cuenta técnicas como los estereotipos, lógica de primer orden, razonamiento clásico, razonamiento por inferencia, técnicas estadísticas, modelos *bayesianos*, etc. Partiendo de la base de que los datos que se pueden obtener de los usuarios son en parte subjetivos y suelen ser incompletos o imprecisos, el uso de técnicas que puedan trabajar con estas limitaciones resultan muy adecuadas. Un ejemplo son las redes de neuronas artificiales (RNA). Algunos estudios se han hecho dentro de este campo, como por ejemplo el que se cita en [75], que es una revisión de técnicas de modelado basadas en RNAs, donde se analizan y discuten actividades de modelado de usuarios e interfaces inteligentes. Entre las actividades que se comentan se pueden destacar las siguientes:

- Interacción hombre–máquina repetida. Los usuarios a menudo repiten tareas en su interacción con un equipo de escritorio, por ejemplo, leer nuevos *e–mails* nada más conectarse al sistema. La idea es tener un mecanismo que sea capaz de recuperar y aprender estas tareas o secuencias repetidas sin la intervención del usuario.
- Tareas de edición de texto. Mediante redes de neuronas se identifican o predicen estrategias del usuario durante la ejecución de tareas de edición de textos. En los experimentos realizados, con seis subobjetivos distintos de edición, se tenían en cuenta no solo comandos del editor (tales como borrar líneas o copiar y pegar) sino que también eran de relevancia las pausas tomadas por el usuario. Mediante una ventana de posibles comandos, las redes son capaces de determinar cuál es la salida apropiada, es decir, el subobjetivo a cumplir.
- Filtrado de mensajes. La idea es utilizar redes neuronales para aprender patrones de texto en los mensajes y así poder hacer un filtrado en, por ejemplo, un grupo de noticias en función del usuario y evitar de esta manera los mensajes no relevantes, es decir, aquellos que no pertenecen a los temas de interés para el usuario.
- Aprendizaje de intereses del usuario por medio de la observación de su comportamiento. Mucha investigación ha sido llevada a cabo sobre este tema, donde el principal fin es el de obtener un modelo de los intereses de un usuario en base a las acciones normales que este puede llevar a cabo, tales como

hacer *click* sobre un hipervínculo o no hacerlo o hacer *scroll* total o parcial de una página, donde, por ejemplo, una instancia de interés puede ser una página en la que el usuario pasa mucho tiempo y en la que además de hacer *scroll* total hasta el final de la misma, sigue el 80 % de los enlaces que en ella aparecen.

En este trabajo, el modelado de usuarios y comportamientos es importante por su relación con la predicción del uso de recursos. Se pretende modelar el comportamiento de los usuarios para predecir la cantidad de recursos de los que harán uso sus trabajos. La siguiente sección recoge una revisión de los principales trabajos en este tipo de predicciones.

3.5. Predicción de uso de recursos

Desde el punto de vista de los centros de computación de alto rendimiento, la interacción del usuario con el sistema es pobre en términos de acciones, que se limitan, principalmente, al envío de trabajos, consulta del estado de dichos trabajos y a la recogida de los resultados. Partiendo de esto, el modelado de usuarios en centros de HPC pasa fundamentalmente por modelar la forma en la que los usuarios hacen las solicitudes de los recursos para sus trabajos teniendo en cuenta el uso real de recursos. Esto, junto con un mecanismo adecuado, permitirá la predicción del uso de recursos de los trabajos. Como ya se dijo, resulta obvio que para lograr una buena planificación de trabajos es necesario contar con una buena estimación del uso que los trabajos van a hacer de los recursos. Entre estos recursos se pueden destacar el tiempo de CPU, el tiempo total del trabajo (denominado *walltime*), la cantidad de memoria o la cantidad de disco. En principio, son los propios usuarios y dueños de los trabajos los que conocen estas cantidades y, consecuentemente, los planificadores solicitan estas estimaciones a los usuarios cuando estos envían sus trabajos. Pero ya sea por equivocación, desconocimiento o simple despreocupación, motivos analizados en la sección 5.1, en muchos casos estas solicitudes no son tan precisas como cabría esperar. En trabajos como [76], [77] o [78] se estudia este hecho. Si los planificadores se basan en estas solicitudes para tomar sus decisiones, ¿cómo será la calidad de las planificaciones que se obtendrán? Algunas investigaciones han apuntado en esta dirección y estudian la predicción del uso real de los recursos con el fin de proveer a los planificadores con más y mejores parámetros en los que basar su operación.

A continuación se revisan algunos de los trabajos más relevantes relacionados con la predicción del uso de recursos. Se destacan tres líneas principales. La primera se centra en la predicción de los tiempos de espera de los trabajos en las colas. Otra tiene como principal objetivo la estimación de los tiempos de ejecución de los trabajos. Finalmente, y para ajustarse a las características de los entornos *grid*, donde los recursos son heterogéneos, los investigadores generalizan las predicciones para tener en cuenta no solo el tiempo de ejecución, sino también otras variables como la cantidad de RAM utilizada.

Como ejemplos de trabajos que tratan de predecir los tiempos de espera en las

colas se presentan los siguientes. En [79], Nurmi *et al.* proponen un sistema llamado *QBETS* en el que predicen límites superiores de espera para los trabajos. Su sistema se apoya en un estimador basado en inferencia no paramétrica y un sistema de clasificación de trabajos generado mediante aprendizaje máquina. Las pruebas las realizan sobre los ficheros históricos de 11 máquinas distintas de los últimos 9 años anteriores a la fecha de su estudio, probando que su método mejora los resultados de otras técnicas existentes.

Kumar *et al.* desarrollan en [80] y [81] un sistema llamado *PQStar* para la identificación de aquellos trabajos en las colas con esperas menores a una hora. Para realizar las predicciones utilizan características de los trabajos como el tiempo de CPU solicitado y del estado del sistema como la ocupación de los nodos de proceso. La idea principal es que la identificación de dichos trabajos permite la mejora global de los tiempos de espera y proporciona información importante al planificador.

Siguiendo con esta línea de investigación y adentrándose en otra, algunos investigadores se centran en estimar los tiempos de ejecución de los trabajos. Para predecir el instante de inicio de los trabajos en un *cluster*, así como sus tiempos de ejecución, una propuesta es presentada en [13] y [82] por Li *et al.*, en el que los autores consideran estas predicciones útiles para la asignación de recursos y el balanceo de carga de trabajo. Sus predicciones se basan en análisis estadístico de datos históricos, técnicas de aprendizaje local y en simulaciones de planificación y mejoran aquellas realizadas por el sistema del *European Data-Grid*, el cluster de producción del NIKHEF [83], el instituto nacional para física subatómica de los Países Bajos. El principal problema de estas tres investigaciones es que, aunque sí resulte de interés predecir los tiempos de espera de los trabajos, podrían realizarse planificaciones más precisas conociendo además el tiempo de ejecución y, en general, el uso de recursos.

Smith *et al.* presentan en [84] una técnica para predecir el tiempo de ejecución de trabajos paralelos basándose en otros trabajos de iguales características ejecutados en el pasado. Mediante técnicas de búsqueda voraces y algoritmos genéticos determinan plantillas para decidir la similitud entre trabajos y poder realizar las predicciones mediante métodos estadísticos. En [12], los mismos autores aplican su sistema de predicción para predecir el tiempo en que los trabajos permanecerán en la cola y para mejorar políticas de planificación basadas en *backfilling* y *primero el más corto*. El problema del uso de plantillas es la generalización que se hace en la predicción de los trabajos debido a la necesidad de clasificación.

En [85] y en [86], Tang *et al.* proponen diversos métodos para realizar estimaciones de las duraciones de los trabajos para mejorar la planificación en sistemas *Blue Gene/P*. Su idea es descubrir cómo las imprecisiones en las estimaciones afectan a los diferentes elementos de las políticas de planificación en colas con prioridad utilizando un esquema de *backfilling*. Este trabajo resulta interesante porque mejoran las estimaciones realizadas por los usuarios y además se optimizan las planificaciones, no obstante, el modelado no se realiza de manera *on–line* para ajustarse dinámicamente al comportamiento de los usuarios y, además, la predicción se basa en la búsqueda de trabajos similares en el histórico, lo que implica realizar una cierta clasificación.

Chen *et al.* presentan en [87] un sistema que utiliza ficheros históricos para predecir el tiempo de ejecución de trabajos apoyándose en modelos ocultos de Markov y análisis de frecuencias. Los autores afirman conseguir un error menor a 200 segundos en el 75 % de los casos. El principal problema en su propuesta radica en que se modela el progreso del trabajo para hacer la estimación, cosa que implica que los trabajos tengan que haber empezado.

En general, en esta línea, aunque se consiguen resultados más o menos precisos, el hecho de considerar solo el tiempo de ejecución de los trabajos limita su aplicación en entornos con recursos heterogéneos. Yendo un paso más allá, en otra línea sí se tiene cuenta este aspecto. Se presentan a continuación algunos ejemplo de trabajos.

Piro *et al.* [14] presentan un sistema para predecir el uso real de recursos en un *grid* utilizando información histórica. No obstante, no se limitan a la predicción de los tiempos de inicio o los tiempos de ejecución como se hace en trabajos previos, sino que estudian la predicción de otros tipos de recursos como la cantidad de memoria o de almacenamiento utilizado por los trabajos por medio de análisis estadístico. Para realizar dichas predicciones los autores clasifican previamente los trabajos apoyándose en plantillas de similitud como se hacía en [84], adoleciendo, por tanto, del mismo problema de generalización debido a la clasificación.

Sonmez *et al.* realizan en [15] un estudio para la predicción de los tiempos de ejecución y de espera en colas para entornos *grid*. Su sistema se basa en tres aspectos: la clasificación de trabajos por aplicación, usuario y *grid*, ficheros históricos con datos de ejecución y simulaciones de ejecución. Para las predicciones se utilizan series temporales. Por otro lado, los autores investigaron la incidencia de los tiempos de espera en el rendimiento de los *grids* al planificar con y sin la estimación del tiempo de ejecución proporcionada por su sistema. Las principales conclusiones extraídas fueron que los algoritmos de planificación y estimación existentes hasta la fecha no son aptos para entornos *grid*, ya que los *clusters* tradicionales poseen distintas características. Este trabajo resulta muy interesante por la separación que se hace por usuarios, permitiendo de alguna manera, capturar el comportamiento de los mismos.

Recientemente, Banalagay *et al.* [16] presentan un método para predecir el uso de recursos, en especial los de tiempo y memoria, en trabajos de análisis de imágenes médicas y prestando especial atención al hecho de que los tiempos varían mucho en función de los recursos en los que se ejecuten dichos trabajos. Aunque proporcionan muy buenos resultados, la solución está limitada a un tipo de trabajo muy concreto y no se especifica si es extrapolable a otros tipos.

Los trabajos en esta línea, aunque sí tienen en cuenta que los datos históricos muy antiguos no son relevantes, no consideran el hecho del comportamiento cambiante de los usuarios para producir sistemas de estimación actualizados de manera *on–line* para realizar predicciones ajustadas en todo momento.

Junto con la revisión presentada en la sección anterior, esta revisión sirve como punto de partida para abordar, en esta tesis, un sistema de predicción de uso de los diferentes recursos desde la perspectiva del modelado del comportamiento

del usuario en su interacción con el sistema. Se busca particularizar el sistema a cada uno de los usuarios por medio de la generación de modelos individuales de cada usuario que se mantengan actualizados según nueva información se vuelve disponible en el sistema. La principal fuente de información para la construcción y actualización de estos modelos son los históricos de ejecución de trabajos que, como se ha visto, es un punto en común en todas las investigaciones revisadas.

3.6. Satisfacción de los usuarios

Otro de los problemas en la planificación de trabajos en entornos HPC, y como se ha visto en la sección 3.3, es que en las políticas de planificación se tienen en cuenta medidas objetivas y fácilmente mensurables que difícilmente representan de manera fiel, o al menos cercana, lo que los usuarios esperan del servicio que el centro les proporciona. Lo habitual en el mundo de la supercomputación es que los recursos disponibles no sean suficientes para atender inmediatamente todas las solicitudes, que unos trabajos tengan más importancia que otros (y, por tanto, haya que asignarles una prioridad mayor) y que los recursos necesitados sean muy dispares. En concreto, el uso de distinto número de procesadores por las tareas gestionadas por el planificador es un problema importante. Si un sistema dispone de x procesadores libres y el siguiente trabajo por orden de llegada necesita y procesadores, siendo $y > x$, pero un trabajo posterior solamente necesita z procesadores, donde $z < x$, existe una clara divergencia entre la optimización de los recursos del sistema, que indicaría que el último trabajo se debe ejecutar inmediatamente, y la satisfacción de todos los usuarios, ya que el usuario del trabajo que necesita y procesadores sufrirá un agravio comparativo. Agravio que será mayor si en vez de un trabajo que requiere menos de x procesadores tenemos muchos, pudiéndose dar el caso de que el trabajo conflictivo nunca se ejecute. Existe, por tanto, una disparidad de intereses entre la optimización del uso de los recursos, que llevaría a atender siempre a los trabajos que pudiesen ser comenzados inmediatamente, y la satisfacción de los propios usuarios, que viene determinada principalmente por el tiempo que estos deben esperar hasta obtener sus resultados. Por tanto, no se puede tardar un tiempo desproporcionado en atender a unos usuarios en aras de optimizar los recursos del sistema, se deberá llegar a un equilibrio entre ambos requisitos. En este trabajo se propone la incorporación de medidas subjetivas en las políticas de planificación con el fin de mantener un alto nivel de satisfacción general tanto en el centro como en el conjunto de usuarios. El punto de partida es hacer una revisión de los trabajos que intentan medir y predecir la satisfacción de los usuarios en sistemas informáticos.

Muchos sistemas, informáticos o no, realizan asunciones en su diseño acerca de lo que persigue el usuario y de su interacción con él. Muchos otros hacen lo mismo pero permiten una cierta configuración, más o menos compleja, que hace que el usuario se sienta más cómodo con el sistema y sus resultados. La mayoría de sistemas caen en una de estos dos clases. El principal problema se halla en que estas asunciones no dejan de ser generalizaciones, cosa que lleva, debido a la naturaleza humana, a grandes desviaciones respecto de la realidad, en la que cada

usuario es diferente y tiene sus gustos, preferencias y circunstancias. No obstante, algunos sistemas sí tienen en cuenta estos aspectos, mediante el uso de mecanismos que modelen al usuario como los presentados previamente, pero en general siguen haciendo asunciones en mayor o menor medida. Un concepto con gran relevancia para esta tesis es el de la satisfacción. La satisfacción es un sentimiento y por tanto, como parámetro, resulta poco medible. La manera más precisa de hacerlo, aunque resulte una intrusión, es preguntarle directamente al usuario por su nivel de satisfacción respecto de su uso del sistema.

Algunas investigaciones se han hecho en el campo de la satisfacción y su consideración en sistemas informáticos. Una de las primeras y fundamentales referencias es la de Bailey y Pearson [88], del año 1983, que recoge el desarrollo de una herramienta para analizar y medir la satisfacción de usuarios de ordenador para la que se identifican 39 factores que afectan a la satisfacción y, posteriormente, se genera un cuestionario para medir esa satisfacción. El trabajo está enfocado a sistemas de información donde lo importante es el acceso a ella, así como su calidad. Lo que se pretende es poder incrementar la productividad de este tipo de sistemas, cuya utilización está directamente ligada a la satisfacción asociada a la percepción que los usuarios tienen del servicio. En el trabajo se define la satisfacción en términos psicológicos como una suma ponderada de las reacciones, positivas o negativas, de los usuarios a los diversos factores identificados, entre los que se pueden destacar los siguientes: relevancia, actualidad, formato, idioma, errores en la recuperación, volumen, etc. El cuestionario se basa en puntuar cada uno de los diferentes factores a través de cuatro pares de adjetivos que los describen, lo que permite calcular la importancia de los factores para cada usuario. Los resultados arrojados por los cuestionarios pueden ser, entonces, estudiados y utilizados como instrumento para el análisis del sistema.

Ese mismo año, Ives *et al.* presentan en [89] otro importante trabajo, un estudio comparativo de cuatro técnicas de medición de satisfacción, incluyendo entre ellas la de Bailey y Pearson, valorándola positivamente sobre las demás y validándola. No obstante, los autores identifican determinados problemas, como la existencia de factores, que ellos llaman escalas, no necesarios y la excesiva longitud del cuestionario, para lo que proponen como solución un conjunto de factores reducido y con menos elementos para evaluar por factor, logrando así una técnica de medición mejorada. Muchos trabajos desde entonces han sido enfocados en este sentido sentando las bases para medir la satisfacción de los usuarios. Entre ellos se puede destacar el de Doll *et al.* en [90], en el que se propone otro instrumento para la medida de la satisfacción. A través de encuestas se determinan cinco factores fundamentales que derivan en la creación de doce elementos que son los que se tendrán en cuenta para medir la satisfacción. La principal diferencia con trabajos anteriores radica en la posibilidad de evaluar cualquier aplicación informática en lugar del típico tratamiento de datos existente hasta la fecha. Esta generalización abrió nuevas vías en la investigación de la medida de la satisfacción.

Posteriormente, Gatian se pregunta si la satisfacción de los usuarios es una medida adecuada del rendimiento y eficacia de un sistema. En su trabajo [91] afirma que la satisfacción de los usuarios es utilizada a menudo como un medida sustituta de la eficacia de un sistema de información, que si es considerado un sistema eficaz

debe repercutir de manera positiva en el comportamiento de los usuarios, como por ejemplo, incrementar su productividad. En su trabajo buscan la relación entre la satisfacción y el comportamiento de los usuarios de un sistema de información en 39 organizaciones diferentes, concluyendo que estas relaciones existen.

Por otro lado, Gelderman demuestra empíricamente en su estudio [92] que la satisfacción y el comportamiento del usuario están significativamente relacionadas y que la relación entre el uso del sistema y su rendimiento es despreciable, de forma que quedan proporcionadas evidencias en favor de la asunción popular de que la medida más apropiada del éxito de los sistemas de información es la satisfacción de los usuarios. No obstante, todavía se utilizan formularios muy extensos para medir la satisfacción, demostrando que otros de menor dimensión arrojan resultados demasiado imprecisos.

Una vez demostrado y aceptado el hecho de que la satisfacción de los usuarios es una medida fiable del éxito y rendimiento de un sistema informático y proporcionados instrumentos generales para realizar dichas medidas, surge la necesidad de evitar la gran intrusión que supone el hacer que muchos usuarios completen largas y tediosas encuestas o sean entrevistados o directamente observados.

Más recientemente, y tomando como base la teoría de que el comportamiento del usuario, al estar relacionado directamente con la satisfacción, puede determinar a esta, Downing presenta en [93] un estudio empírico en el que se utiliza el análisis de datos históricos del comportamiento de los usuarios para intentar medir la satisfacción de estos con el sistema y lo valida mediante la aplicación simultánea del cuestionario de satisfacción tradicional propuesto por Doll. En el trabajo se estudia un sistema interactivo de respuesta por voz[7] (*Interactive Voice Response System*, IVRS) para proporcionar información sobre el plan de jubilaciones a unos diez mil empleados de la empresa *Savings Express*. El sistema monitorizó y almacenó el comportamiento de los usuarios, básicamente pulsaciones de botones de teléfono, durante seis meses de manera continuada para que, después, un conjunto de reglas pudieran extraer información sobre los mismos parámetros utilizados en las encuestas. Las reglas fueron creadas en colaboración con un grupo de expertos que establecían la relación entre las pulsaciones y los valores de los parámetros. Los buenos resultados arrojados por el trabajo abren una nueva vía de investigación en sistemas de medida de satisfacción sin intrusión.

Apuntando en otras direcciones, algunos autores dirigen sus esfuerzos a generar instrumentos de medida basados en encuestas para la realidad actual de internet y la *world wide web* o la computación general ubicua, orientándose incluso hacia sistemas de redes sociales, y otros trabajan en sistemas capaces de medir o estimar sin intrusión la satisfacción. Entre los primeros se pueden destacar los trabajos de Aladwani y Palvia [94], que estudia el desarrollo de un cuestionario de veinticinco factores para medir la satisfacción de usuarios de aplicaciones *web*, o el de Sullivan *et al.* [95], en el que se estudian los factores clásicos que inciden en la satisfacción en sistemas tradicionales para adaptarlos al contexto de los proveedores de contenido con características sociales. Entre los segundos se puede considerar, por

[7] Se trata del típico sistema telefónico con preguntas y respuestas grabadas que guían al usuario para la consecución de un fin determinado.

ejemplo, el trabajo de Chen *et al.* [96] en el que se desarrolla un modelo para medir la satisfacción de los usuarios del célebre sistema *Skype* pero generalizable a otros sistemas de voz sobre *Internet Protocol* (IP) o *VoIP*. El modelo propuesto por los autores se basa únicamente en medidas de red, no utilizando en absoluto análisis de señales de habla o encuestas a los usuarios, en el que, además de factores tales como la degradación de la señal se capturan otros como repeticiones al hablar o retrasos en la conversación, aunque está principalmente basada en la duración de las llamadas. Los resultados del trabajo demuestran que la satisfacción está directamente ligada a estos factores y que existe una fuerte correlación entre la duración de las llamadas y la interacción de los usuarios. Además, aparte de ser directamente implementable en las aplicaciones, permiten un control adicional para evaluar el rendimiento y la calidad del servicio (QoS).

De todas formas, los trabajos que pretenden obtener medidas de satisfacción sin intrusión son una extrema minoría y difíciles de encontrar entre la extensa literatura, que se centra en hallar, crear y mejorar factores y cuestionarios de satisfacción en diversos ámbitos. El motivo de esto, como ya se ha dicho, es la dificultad de medir factores subjetivos que, posiblemente, vengan determinados por muchos aspectos.

A la vista de estos trabajos se pueden extraer diferentes conclusiones. La medida de la satisfacción tiene gran relevancia como indicador del éxito de un sistema. Los investigadores han desarrollado, y siguen, a lo largo de los años cuestionarios que sirven para medir la satisfacción de usuarios y clientes, cuestionarios basados en factores previamente determinados. No obstante, esta manera de obtener medidas de satisfacción se considera de alta intrusión pues suele requerir varios minutos, máxime si se pretenden obtener medidas de manera continuada en entornos altamente cambiantes.

No se han encontrado trabajos relevantes de medida y uso de la satisfacción en centros de computación de alto rendimiento ni, en general, trabajos en los que se obtengan medidas de satisfacción para variar el comportamiento del sistema de manera dinámica con el fin de mantener un grado mínimo de satisfacción entre sus diversos usuarios e, incluso, mejorarlo.

Por último, cabe destacar que la mayoría de estos trabajos relacionan directamente la satisfacción de los usuarios con la utilidad y rendimiento del sistema pero considerando que la satisfacción se genera por igual en todos los usuarios cuando esto no es necesariamente cierto. De esta manera, no se hace diferencia entre usuarios ni se particulariza para mejorar los sistemas teniendo en cuenta las características propias e individuales de los usuarios.

Como se dijo, esta tesis propone un sistema de medida y predicción de satisfacción particularizado a los usuarios que puede ser utilizado en las políticas de planificación de un centro de HPC. Junto con el planificador propuesto y el sistema de predicción de recursos solo queda salvar el último de los escollos, que es el modelado de recursos, que permitirá mejorar las planificaciones en función de las predicciones de estado de los recursos, la inclusión de criterios en las políticas de planificación que estén directamente relacionados con la satisfacción del centro, como el coste o el gasto energético y, en general, la adaptación del sistema propuesto para su funcionamiento en entornos heterogéneos de tipo *grid*.

3.7. Modelado de recursos

En el momento en el que en un sistema de computación en paralelo se permiten recursos heterogéneos, que es el caso general de los centros de HPC, hay un claro beneficio en lo que a flexibilidad, facilidad de ampliación y escalado del sistema se refiere. No obstante surgen dos nuevos problemas:

- En entornos tipo *grid*, no se puede asumir que la disponibilidad de los recursos sea del 100 %. Tampoco se puede hacer la asunción de que los procesadores disponibles no tengan ningún otro tipo de carga. Esto implica que la dificultad inherente a la solicitud de recursos crece, pues al no saber en qué tipo de recurso puede llegar a ejecutarse un trabajo, el usuario tampoco sabrá qué cantidad concreta necesita, sobre todo en lo que a tiempo de CPU se refiere.
- El sistema de modelado de usuarios y predicción de recursos, tampoco puede asumir que todos los recursos tienen las mismas características. Es necesario, por tanto, realizar una modificación para poder soportar esta nueva características y que los modelos de usuario, así como las predicciones de recursos funcionen de manera coherente con independencia de las características de los recursos.

En esta tesis se propone una solución que permite modelar los recursos de un sistema para poder tenerlos en cuenta para:

1. Realizar una mejor predicción del uso de recursos por parte de los trabajos por medio de la estimación del estado de los recursos en el futuro, modelando características como la carga de CPU o la disponibilidad.
2. Realizar una caracterización de recursos de distinta potencia y capacidad. Esto hace adecuado al sistema de modelado de recursos para su uso con tecnologías *grid*.
3. Permitir a los administradores del centro de HPC incluir en las políticas de planificación criterios directamente asociados con sus intereses como el coste o el gasto energético, buscando así satisfacer al propio centro.

A continuación se hace una revisión de trabajos que tratan de resolver el problema de modelado de recursos heterogéneos. En la revisión se pueden agrupar las investigaciones en tres categorías. Una primera se centra en el perfilado de recursos para la estimación de su estado y del uso real de recursos por parte de los trabajos. Otra línea se enfoca en aplicar el modelado de los recursos para realizar planificaciones más precisas. Finalmente y un poco al margen de las otras, algunos investigadores han dirigido sus esfuerzos hacia el desarrollo de herramientas que permiten la simulación de entornos heterogéneos que sirven para el estudio de dichos entornos y que permiten simular planificaciones y modelado de recursos en mayor o menor medida.

Como ejemplos de la primera línea de investigación, el perfilado de los recursos, se comentan a continuación algunos trabajos significativos. En [97], Dinda *et*

al. presentan un detallado estudio estadístico para la predicción de la carga de CPU media, demostrando que es un parámetro medible y predecible. El principal problema es que los autores realizan predicciones puntuales en intervalos de tiempo muy pequeños de hasta 30 segundos.

Wolski estudia en [98] diversos métodos estadísticos para la predicción del rendimiento de recursos en entornos *grid* orientados a ser utilizados en planificadores adaptativos así como para predecir fallos de manera *on–line*. Este trabajo prueba cómo las técnicas estadísticas resultan adecuadas al problema del perfilado de recursos.

EMPEROR [99] es un meta–planificador para la arquitectura OGSA (*Open Grid Services Arquitecture* [100]) desarrollado por Adzigogov *et al.* Su propuesta se basa en el uso de modelos autorregresivos para la predicción dinámica de recursos, en concreto de la carga de CPU y la memoria, para poder, consecuentemente, estimar el tiempo de ejecución de los trabajos. No obstante, el principal problema de este meta–planificador es que no resuelve la planificación en los sistemas finales, que deben ejecutar su propio planificador con los trabajos recibidos del primero.

Nadeem *et al.* [101] proponen un sistema para caracterizar los recursos teniendo en cuenta sus políticas de disponibilidad y cuantificando su comportamiento con la ayuda de modelos de disponibilidad. Su sistema explota estas caracterizaciones para realizar predicciones de disponibilidad a través de reconocimiento de patrones y clasificación. Aunque este trabajo resulta muy interesante, los autores se centran solo en disponibilidad y no en otras propiedades, como puede ser la carga de CPU.

Entrando en la segunda de las líneas de investigación comentada, algunos trabajos aplican directamente el modelado de recursos en tareas de planificación con el fin de mejorarla. Un ejemplo de uno de estos trabajos es [102]. En él, Elmroth y Tordsson presentan un sistema para la asignación de recursos y planificación en entornos *grid* para minimizar el tiempo de servicio total, que incluye la espera del trabajo en cola, el propio tiempo de ejecución del trabajo y el de transferencia del ejecutable y el flujo de entrada y salida a y desde el recurso. Para la determinación de los recursos a usar se basan en resultados de ejecuciones de *benchmarks* de los diferentes recursos así como en predicciones del estado de la red. El principal problema es que parte de sus algoritmos se basan en reservas de recursos con su consiguiente bloqueo.

En [103], Reig *et al.* proponen un sistema de predicción de uso de recursos en entornos heterogéneos para planificadores *deadline*[8] mediante clasificadores y aprendizaje máquina. El principal problema radica en que no tienen en cuenta la disponibilidad y carga de los recursos en el tiempo y que se asumen trabajos no paralelos.

Ramachandran *et al.* [104] estudian la predicción de la disponibilidad de recursos para poder asignar aquellos con mayor probabilidad de disponibilidad con el fin de que los trabajos no se vean interrumpidos. Su investigación se centra principalmente en redes P2P (*Peer–to–Peer*) donde la gestión de recursos es distribuida.

[8] En los planificadores *deadline* se debe garantizar un instante de comienzo determinado para cada trabajo.

Aunque los autores optimizan el sistema de planificación por medio de la asignación del mejor recurso, el modelado de recursos para predicción se hace mediante agrupaciones de múltiples recursos con patrones de uso similares, haciendo que se pierdan las particularidades de cada recurso.

Un poco al margen de los trabajos previos, algunos autores han desarrollado sistemas que sirven como base al estudio de entornos con recursos heterogéneos, principalmente simuladores. Aunque se han desarrollado varios simuladores de *grid* para la ejecución de experimentos y el estudio previo a la puesta en producción, como el *SimGrid* de Casanova *et al.*[105], no suelen tener en cuenta, y si lo hacen no proporcionan mecanismos para resolverlo, el problema de la disponibilidad de los recursos. Un trabajo digno de mención es [106], donde Buyya y Murshed desarrollan, en un completo trabajo, una herramienta llamada *GridSim Toolkit* para simular la gestión de recursos distribuidos y heterogéneos y la planificación en entornos *grid* que permite el modelado de los recursos. Los autores tienen en cuenta los retos que estos entornos conllevan, que son los continuos cambios en los recursos, la disparidad entre las políticas de los diferentes dueños o administradores de dichos recursos, la tolerancia a fallos, etc. No obstante, los problemas de la compleja configuración de los gestores de colas se trasladan al simulador y, aunque resulte muy útil para el estudio, no proporciona soluciones para poder tener en cuenta los problemas comentados.

En esta tesis se aborda el problema de la planificación en entornos heterogéneos donde, para poder introducir criterios como la satisfacción de los usuarios y del centro en las políticas de satisfacción, es necesario optimizar el sistema en varios puntos. Uno de estos puntos es el tratamiento de los diferentes recursos. Para realizar planificaciones precisas es necesario estimar el uso real de recursos por parte de los trabajos. Para poder realizar esta estimación, así como para poder planificar de manera adecuada, es muy interesante poder predecir el estado de los recursos en el futuro. Para esto hay que tratar el problema del modelado dinámico e individual de recursos. Y no solo desde el punto de vista de la disponibilidad, carga de CPU u ocupación de memoria, sino que también hay que considerar otros parámetros como pueden ser el coste económico o la energía utilizada, pues este tipo de parámetros tienen mucho que ver con la satisfacción de los propios centros de HPC.

3.8. Discusión

En general, el trabajo realizado hasta la fecha trata de cumplir alguna política de utilización del sistema que tenga en cuenta implícitamente un equilibrio entre utilización de recursos y satisfacción del usuario. En las investigaciones desarrolladas se utilizan técnicas estadísticas y de inteligencia artificial para solucionar parte de los problemas de la planificación de procesos. Sin embargo, todos ellos asumen que la satisfacción del usuario consiste en minimizar su tiempo de espera por los resultados o alguna medida concreta y generalizada similar. Por otro lado, muchos trabajos se centran en optimizar una parte concreta de un sistema haciendo diversas asunciones sobre las demás, que de no ser ciertas, limitan mucho el alcance de dichos trabajos.

En este trabajo se presenta un sistema de planificación de colas de procesos para computadores de alto rendimiento que satisface diversos objetivos. Para ello se pretenden integrar diversas mejoras, algunas de las cuales han sido comentadas en la revisión previa, pero otras son novedosas. Además, se busca la aplicación de técnicas como el modelado de usuarios y comportamientos para adaptarse al usuario. En concreto, dichos objetivos son:

- Evitar las frecuentes reconfiguraciones del planificador debido a cambios en el comportamiento de los usuarios y los recursos y facilitar la tarea de especificar la política de planificación a los administradores del centro de HPC. Además, esto evita la necesidad de acogerse a una política en concreto que pueda distar bastante de ser la óptima para determinados estados de la cola de trabajos.

- Realizar de manera autónoma y transparente modelos de usuario que permitan optimizar las solicitudes de recursos. Esto proporcionará al planificador información más precisa para la toma de decisiones de planificación. Se busca además una adaptación particularizada a cada uno de los usuarios huyendo de categorizaciones y generalizaciones.

- Tener en cuenta la subjetividad del usuario materializada en el concepto de satisfacción, evitando hacer asunciones sobre lo que quiere el usuario y no dando por sentado que todos ellos son iguales. Además, se pretende evitar la generalización causada por un proceso clásico de identificación de necesidades en el diseño del sistema. Todo esto minimizando la intrusión por parte del sistema para hacer la interacción del usuario con el sistema lo más agradable y sencilla posible.

- Monitorizar los recursos con el fin de extraer modelos de su comportamiento que permitan realizar predicciones acerca de su disponibilidad, carga de CPU, de memoria, coste, gasto energético, etc., para mejorar la precisión de las planificaciones y facilitar la tarea de definición de políticas de planificación.

Al combinarlos entre sí, estos objetivos dan lugar al sistema presentado seguidamente, bautizado como *EvoProc*, nombre que cobra forma partiendo de los términos *evo*lución y *proc*esos. Este sistema surge de una colaboración con el Centro de Supercomputación de Galicia en el marco de una proyecto de investigación subvencionado por la Xunta de Galicia, denominado “Arquitectura de agentes autónomos para el modelado de usuarios y la gestión óptima de un centro de supercomputación”. EvoProc, además de lidiar de forma innovadora y directamente con los problemas comentados en esta sección existentes en trabajos anteriores, supone una aproximación novedosa por la integración de cada uno de estos cuatro objetivos en una única solución.

Capítulo 4

Planificación evolutiva

Si en el camino cambian tus prioridades, tal vez debas cambiar también de camino.

A la ya de por sí complicada tarea de configurar un planificador en un centro de computación de alto rendimiento, debido al alto número de parámetros que manejan, se suma un factor que no ayuda en absoluto: el cambio continuo, tanto en el conjunto de usuarios como en el comportamiento de los mismos. Esto hace que los planificadores y políticas configurados pierdan eficacia y necesiten una nueva reconfiguración. Este capítulo se centra en la propuesta del desarrollo de un planificador de procesos evolutivo que elimine, o al menos alivie, la necesidad de las frecuentes reconfiguraciones a la vez que hace más sencilla esta tarea. Además, un planificador así permite la incorporación de políticas de planificación definidas de manera más natural y contribuirá a facilitar, por tanto, la incorporación de medidas que tengan en cuenta la subjetividad del usuario, en concreto, la satisfacción. A continuación se explicarán los conceptos fundamentales que sustentarán el diseño del planificador propuesto.

4.1. Preámbulo

Se podría definir el término *planificación* como la acción y efecto de planificar, es decir, el desarrollo de un plan metódicamente organizado para la consecución de un fin concreto[1].

De manera general, la planificación es una operación que consiste en encontrar un orden adecuado para abordar las tareas necesarias para la consecución del fin deseado. Estas tareas serán servidas por un grupo de recursos de diversa índole y generalmente limitado. Este modelo se puede complicar tanto como se quiera por

[1] Basado en la definición que la Real Academia Española hace del vocablo.

medio de la imposición de restricciones que pueden pasar por dependencias temporales entre tareas o disponibilidad de recursos variable, entre otras. No obstante, en el ámbito de los supercomputadores se introducen factores que alteran levemente, aunque de manera significativa, el concepto tradicional de planificación. La principal idea es que el proyecto a planificar no acaba ya que jamás dejan de llegar nuevas tareas, que son los trabajos que envían los usuarios. Esto obliga a realizar una replanificación completa ya que no se puede asumir que poner un nuevo trabajo al final de una cola para su ejecución sea la mejor opción, aunque esto dependerá de la política de planificación elegida. Por otro lado, la variabilidad inherente de los usuarios, los trabajos y las necesidades obligan a reajustar o, incluso, replantearse periódicamente dichas políticas para lograr resultados satisfactorios, pues esta obsolescencia que se adquiere gradualmente causa desvíos cada vez mayores en los objetivos globales perseguidos por el centro de HPC.

Hay que detenerse un poco más en todos los conceptos presentados en esta breve introducción para poder comprender las ventajas que brinda un planificador evolutivo y para poder seguir el diseño que de él se hace en este trabajo. Eso es exactamente lo que se pretende con los apartados que siguen.

4.1.1. El problema de la planificación

El problema de la planificación, tal y como se entiende en este trabajo, se puede ver como un caso modificado del famoso problema del viajante o *traveling salesman problem* (TSP), o bien del *job shop problem* (JSP) o el *knapsack problem*.

El TSP es un problema, de entre los más importantes de combinatoria, que representa y simboliza un tipo muy concreto de problemas de fácil solución teórica pero difícil en la práctica, pues son problemas NP–completos. Definido originalmente por el matemático irlandés W. R. Hamilton y el matemático inglés Thomas Kirkman en el siglo XIX [107], su enunciado dice así: para N ciudades dadas, el objetivo es encontrar una ruta que comience y termine respectivamente en dos ciudades concretas y que pase una sola vez por cada ciudad minimizando la distancia recorrida. Formalmente, el problema consiste en encontrar una permutación $P = \{c_0, c_1, ..., c_{N-1}\}$ tal que $d_P = \sum_{i=0}^{N-1} d[c_i, c_{i+1}]$ sea mínimo. La distancia entre ciudades viene dada por una matriz $D : N \times N$ donde $d[x, y]$ representa la distancia entre las ciudades X e Y.

La aplicación de técnicas de fuerza bruta para la resolución de este tipo de problemas es satisfactoria en casos en los que el número de ciudades es extremadamente pequeño, en cambio, cuando este número crece moderadamente, resulta inviable. Por ejemplo, para 20 ciudades, la cantidad total de posibles permutaciones asciende por encima de $2,4 \cdot 10^{18}$, es decir, más de 2,4 trillones.

Se han intentado aplicar diversas técnicas de optimización para resolución del TSP logrando solo aproximar de manera aceptable el resultado. Por ejemplo, los algoritmos evolutivos permiten encontrar soluciones razonablemente buenas para redes con millones de nodos utilizando para ello tiempos asumibles (semanas o incluso meses) en un supercomputador.

Pero el TSP no es un problema isomorfo al de la planificación de trabajos en un supercomputador. Considerar un caso en el que los trabajos de una cola de espera son las ciudades a recorrer y en el que un único procesador o unidad de proceso es el viajante es quedarse corto, ya que el problema aquí tratado pretende el uso de múltiples procesadores y no de un único recurso.

El objetivo del *job shop problem* es el de encontrar el orden adecuado en el que M máquinas puedan servir N trabajos minimizando el tiempo total, el llamado *makespan*. En la formulación más simple de este problema se considera que cada trabajo está compuesto de S subtareas, cada una de las cuales se lleva a cabo en cada una de las máquinas, que son idénticas y solo pueden servir una subtarea a la vez. De manera formal, sean M un conjunto de máquinas $M = M_1, M_2, ..., M_m$ y J un conjunto de trabajos $J = J_1, J_2, ..., J_n$, sea $\mathscr{X}$ el conjunto de todas las asignaciones secuenciales de trabajos a máquinas tales que cada trabajo es procesado por cada máquina exactamente una vez, donde cada elemento $x \in \mathscr{X}$ pueda ser escrito como una matriz $n \times m$, en la que la columna i represente la lista ordenada de los trabajos que la máquina M_i procesará y suponiendo que existe una función de coste $C : \mathscr{X} \to [0, +\infty]$ que puede ser interpretada como el tiempo total de proceso y funciones $C_{ij} : M \times J \to [0, +\infty]$ que representan el tiempo invertido por la máquina M_i en procesar el trabajo J_j, el objetivo del JSP es encontrar una asignación de trabajos $x \in \mathscr{X}$ tal que $C(x)$ sea mínimo, es decir, que no existe ningún $y \in \mathscr{X}$ de forma que $C(x) > C(y)$. El problema del JSP difiere del de la planificación de procesos en que los trabajos son servidos en una sola máquina con lo que no deben pasar por todas ellas.

En otro de los problemas, el KSP, o problema de la mochila, la idea es hallar, dado un conjunto de elementos con un valor y un peso determinados, la cantidad de cada elemento para añadir a una colección de forma que no se sobrepase un peso determinado, que es la capacidad de la mochila, maximizando la suma de valores de los elementos. La formulación más común de este problema es aquella en la que solo existe un elemento de cada tipo (con un par valor–peso concreto). Formalmente, dados N elementos, $z_1, z_2, ..., z_n$ donde cada z_i tiene un valor v_i y un peso w_i, dado x_i, que es el número de copias del elemento w_i que se introducirán en la mochila (que debe ser 0 o 1) y dado el peso máximo W que soporta la mochila y asumiendo que los pesos y los valores son no negativos, se trata de maximizar la expresión $\sum_{i=1}^{n} v_i \cdot x_i$ de forma que $\sum_{i=1}^{n} v_i \cdot x_i \leq W$ con $x_i \in 0, 1$. A pesar de la similitud del KSP con el problema de la planificación de procesos y de que W se consideraría infinito, hay otra diferencia fundamental, y es el hecho de que en la planificación importa el orden en que son tomados los elementos, mientras que el KSP no lo tiene en cuenta. Al cambiar el orden de los elementos se verían alterados los valores v_i afectando al resultado final.

Aunque similar en algunos aspectos a los tres problemas presentados aquí, los tres NP–completos, el problema de la planificación de trabajos en múltiples procesadores, debido a sus particularidades, se merece su propia categoría de problemas pues se puede extrapolar a múltiples ámbitos. Además, las técnicas estudiadas en este trabajo podrían aplicarse, con las modificaciones adecuadas, a los primeros y viceversa. Se detallan a continuación, en las próximas secciones, una serie de cuestiones, ya concretas del problema de la planificación de procesos que deben tenerse

en cuenta en este estudio.

4.1.2. Planificación continua

Uno de los problemas con los que hay que lidiar en el mundo de la planificación de procesos y que no se considera en los modelos de problemas presentados es el carácter dinámico en el conjunto de trabajos a planificar. En problemas tales como el TSP o el JSP el conjunto de trabajos es finito y concreto y se realiza una única planificación para ese conjunto dado. Es obvio que si el conjunto cambia la planificación realizada puede quedar invalidada. Y el conjunto puede cambiar, pues en una cola de trabajos podría desaparecer uno de ellos porque el usuario ha decidido cancelarlo. También podrían aparecer más, pues nuevos usuarios han podido enviar nuevos trabajos al sistema de HPC. En dichos casos sería necesario desechar la última planificación realizada y comenzar una nueva o bien podría utilizarse como punto de partida para esa nueva planificación. Es por esto que la necesidad de planificar puede resultar prácticamente continua.

4.1.3. Naturaleza cambiante y reajuste del planificador

Ni siquiera imaginando un entorno estático en donde los usuarios de un centro de HPC son siempre los mismos y además se comportan con regularidad, enviando trabajos que siempre son de las mismas características, se podría establecer de manera óptima una política de planificación satisfactoria, principalmente porque el número de usuarios podría ser muy elevado. Acotando dicho número tal vez sería posible definir una política que satisfaga de la mejor manera los objetivos perseguidos por el HPC y por sus usuarios. Pero, ¿qué sucede si los trabajos, los propios usuarios o el conjunto de usuarios cambian? ¿Sigue siendo igualmente válida la política definida? Y más sutilmente, ¿qué ocurre si las necesidades de los usuarios van cambiando poco a poco de manera casi imperceptible? Un centro de HPC es un entorno muy complejo en el que no se pueden asumir estas condiciones estáticas de manera que el problema se ve magnificado: cientos o miles de usuarios con necesidades cambiantes día a día obligan a un reajuste continuo del planificador, reajuste muy complicado y que requiere gran cantidad tiempo en el que hay que tener en cuenta muchos factores.

4.1.4. Políticas de planificación

Las políticas de planificación determinan los objetivos del centro de HPC que deben alcanzarse por medio de la explotación de su sistema de HPC. La manera de guiar el proceso principal en que los usuarios envían trabajos, esperan y se ejecutan es estableciendo claramente la forma en la que esos trabajos deben esperar y ejecutarse. La vía para hacer esto es modificar el comportamiento del propio planificador del JMS, pues es el planificador el que decide el momento en el que los trabajos pasan a ejecución y es el que los puede hacer esperar el tiempo necesario,

además de asignar recursos concretos para la ejecución de esos trabajos.

Entre los posibles objetivos que un centro de HPC puede tener se pueden mencionar algunos:

- Terminar lo antes posible, es decir minimizar el *makespan*.
- Maximizar a lo largo del tiempo la ocupación de los recursos con la idea de tenerlos al 100 % de ocupación durante todo el tiempo posible.
- Maximizar el número de trabajos por unidad de tiempo, es decir, maximizar el *throughput*.
- Priorizar los trabajos de los usuarios más importantes.
- Minimizar el coste económico. Es posible contar entre los recursos con algunos que impliquen gasto monetario en función de su uso, por lo que se busca la manera de minimizarlo.
- Asegurarse de que ningún trabajo espera más de un tiempo máximo predefinido.

Una manera de ver estas políticas es pensar en priorizar trabajos, porque en el fondo, no es más que eso. La idea es asignar prioridades a los trabajos para que, al ejecutarse en el orden que determinan esas prioridades, se logre el objetivo perseguido. Pero no es habitual perseguir un solo objetivo y configurar el planificador para lograrlo en la medida de lo posible, sino el combinar diferentes objetivos para alcanzar uno más complejo o incluso variarlos en función de la temporada, el número de usuarios, el tipo de trabajos y diversos factores en esta línea. Una forma de hacer esto es asignando pesos a cada uno de los objetivos para que un valor global de prioridad pueda ser asociado a los trabajos. Esta es la aproximación llevada a cabo por el planificador Maui [4]. Pero, ¿cuál es la forma de asignar prioridades a los trabajos? Hay que fijarse en determinados parámetros de los trabajos para poder asignar las prioridades. Por ejemplo, el *walltime* de un trabajo puede ser usado para favorecer un trabajo en función de su duración, o su tiempo de espera para subir su prioridad y evitar que esperen demasiado tiempo más. Partiendo de estos datos y otros tales como el uso de recursos actual por parte de los trabajos de un usuario o el uso de recursos registrado en históricos se pueden elaborar complejas reglas de asignación de prioridades que deben tener en cuenta el equilibrio de los objetivos perseguidos según su ponderación.

Además de dotar de prioridades a los trabajos, y en el caso de tener un sistema de HPC con recursos heterogéneos, las políticas de planificación deben decidir qué recursos de los disponibles se asignan a los trabajos. Nuevamente, se puede optar por diversas alternativas de asignación entre las que destacan las siguientes:

- El nodo con menos carga de trabajo. Esta política de asignación es válida para trabajos que se sirvan inmediatamente y bajo el supuesto de ejecución de trabajos en modo compartido, es decir, más de un trabajo por nodo simultáneamente.

- El primer nodo disponible. Se selecciona el primer nodo presentado por el JMS. Es una simple política FCFS (*First Come First Served*) lo que hace de la asignación un algoritmo muy eficiente.
- El nodo con más prioridad. Se asocian prioridades estáticas o dinámicas basadas en diferentes parámetros variables de los nodos y se asignan los nodos a los trabajos en orden de prioridad.
- El nodo con los mínimos recursos. Se asigna aquel nodo que tenga los mínimos recursos posibles pero suficientes para servir al trabajo.
- El nodo más rápido. Se asigna el nodo más potente para favorecer su terminación lo antes posible. De esta manera, los nodos menos rápidos se asignan solo si no queda otro remedio.

En resumen, las políticas de planificación deciden el orden óptimo de trabajos que satisface de manera equilibrada los objetivos perseguidos por el centro de HPC. Además, si se trata de un sistema de HPC no heterogéneo, las políticas tendrán en cuenta el orden de asignación de recursos a los trabajos.

4.1.5. Optimizando un planificador

A pesar de lo concisamente que puedan estar definidas las políticas, hay determinadas operaciones que se pueden llevar a cabo para mejorar la planificación. Una de ellas, la que más incidencia tiene en el desarrollo de este trabajo, es el llamado *backfilling*, que consiste en explotar los huecos de recursos que puedan quedar entre ejecuciones de trabajos para adelantar un trabajo de la cola de espera sin que repercuta en la planificación del resto. Mediante esta técnica se logra que varios de los trabajos puedan ejecutarse, y por tanto, finalizar, antes de lo estimado sin que el resto se vea influido negativamente. Para terminar de aclarar el concepto de *backfilling* se presenta un ejemplo en la figura 4.1. En ella se pueden ver dos

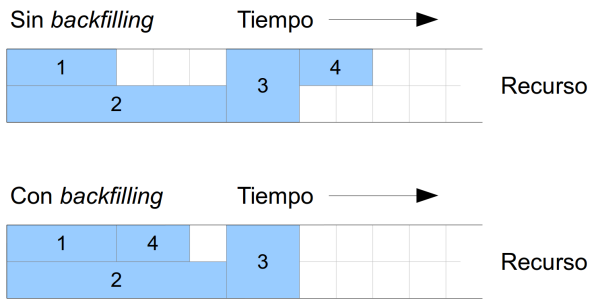

Figura 4.1: Ejemplos de planificación con y sin *backfilling*

escenarios de planificación distintos, donde uno de ellos no aplica la técnica y el otro sí. En el ejemplo, los trabajos 1 y 2 llegan al sistema, que está compuesto de un

único recurso con dos nodos de ejecución, y pasan a ejecución simultáneamente, pues ambos necesitan un único nodo. Mientras se están ejecutando, llega un tercer trabajo, el 3, que al necesitar dos nodos, tiene que esperar a que 1 y 2 terminen, posteriormente llega el trabajo 4 que solo necesita un nodo. El escenario superior, que no aplica la técnica, muestra como, al finalizar los trabajos 1 y 2, el 3 puede entrar a ejecutarse, y una vez que este finaliza, el 4 puede comenzar su ejecución. Mediante el uso de la técnica de *backfilling* el planificador detecta el hueco existente entre la finalización del trabajo 1 y el comienzo programado del trabajo 3 y determina que puede utilizarse para la ejecución del trabajo 4 sin que el 3 se vea retrasado. Como puede observarse, el ejemplo muestra las evidentes ventajas de la aplicación de esta técnica.

4.1.6. Planificando para satisfacer al usuario

Como uno de los principales objetivos de este trabajo se pretende dar la posibilidad a los administradores de centros de HPC de incluir en sus políticas de planificación términos que consideren la satisfacción de los usuarios. Las políticas clásicas, y sus variantes, de minimización del *makespan*, maximización del uso de recursos o del número de trabajos resueltos por unidad de tiempo, por ejemplo, no tienen en cuenta las particularidades de cada usuario. Pero es complicado considerarlas. La principal dificultad de esto radica en la manera de medir la satisfacción. Este tema se abarcará en profundidad en el capítulo 6. Parar ir introduciéndolo podría pensarse en una función, obtenida de una u otra manera, cuyo resultado fuera un indicativo de cuán contento o satisfecho queda el usuario con el servicio recibido. Una función así podría ser usada en un proceso de planificación para lograr equilibrar la satisfacción de todos los usuarios, por ejemplo, pues da la posibilidad de estimar la satisfacción futura de un usuario en función de las predicciones del uso de recursos obtenidas a través del sistema presentado en el capítulo 5 y de los tiempos de espera estimados.

4.2. Principales ideas

Como se ha visto, coexisten diversos problemas que hacen de la planificación una operación compleja y delicada, cosa que se ve agravada cuando se intenta incluir la variable de la satisfacción del usuario. Por todo esto se propone aquí un enfoque para la planificación diferente del habitual pero ya estudiado por varios autores como se puede comprobar en el apartado 3.3 de este trabajo. Este enfoque pasa por el uso de algoritmos evolutivos como componente principal de un planificador de procesos. La idea es aprovechar las ventajas y potencia de los mencionados algoritmos para solucionar o, al menos, aliviar, los citados problemas:

- Los algoritmos evolutivos permiten explorar y considerar un mayor rango de posibles soluciones que las proporcionadas mediante la aplicación de las reglas de planificación establecidas por el centro de HPC.

- El establecimiento de políticas puede ser hecho de manera natural especificando qué se quiere y no cómo se tiene que hacer. La ventaja radica en definir una función que mida la calidad de una posible planificación en función de lo que se desee en cada momento. Esta función, de más fácil definición que, por ejemplo, una política basada en reglas, se puede usar de manera satisfactoria como parte de la función de *fitness* de un algoritmo evolutivo.
- El definir el qué de una política y no el cómo hace que el propio algoritmo de planificación sea dinámico y adaptable a cambios en los trabajos, los propios usuarios o el conjunto de los usuarios, pues se trabaja sobre medidas en un dominio continuo y no sobre reglas categóricas o rangos estáticos.
- Si se pudiera medir la satisfacción del usuario para una posible planificación, como se sugiere en el apartado previo, sería trivial su inclusión en las políticas de planificación y por tanto su uso por parte de la mencionada función de *fitness*.

No obstante, esta aproximación no está exenta de problemas, pues planificar evolutivamente requiere más tiempo y recursos que un proceso clásico. El punto más crítico estriba en el tiempo, pues se espera una planificación ágil. En la sección 4.5 se abordará el tema del tiempo invertido por el planificador.

4.3. Planificando evolutivamente

Esta sección recoge el desarrollo basado en las ideas anteriormente expuestas, es decir, cómo se ha abordado la construcción de un planificador basado en algoritmos evolutivos, cuáles son los algoritmos concretos que se han utilizado y qué decisiones de diseño han guiado el proceso. Posteriormente se describirá la implementación realizada. El sistema propuesto, que forma parte fundamental del EvoProc, se ha denominado SPE, Sistema de Planificación Evolutiva.

4.3.1. Aproximación

El proceso de búsqueda y optimización de planificaciones debe hacerse en un espacio de soluciones que representan posibles planificaciones. Dado que se han usado con éxito los algoritmos evolutivos en problemas de planificación en otros trabajos, como los presentados en la sección 3.3, se propone en esta tesis el uso de dichos algoritmos. Por otro lado, se ha visto en el apartado 4.1.1 que una planificación, en un problema como el TSP, no es más que un orden en las ciudades a visitar y que, generalizando, resulta ser un orden en los trabajos o tareas a realizar en problemas como el JSP. De la misma manera, en el actual problema, una planificación para la ejecución de trabajos es la determinación del orden idóneo que deben tener esos trabajos en una cola de espera. Se debe hacer notar que, a pesar de que en cada ciclo de planificación, solo los primeros trabajos de la cola pasan a ejecución (mientras queden recursos libres), es necesario determinar el orden de todos y cada

uno de los trabajos. Buscar solamente el trabajo (o los trabajos) que van a pasar a ejecución inmediatamente, siguiendo una estrategia voraz, teniendo solo en cuenta los beneficios puntuales sin considerar el resto de trabajos podría llevar a una mala planificación. Es por ello que estos trabajos deben elegirse haciendo una planificación completa para poder determinar el beneficio global y, una vez hecho esto, tomar los trabajos correspondientes para ponerlos en ejecución. Esto convierte el problema en uno de ordenación.

Introducidas estas ideas, se delimita a continuación el modelo de colas con el que se va a trabajar y que se considera aquí como el modelo canónico de supercomputador. Y se hace porque es posible habilitar diversas configuraciones para un supercomputador, entre las que se destacan las siguientes:

Una única cola para todos los nodos de procesos En esta configuración, todos los trabajos esperan en la misma cola, que es única y todos los nodos de proceso están disponibles para los trabajos. Evidentemente, se pueden establecer restricciones tales como el número máximo de procesadores a utilizar. Se le da a este modelo el nombre de 1:1 (1 cola, 1 conjunto de nodos).

Varias colas para todos los nodos de proceso Se configuran diversas colas con diversas características para todos los nodos de proceso. Esta clasificación puede ser en función de las cantidades de recursos solicitadas por los dueños de los trabajos. Por ejemplo, podría existir una configuración con dos colas, una para trabajos cortos y otra para trabajos largos, o una para trabajos monoprocesador y otra para trabajos paralelos. Este tipo de configuración complica la declaración de la política de planificación, pues tiene que tener en cuenta los diversos tipos de colas y asignar los recursos en función de ellas. Este modelo se denominará N:1 (N colas, 1 conjunto de nodos).

Varias colas para varias particiones de los nodos de procesos En esta configuración existen varias colas, al igual que el caso anterior. La diferencia está en que el conjunto total de nodos de proceso se divide en subconjuntos para atender a las diversas colas. Esto da lugar a dos tipos de configuraciones:

- Se hacen tantas particiones de nodos como colas y cada partición atiende a una sola cola y los trabajos de una cola son ejecutables en esa única partición. A pesar de resultar más complejo en configuración que el modelo 1:1, este modelo es, en realidad, un conjunto de N modelos 1:1. La principal ventaja es que al existir N colas ya quedan clasificados los trabajos en función de las características deseadas. De esta manera se tiene más control sobre la ejecución de los trabajos acotando toda la operación. La desventaja es que, al dividir el conjunto de nodos y hacerlo disjunto para cada cola se reduce la disponibilidad máxima de recursos para los trabajos, pues solo se puede optar a los ofrecidos por una partición concreta. Se le da a este modelo el nombre de N:N.
- Se hacen varias particiones de nodos y varias colas. En este modelo, varias particiones pueden atender a varias colas y los trabajos de una cola pueden ser atendidos en varias particiones con toda la flexibilidad y complejidad que ello supone. Por ejemplo, los trabajos de dos colas de

trabajos cortos podrían ser ejecutados en cinco particiones diferentes y cada una de esas particiones podría atender a diversas colas donde, además, una misma cola podría ser atendida por varias de esas cinco particiones. Se denomina a este modelo M:N.

Por ser una configuración básica, la mayor parte de este trabajo se orienta hacia el modelo 1:1, considerado como el modelo canónico. Además, el trabajar sobre el modelo 1:1 repercute directamente en el modelo N:N, que, como se ha visto, no es más que *N* modelos 1:1. Por otro lado, es importante destacar que son precisamente los problemas comentados anteriormente lo que hizo que se adoptaran modelos no 1:1 en los centros de HPC. En este trabajo se pretende resolver dichos problemas, con lo que dejarían de tener sentido los citados modelos en favor del 1:1.

La operativa del modelo 1:1 es sencilla: cuando haya recursos disponibles y trabajos en la cola se planifica para saber qué trabajos pasan a ejecución. Si no hay trabajos o no hay recursos no se planifica. Bajo esta forma de operar, que podrá ser modificada como se verá posteriormente, ha de establecerse la manera en la que actuará la solución evolutiva propuesta, para lo cual hay que combinar varios aspectos:

- Se asume que el JMS hará pasar a ejecución los trabajos que se encuentren en las primeras posiciones de la cola, por orden, mientras haya recursos disponibles. Esto es equivalente a establecer una simple política FIFO en el JMS.
- Como el JMS funciona bajo los preceptos de la política FIFO, hay que alterar el orden de los trabajos en la cola para que pasen a ejecución los trabajos deseados. El encargado de encontrar ese orden será el algoritmo evolutivo. La responsabilidad de guiar el proceso hacia el orden adecuado será de la función de *fitness* que, como se ha dicho, implementa la política de planificación pretendida.
- Para poder realizar una operación de ordenación en la cola, es necesario determinar el fenotipo y el genotipo de los individuos.
- Cuando el JMS necesite disparar el proceso de planificación se invocará al planificador evolutivo que realizará la planificación. Tras esto, los primeros trabajos de la cola planificada pasarán inmediatamente a ejecución.

Además del modelo de operación a utilizar, se asumen para el trabajo, excepto donde esté indicado, sistemas de HPC homogéneos, es decir, aquellos en los que todos los nodos de proceso cuentan con las mismas características.

Teniendo estas ideas en cuenta, se detalla a continuación la manera de llevar a cabo una planificación evolutivamente.

4.3.2. Implementación

Esta sección se adentra en los pormenores del planificador evolutivo desarrollado en esta tesis atendiendo principalmente a tres temas básicos como son la codifi-

cación de la planificación, los algoritmos a utilizar y la forma de evaluar la calidad de las planificaciones.

4.3.2.1. Codificación

Una óptica más formal del problema dice que la solución pasa por encontrar la permutación p de los q trabajos de una cola en un modelo 1:1 tal que el resultado de $f(p)$ sea óptimo, donde la función f es una función de *fitness* representativa de la política de planificación deseada. Partiendo de esta base y aplicando una codificación directa y natural, si el espacio de posibles soluciones, y por tanto, posibles planificaciones, son las $q!$ permutaciones de trabajos, los individuos que conformarán la población que permita realizar la exploración en este espacio será, directamente, un subconjunto de este espacio. Así pues, el problema es encontrar una codificación eficiente para representar a estos individuos, es decir, un orden. Asignando un número de índice o posición a cada uno de los trabajos de en la cola se obtiene una codificación también natural e intuitiva. Suponiendo una cola con cuatro trabajos cuyos nombres son A, B, C y D y asociando un número de índice a cada trabajo que permite ordenar de menor a mayor la cola, se obtiene una posible planificación. De esta manera, el contenido de la cola es $A(i), B(j), C(k)$ y $D(l)$. Basta con ordenar los trabajos en función de los índices i, j, k y l para obtener la planificación dada por ellos. Por ejemplo, la codificación $A(10), B(40), C(30)$ y $D(20)$ resulta en una planificación $ADCB$. Con esto, la codificación real que se necesita queda conformada, solamente, por los números de índice o posición de los trabajos. La ventaja de esta codificación es que resulta natural y directa, pues representa una permutación de los trabajos. Además, ha sido utilizada con éxito en algunos de los trabajos comentados en la sección 3.3 y en general en varios trabajos, como [108], [109] o [110].

4.3.2.2. Algoritmos

La codificación presentada permite la aplicación de diversos algoritmos evolutivos. El estudio de los resultados de varios algoritmos se presenta posteriormente en la sección 4.5. Los algoritmos estudiados en este trabajo son el genético canónico, el micro–genético (MGA o μGA) [111], el CMA–ES (*Covariance Matrix Adaptation – Evolutionary Strategy*) [112], el DE (*Differential Evolution*) [113]. Además, a efectos comparativos, se han realizado pruebas con otros algoritmos heurísticos como son la búsqueda tabú, el ascenso a colinas y el *simulated annealing*.

En general, sin entrar a fondo, estos algoritmos funcionan de la misma manera, que se puede resumir en estos pasos:

1. Generar una población inicial de individuos de manera aleatoria o dirigida.

2. Realizar la evaluación de cada uno de los individuos de la población para asignarles un valor de *fitness* o calidad.

3. Aplicar operadores de reproducción y mutación entre los individuos de la población para obtener nuevos individuos.

4. Generar una nueva población a partir de los nuevos individuos e individuos de la población original.

5. Volver al paso 2 hasta que se cumpla algún criterio de parada, como por ejemplo superar un tiempo máximo determinado o se alcance un valor de calidad concreto.

6. Tomar el mejor individuo de la planificación.

De esta manera, se logran individuos que representan planificaciones concretas sin elaborar ni aplicar reglas explícitas de planificación, pues es el propio algoritmo el que guía la ejecución para lograr optimizar una función de evaluación representativa de la política de planificación deseada.

4.3.2.3. Evaluando las planificaciones

Para cualquiera de las mencionadas técnicas se necesita, como se dijo antes, una función de *fitness* que sirva para medir la calidad de una planificación concreta. Para ello, es necesario realizar una serie de pasos previos a la evaluación. Además de realizar la evidente decodificación de la planificación según la propuesta previa, hay que encontrar la manera de estimar ciertos datos de la planificación. Para ello se simulará el paso del tiempo, la ocupación y liberación de recursos y la puesta en ejecución de los trabajos de la cola. De esta manera se podrán obtener datos fundamentales para la evaluación de la calidad de una planificación, donde se pueden destacar algunos principales, como el momento en el que un trabajo entra en ejecución y el momento en el que termina. Así, la utilización de un sistema de optimización de solicitudes es crucial, pues permite una estimación más exacta de los citados datos, aspecto que será tratado en el capítulo 5. Esquematizando, los pasos a seguir para la evaluación para cada uno de los individuos de una población de planificaciones son:

1. Decodificar al individuo, que permite reconstruir una planificación tal y como la entenderá un JMS o un simulador de ejecución.

2. Ejecutar un simulador para obtener datos estimados concretos de una planificación, en especial, los relativos al uso de recursos por parte de los trabajos.

3. Y, finalmente, la evaluación de esa planificación según la política de planificación deseada.

De esta manera, cada individuo llevará asociada su calidad para permitir al algoritmo guiar la búsqueda hacia la mejor solución óptima.

4.3.2.4. Simulación de las planificaciones

Como ya se explicó, el proceso de simulación es utilizado para obtener determinados datos acerca de la planificación que no se pueden obtener a priori. Estos datos son fundamentales para el cálculo de la calidad de una planificación, por lo que serán utilizados para evaluar a los individuos de la población que representan planificaciones para un conjunto de trabajos dado. Esta operación, se repetirá múltiples veces a lo largo del proceso de búsqueda, pues es un paso previo a cada evaluación, motivo por el cual es un requisito su eficiencia temporal. La idea principal es simular el paso del tiempo y la ocupación y liberación de recursos según los trabajos van poniéndose en ejecución o terminando, respectivamente.

En concreto, los datos que se buscan, los que tienen más interés, son los instantes de comienzo y finalización de los trabajos, ya que ellos determinan el resultado de la planificación y sirven para calcularlo en función de muchas posibles políticas de planificación. Un ejemplo permitirá clarificar la forma de cálculo de los mencionados instantes, para lo cual se supondrá un sistema de HPC homogéneo bajo un modelo de funcionamiento 1:1. Dado un sistema de HPC de 4 nodos de proceso, 32Mb de memoria virtual, 6 trabajos y la planificación A, B, C, D, E y F, la imagen 4.2 representa el resultado de haber ejecutado los trabajos en el orden establecido. Las característica de los trabajos, para los que se asume que han llegado en el instante 0, son las recogidas en la tabla 4.1, donde el tiempo y la cantidad de memoria son proporcionados por el usuario en su solicitud o bien son estimados, como se adelantaba en la introducción a este trabajo, por un sistema de optimización de solicitudes, que se desarrolla en el siguiente capítulo.

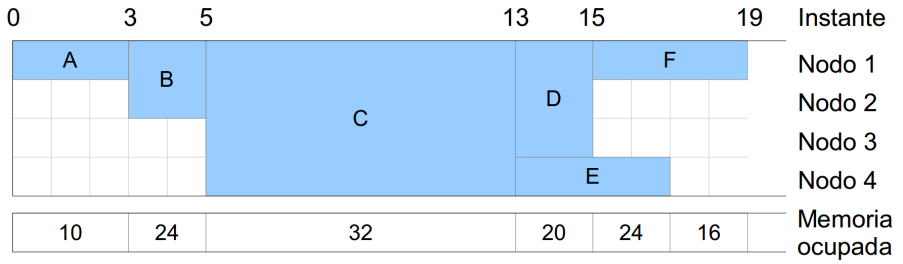

Figura 4.2: Imagen representativa de la planificación de un ejemplo de simulación

En el momento inicial el sistema de HPC tiene todos sus recursos disponibles. El JMS, tras la planificación, asigna trabajos al sistema de HPC, que comienzan su ejecución, hasta que no haya recursos disponibles. Así pues, como al comienzo de la simulación hay 4 nodos libres y 32 Mb de memoria libres el trabajo A puede comenzar, pero no el B, ya que al entrar el A y ocuparse los 10 Mb de memoria que este necesita, no quedan suficientes recursos para que comience el B, que necesita 24 Mb. Como, además, el orden de ejecución ha de ser estricto, no puede entrar ningún otro y los trabajos deben esperar a que queden recursos libres. A las 3 horas (instante 3) el trabajo A finaliza liberando todos los recursos, con lo que el trabajo B puede entrar. Nuevamente, el trabajo C debe esperar a que acabe el B, ya que

necesita 4 nodos de ejecución y 32 Mb de memoria y están ocupados 2 nodos de 4 y 24Mb de 32. Siguiendo con toda la traza de la ejecución, cuando *B* termina (instante 5), *C* comienza. Al terminar *C* (instante 13) pueden comenzar *D* y *E*, ya que hay nodos y memoria suficiente para los dos. Al terminar *D* (instante 15) se liberan los recursos que tenía ocupados y *E* sigue ejecutándose, no obstante, existen recursos suficientes para que pueda empezar la ejecución de *F* y así se hace. Posteriormente terminan *E* (instante 17) y *F* (instante 19).

Trabajo	Nodos	Tiempo (horas)	Memoria (Mb)
A	1	3	10
B	2	2	24
C	4	8	32
D	3	2	12
E	1	4	8
F	2	4	16

Tabla 4.1: Características de los trabajos para un ejemplo de simulación

Con este proceso se han podido obtener los instantes en que todos los trabajos comienzan y terminan sus ejecuciones, datos que se pueden observar en la tabla 4.2. No obstante, el proceso permite conocer otras cosas como el tiempo de espera de los trabajos, la cantidad de trabajos ejecutados en un tiempo dado, la cantidad media de trabajos ejecutados por unidad de tiempo, la ocupación de recursos en cualquier instante, etc., medidas útiles para la implementación de políticas de planificación.

Trabajo	Comienzo	Fin
A	0	3
B	3	5
C	5	13
D	13	15
E	13	17
F	15	19

Tabla 4.2: Horas de inicio y de finalización de los trabajos para un ejemplo de simulación

4.3.2.5. Implementación de políticas de planificación

En la sección 4.1.4 se comentó el porqué de las políticas de planificación y cuáles son las más comunes. Además, se esbozó la manera de abordar o implementar estas políticas para su funcionamiento en el planificador de un JMS. Posteriormente se habló de una de las formas más típicas de aumentar el rendimiento de los planificadores, la técnica *backfilling*. La implementación de políticas triviales o sencillas no acarrea ninguna complicación, pues suelen venir predefinidas y configuradas por

defecto para su uso en un centro de HPC. No obstante, rara vez se utilizan tal cual debido a los diferentes intereses que pueda tener el personal de los centros. Es por tanto que resulta necesario la implementación de una o varias políticas concretas. Esto requiere de experiencia y de tiempo, pues la cantidad de parámetros a manejar es elevada. Además, tras haber implementado la política es necesario probarla de alguna manera para comprobar que efectivamente arroja los resultados esperados. Muchas veces, algunas de estas pruebas han de ser en producción, siempre y cuando no haya un simulador disponible, lo que genera cierta incertidumbre y afecta tanto al centro como a los usuarios.

La ventaja de utilizar métodos de optimización como los sugeridos aquí es la implementación sencilla de políticas de planificación. Basta para ello con proveer una función que, dada una planificación concreta y el resultado de su simulación, proporcione un valor indicativo de la calidad de la planificación. En su configuración más trivial, esta función pasa a ser directamente la función de calidad o *fitness* que guíe al algoritmo de optimización.

Como ya se dijo, la diferencia entre implementar una política de manera tradicional frente a abordarlo a través de esta otra alternativa es hacerlo especificando clara y concisamente cómo se quiere lograr, con toda la configuración o creación de reglas que ello requiera, frente a especificar el qué se quiere lograr simplemente puntuando las planificaciones. Sirva un ejemplo para aclarar esto. Supóngase que se desea implementar una política para minimizar el *makespan*. Como se sabe, el *makespan* viene dado por el tiempo que transcurre entre que el primer trabajo comienza a ejecutarse y el último trabajo termina su ejecución. El problema de la minimización de esta medida es un problema ampliamente estudiado que no tiene una solución trivial o directa. Por ejemplo, Ghirardi y Potts presentan en [114] un método para la minimización del *makespan* de trabajos en un máquina de computación paralela que produce aproximaciones en tiempo polinómico para conjuntos de hasta mil trabajos y en *clusters* de hasta cincuenta nodos. Su método está basado en una reciente modificación del clásico algoritmo *beam search* o recorte de caminos, que es, a su vez, una adaptación de la búsqueda en anchura. El algoritmo fue presentado por Croce et al. en [115] y bautizado como *recovering beam search* (RBS). Otro ejemplo más actual es el de Damodaran et al. [116] en el que se recoge una heurística constructiva para la minimización de esta medida basada en otra heurística propuesta por Dupont et al. en [117] con resultados en tiempo relativamente corto, según los autores, aplicables en un caso real.

Con el uso de un algoritmo de optimización, por ejemplo, un evolutivo, la construcción de la política resulta mucho más cómoda y sencilla. Para el caso concreto del *makespan* esta sencillez se acentúa hasta el extremo, pues basta con implementar una función que devuelva el valor del *makespan* obtenido por el simulador tal y como se ve en la sección anterior. Con esto, y configurando el algoritmo para minimización, se logra la implementación de la política de manera muy directa.

Concluyendo, los objetivos perseguidos pueden tener una implementación tradicional más o menos compleja. No obstante, esta complejidad siempre se ve reducida al verse implementada la política como una función de calidad.

4.3.3. Integración con SGE

Inicialmente se realizó un desarrollo de EvoProc concebido para ser integrado en el sistema de gestión de colas Sun Grid Engine, debido a su amplia difusión en centros de HPC y por ser utilizado en el Centro de Supercomputación de Galicia, con el que la Universidade da Coruña tuvo una colaboración en el marco de un proyecto de investigación[2], como ya se indicó en la introducción de este trabajo, financiado por la Xunta de Galicia y que ha sido parte del trabajo de esta tesis. La idea principal recaía en la integración de la implementación general previamente descrita en el propio SGE. Dicha integración pasaba por la modificación del propio algoritmo original de planificación del gestor para, en lugar de utilizar las reglas de las políticas utilizadas junto con sus restricciones, se hiciera uso del algoritmo evolutivo para generar las planificaciones. A su vez, y dado que el algoritmo evolutivo utiliza un simulador de ejecución de planificaciones, fue necesaria la implementación del simulador en los términos del propio SGE. Este desarrollo permitía la integración de los subsistemas de EvoProc presentados en los capítulos 5, 6 y 7.

El principal problema de este desarrollo es que resultó demasiado dependiente del propio SGE, por lo que en aras de la portabilidad se decidió realizar un desarrollo independiente del JMS que pudiera ser usado por cualquier otro gestor por medio de la implementación de una pequeña capa de adaptación. Además, las continuas actualizaciones del SGE generaban problemas de incompatibilidad entre versiones que obligaban a modificar parte del desarrollo realizado, apoyando así la decisión de mantener únicamente una implementación general del sistema. Se pueden consultar algunos detalles técnicos de la integración con el SGE en el apartado B de los apéndices de esta memoria.

4.4. Esquema del proceso de planificación

Con todo lo presentado aquí, el ciclo de vida de la planificación en un sistema de HPC queda reflejado en la figura 4.3. En la figura se presentan en azul las acciones realizadas por el administrador, en verde las realizadas por los usuarios y en rojo las realizadas por los sistemas propuestos en esta tesis. En gris se puede ver la acción de ejecución de trabajos, por parte de los nodos de ejecución.

En el mencionado proceso de planificación, la única tarea del administrador del centro es la de establecer las políticas de planificación de manera natural según lo comentado en este capítulo. El usuario, por su parte, se conecta al sistema y envía trabajos para su ejecución. Las solicitudes llegan al planificador que comienza la evolución realizando simulaciones de planificación que posteriormente evalúa. Durante esta fase de evaluación se hace uso de las políticas de planificación establecidas por el administrador. Cuando termina la evolución, la mejor de las planificaciones encontradas es utilizada y algunos trabajos son puestos en ejecución, momento en el cual termina el ciclo.

[2]El título del proyecto es "Arquitectura de agentes autónomos para el modelado de usuarios y la gestión óptima de un centro de supercomputación".

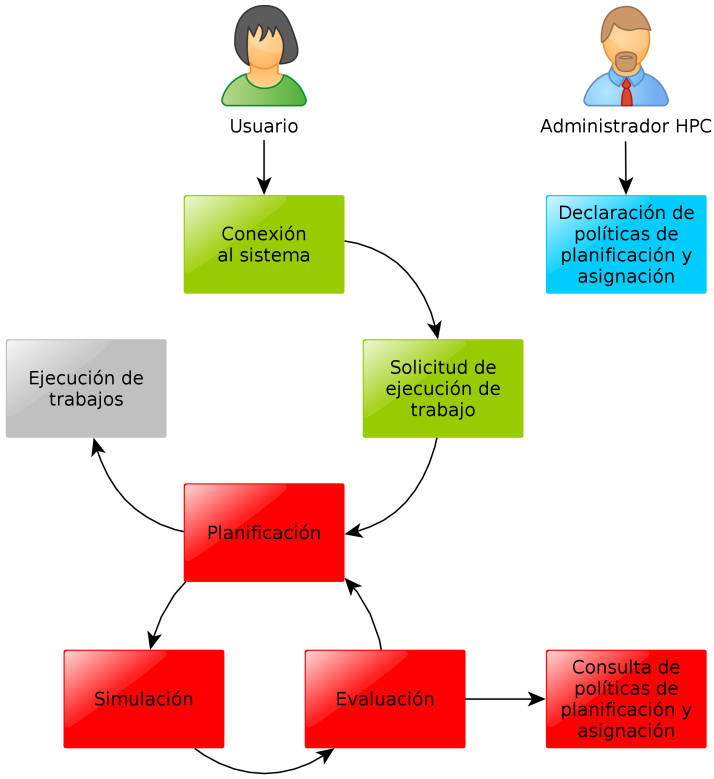

Figura 4.3: Eventos en el proceso de planificación evolutiva

El esquema aquí presentado sirve de base para las posteriores mejoras presentadas en los capítulos 5, 6 y 7, sobre el que se integrarán los nuevos subsistemas.

A continuación, en lo que resta de capítulo, se presentan una serie de experimentos de planificador con la solución aquí propuesta para terminar con una breve conclusión acerca de la aproximación propuesta.

4.5. Pruebas y resultados

Se busca en este apartado demostrar que el planificador evolutivo resulta eficaz aportando planificaciones y que, además, resulta trivial la implantación y aplicación de políticas de planificación gracias a él. Nuevamente, y aunque se haya realizado un estudio con múltiples algoritmos evolutivos y técnicas de optimización y comparado sus resultados, no se trata en esta tesis de encontrar el algoritmo que mejor se adapte al problema o de batir ninguna marca, sino de proporcionar una solución plausible.

Se han dividido los experimentos del estudio en varios bloques para demostrar diversas hipótesis y comprobar distintas características de la planificación. De esta manera se presenta una primera prueba para descartar un planificador por fuerza bruta en favor de las técnicas de optimización. Posteriormente se comprobará cómo funcionan distintas técnicas con una cola de trabajos dada para finalizar con un bloque de pruebas sobre colas de trabajos generadas de manera aleatoria de diferentes tamaños que, además de permitir comparar el rendimiento de las diferentes técnicas utilizadas sirve como pequeño análisis de escalabilidad.

Todos aquellas pruebas cuyos resultados, presentados a continuación, tengan que ver con medidas de tiempo de ejecución han sido realizadas en un equipo con procesador *Intel i7-3630QM* a 2400 MHz con 8 Gb de memoria RAM.

4.5.1. Planificando por fuerza bruta

Como se dijo, se pretende mostrar que un algoritmo de búsqueda ciega como el de fuerza bruta, que busca la solución óptima en todo el espacio de búsqueda no resulta adecuado para el problema aquí tratado por el elevado tiempo de ejecución necesario, lo que lo hace inútil en un escenario real.

Se han llevado a cabo pruebas con dos políticas de planificación diferentes y, por tanto, con dos funciones de evaluación diferentes. Una es la minimización del *makespan*. La otra busca mantener un ritmo constante en la puesta de trabajos en ejecución intentando que dicho ritmo sea lo más alto posible. En otras palabras, se busca maximizar la frecuencia de trabajos puestos en ejecución. Para diversas longitudes de cola (T) entre los 4 y los 13 trabajos se han simulado sistemas de HPC homogéneos de entre 1 y 64 procesadores (P). Las tablas expuestas a continuación muestran, en cada casilla, el tiempo en segundos que ha tardado la prueba (valor superior) y el número de soluciones óptimas (valor inferior) obtenidas por el algoritmo de fuerza bruta. Los trabajos de la cola se han generado de manera aleatoria con duraciones entre 1 y 500 segundos teniendo todos la misma fecha y hora de envío al sistema. Por otro lado, el número de procesos se ha calculado al azar entre 1 y el número máximo de procesadores del sistema de HPC.

T/P	1	2	4	8	16	32	64
4	0,014 24	0,004 24	0,005 24	0,004 24	0,005 12	0,004 24	0,005 12
6	0,144 720	0,075 72	0,047 240	0,031 468	0,026 92	0,013 72	0,013 76
8	0,214 40320	0,132 2880	0,149 1752	0,131 9978	0,130 3324	0,161 4260	0,146 3608
9	1,311 362880	0,520 5760	0,474 13488	0,543 860	0,472 1200	0,485 504	0,481 6234
10	5,497 3628800	5,149 40320	5,141 48672	5,304 9432	5,177 6456	5,372 1192	5,437 472
11	62,986 39916800	62,932 34560	63,258 2176	64,375 5760	65,665 4352976	66,255 1792	64,473 11028
12	766,129 479001600	674,469 2903040	678,351 46872000	675,437 76417200	689,502 27452582	710,474 22692	687,855 15840
13	10466,705 1932053504	9470,291 1451520	9421,772 29151360	9875,016 11705040	9635,060 1683642240	9588,305 91513590	9935,785 14632

Tabla 4.3: Resultados para el algoritmo de fuerza bruta minimizando el *makespan*

En el primer caso, tabla 4.3, en donde se minimiza el *makespan* se pueden realizar varias observaciones. El tiempo de ejecución de las pruebas depende mucho más, como resulta evidente, del número de trabajos en cola que del número de procesadores disponible. De hecho, el tiempo de ejecución crece consistentemente con el número de permutaciones de trabajos en la cola, es decir con el factorial del número de trabajos. Sin embargo, no hay una diferencia apreciable manteniendo fijo el número de trabajos en cola y variando el tamaño del sistema de HPC.

A la vista de estos resultados se puede concluir, además, y como se indicó previamente, que la minimización del *makespan* no resulta una buena política de planificación, pues el conjunto de soluciones óptimas resulta tan elevado que cabe pensar que tomando una de ellas al azar difícilmente se cumplirán otros requisitos deseables para la planificación. Cabe destacar el caso del uso de un solo procesador para ejecución, en el que, para la minimización del *makespan*, cualquier solución es óptima, pues al ejecutarse todos los trabajos secuencialmente siempre se obtiene el mismo valor independientemente del orden.

En los datos relativos al segundo caso, tabla 4.4, se puede comprobar que los tiempos de ejecución se mantienen similares a los del primer caso compartiendo las mismas observaciones respecto del número de trabajos en cola y del número de procesadores. No obstante, el uso de otra política queda claramente reflejado en el número de soluciones óptimas para cada combinación de parámetros, que resulta muy inferior respecto del caso anterior.

T/P	1	2	4	8	16	32	64
4	0,005 6	0,001 4	0,001 6	0,002 1	0,002 5	0,003 8	0,001 6
6	0,005 120	0,030 60	0,021 6	0,010 120	0,005 60	0,053 2	0,026 96
8	0,072 5040	0,052 2400	0,056 72	0,059 8	0,053 1152	0,103 6	0,058 174
9	0,534 40320	0,509 14400	0,491 12	0,483 1440	0,480 48	0,561 198	0,519 2
10	5,342 362880	5,407 12	5,259 5184	5,306 36	5,669 8	5,691 24	5,481 84
11	64,919 3628800	63,933 240	66,020 384	66,998 4	66,273 48	64,727 127080	65,033 86400
12	792,194 39916800	716,823 3840	723,914 96	737,233 2	749,709 4	743,578 552	749,625 24
13	10805,459 479001600	9875,171 288	10116,399 691200	10037,106 8	9973,28 24	9819,248 128	9790,430 256

Tabla 4.4: Resultados para el algoritmo de fuerza bruta maximizando la frecuencia de puesta en ejecución

Para ambas políticas de planificación el tiempo de ejecución del planificador resulta muy elevado aún teniendo en cuenta que las funciones de evaluación utilizadas son de bajo coste computacional. Políticas con funciones más complejas implicarían tiempos de ejecución mayores. Adicionalmente, para ejemplos reales dentro de los parámetros manejados aquí se podría establecer un número máximo de once trabajos, o doce dependiendo del caso, para considerar aplicable esta técnica.

Se concluye a la vista de estos resultados que, para el problema aquí tratado, un algoritmo de fuerza bruta no es aplicable cuando el número de trabajos excede

de un umbral muy pequeño. No obstante, por debajo de dicho umbral resulta más seguro el uso de esta técnica frente a cualquier otra ya que siempre proporciona la solución óptima en un tiempo aceptable.

4.5.2. Planificando con carga media

Una vez descartado el algoritmo de fuerza bruta se hace muy interesante su comparación con otros algoritmos de optimización para un caso concreto. A tal efecto se ha diseñado un ejemplo con once trabajos para un sistema de HPC homogéneo de cuatro procesadores. El ejemplo considera trabajos que difieren en requisitos de tiempo de ejecución y de número de procesos y se han dispuesto en el tiempo según su orden de llegada como para que la planificación basada en una política FIFO no resulte óptima desde el punto de vista de la optimización del *makespan*. Se calcula entonces la planificación óptima por medio del algoritmo de fuerza bruta para comprobar cómo difiere de la planificación FIFO por defecto. A continuación se aplicarán otras técnicas de optimización comprobando los resultados obtenidos así como la diferencia de tiempos de una a otra. La función de evaluación utilizada es la minimización del *makespan*, ya que, aunque no resulte una política adecuada para un caso real, sí lo es a efectos de estudio.

Trabajo	Procesadores	Tiempo de ejecución (s)
1	1	25000
2	4	4000
3	1	5000
4	2	10000
5	1	10000
6	2	2000
7	1	8000
8	1	17000
9	1	11000
10	3	3000
11	1	19000

Tabla 4.5: Requisitos de los trabajos utilizados para la comparativa de técnicas

Los requisitos de los once trabajos preparados para el ejemplo quedan reflejados en la tabla 4.5, en la que los trabajos están ordenados de manera natural según su orden de llegada, siendo el primero mostrado el primero en llegar al sistema de HPC, el siguiente, el segundo y así sucesivamente.

De esta manera, la planificación FIFO resultante se ve reflejada en la figura 4.4 arrojando un *makespan* de 74.000 segundos y logrando un 48,65 % de ocupación de los procesadores. Por medio del uso del algoritmo de fuerza bruta las planificaciones óptimas logradas, que son 1.824 de un espacio total de 39.916.800, alcanzan, en un tiempo en torno a los 65 segundos, un *makespan* de 36.000 segundos y un 100 % de ocupación de los procesadores. Una de las soluciones del conjunto de soluciones

óptimas obtenido se puede ver en la figura 4.5. Es fácil ver otras soluciones óptimas en dicha figura, pues intercambiando, por ejemplo, los trabajos 7 y 9, o 4 y 5, el *makespan* sigue siendo mínimo.

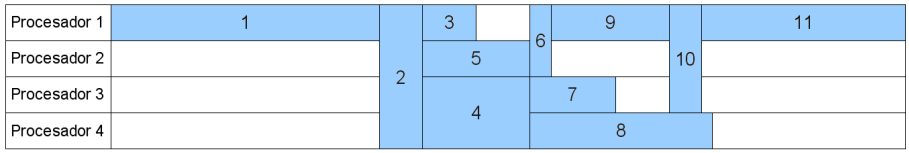

Figura 4.4: Planificación para una cola con política FIFO según orden de entrada

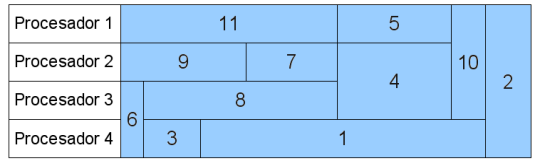

Figura 4.5: Planificación óptima para una cola con algoritmo de fuerza bruta

En la tabla 4.6 se recogen los resultados de diferentes técnicas de optimización utilizadas así como los parámetros usados para su comparativa. Cada algoritmo se ha ejecutado veinte veces para promediar el tiempo y para comprobar el número de veces que se alcanza la solución óptima. De izquierda a derecha, las columnas reflejan la siguiente información: el algoritmo utilizado, el tiempo de ejecución promediado en segundos, el número de veces que se encontró una solución óptima (de las 20 ejecuciones), el número de iteraciones que se ejecutó el algoritmo, el número de individuos de la población si es aplicable y los parámetros comunes utilizados para cada algoritmo.

En la tabla 4.7 se pueden observar las mismas medidas que en el caso anterior pero para una cola de 220 trabajos. Dicha cola se genera replicando 20 veces la cola de 11 trabajos comentada previamente, en la que el *makespan* óptimo es de 720.000 segundos con un 100 % de ocupación de los procesadores. El hecho de replicar los trabajos hace que el número de soluciones óptimas para esta cola crezca en mayor medida, por los mismos motivos expuestos antes. En la citada tabla, y de izquierda a derecha, las columnas representan la siguiente información: el algoritmo utilizado, el tiempo de ejecución promediado en segundos, los dos mejores valores de *fitness* alcanzados, que en este caso es el *makespan*, junto con el número de veces que se ha alcanzado cada uno, el número de iteraciones del algoritmo, el número de individuos de la población si es aplicable y los parámetros comunes utilizados en cada caso. Tanto para las pruebas de la cola de 11 trabajos como para las de la cola de 220 los valores de los parámetros se han extraído mediante un barrido en torno a los valores más habituales para cada algoritmo. Así, por mencionar algunos, para el algoritmo genético se ha barrido la probabilidad de cruce en torno al 60–70 % y la de mutación en torno al 1 %, para el algoritmo SA, los valores de temperatura

inicial y tasa de enfriamiento son determinados por el propio algoritmo. Para el algoritmo DE se utilizan los valores de λ y F sugeridos por los propios creadores, mientras que para el CMA–ES, se utiliza un λ igual al tamaño de la población y un μ barrido entre 1 y 8.

Alg.	T. ejec. (s)	S. óptimas	Iter.	Pobl.	Parámetros
CMA–ES	0,187	11 (55 %)	2000	17	$\lambda = 17$ $\mu = 1$ Sin reinicio de población
DE	0,156	20 (100 %)	1000	50	$\lambda = 1$ Estrategia *current to p best mutation strategy* (F = 0,9, p = 0,1) Cruce binomial (CR = 0,0)
GA (BLX)	0,057	20 (100 %)	50	110	Cruce BLX (p = 0,6, $\alpha = 0{,}5$) Mutación Michalewicz no uniforme (p = 0,01)
GA (CX)	0,057	20 (100 %)	50	110	Cruce CX (p = 0,6) Mutación *swap* de 2 genes (p = 0,01)
μGA	0,056	20 (100 %)	50	5	Cruce CX (p = 0,8) Mutación *swap* de 2 genes (p = 0,5) Se supone que la población converge a las 10 generaciones
Aleatorio	0,229	20 (100 %)	1000000	-	-
Gradiente	1,607	1 (5 %)	1000000	-	Punto de comienzo aleatorio
SA	0,084	3 (15 %)	138	-	Temperatura inicial ≈ 850 Tasa de enfriamiento ≈ 93 (parámetros e iteraciones determinados por el algoritmo) Partiendo de cola aleatoria
Tabú	0,091	20 (100 %)	1000	-	*Tenure* = 6, determinado empíricamente Partiendo de cola aleatoria

Tabla 4.6: Resultados de la comparativa de técnicas para cola de 11 trabajos

Alg.	T. ejec. (s)	*Fitness*	Iter.	Pobl.	Parámetros
CMA–ES	100,957	730000 (1) 742000 (2)	5000	330	$\lambda = 330$ $\mu = 1$ Sin reinicio de población
DE	104,276	734000 (4) 737000 (4)	1000	220	$\lambda = 1$ Estrategia *current to p best mutation strategy* (F = 0,9, p = 0,1) Cruce binomial (CR = 0,0)
GA (BLX)	117,053	740000 (1) 748000 (1)	600	2200	Cruce BLX (p = 0,6, $\alpha = 0{,}5$) Mutación Michalewicz no uniforme (p = 0,01)
GA (CX)	109,230	730000 (1) 739000 (2)	600	2200	Cruce CX (p = 0,6) Mutación *swap* de 2 genes (p = 0,01)
MGA	102,362	720000 (2) 722000 (5)	20000	4	Cruce CX (p = 0,95) Mutación *swap* de 2 genes (p = 0,65) Se supone que la población converge a las 20 generaciones
Aleatorio	118,247	816000 (1) 828000 (1)	2000000	-	-
Gradiente	104,751	726000 (1) 736000 (1)	12000	-	Punto de comienzo aleatorio
SA	56,855	738000 (3) 746000 (1)	2500	-	Temperatura inicial ≈ 200 Tasa de enfriamiento ≈ 51 (parámetros determinados por el algoritmo) Partiendo de cola aleatoria
Tabú	109,975	728000 (1) 758000 (1)	15000	-	*Tenure* = 110, determinado empíricamente Partiendo de cola aleatoria

Tabla 4.7: Resultados de la comparativa de técnicas para cola de 220 trabajos

A la vista de los resultados de las tablas 4.6 y 4.7 se pueden extraer las siguientes conclusiones:

- Para una cola de 11 trabajos, un algoritmo de búsqueda aleatoria, aunque no resulta el más rápido, sí es bastante eficaz dando con una solución óptima en poco tiempo y el 100 % de las veces. Las cosas cambian cuando la cola de trabajos es más grande, como se puede apreciar en el caso de los 220 trabajos, donde dicho algoritmo es el que sale peor parado, quedando demostrado que es fundamental la información para realizar una buena planificación.

- El algoritmo de gradiente resulta el peor en el caso de la cola de 11 trabajos a pesar de haberle permitido ejecutarse más tiempo que al resto, alcanzando una única vez la solución óptima. Se puede sacar la conclusión de la existencia de muchos mínimos locales que hacen que el algoritmo se detenga. Para el caso de la cola de 220, y debido al gran aumento del número de soluciones óptimas en el espacio de búsqueda, el algoritmo resulta mejor, alcanzando una vez el valor de 726.000 segundos, que no dista demasiado del *makespan* óptimo.

- En la cola de 11 trabajos, el CMA–ES y el SA alcanzan alguna vez la solución óptima pero no siempre, resultando peores que el algoritmo de búsqueda aleatoria a pesar de ser algoritmos más elaborados. Probablemente esto se deba a que ambos algoritmos tienen dificultades en espacios de búsqueda con múltiples óptimos locales distantes entre sí. El resto de algoritmos, el DE, el GA, el MGA y la búsqueda tabú, resuelven el problema sin complicaciones alcanzando el 100 % de las veces una solución óptima.

- En el caso de los 220 trabajos, el SA, debido a su implementación, detiene su ejecución a los 57 segundos, distando bastante del orden de los dos minutos en los que se ejecuta el resto de algoritmos. Las soluciones obtenidas son peores que las del algoritmo de gradiente por lo que no resulta destacable.

- La búsqueda tabú, que resulta satisfactoria para el caso de 11 trabajos, en el de 220 logra una solución aceptable de 728.000 segundos. No obstante, la segunda mejor solución, de 758.000, dista más de lo esperado.

- El caso del algoritmo genético canónico configurado con un cruce BLX (*Blend crossover*) [118] resulta mediocre al igual que el DE, a pesar de que este último es más estable y se acerca más veces a las mejores soluciones encontradas.

- Los algoritmos CMA–ES y el genético canónico con el cruce CX (*Cycle Crossover*) [119] se acercan algo más a una solución óptima. No obstante, todavía quedan por debajo de la mejor solución encontrada por la búsqueda tabú.

- La mención especial la obtiene el algoritmo micro–genético, MGA, que ha sido el único en lograr dos veces la solución óptima y cinco veces más una solución mejor que la mejor de cualquier otro de los algoritmos, hallando que este tipo de algoritmo puede resultar muy adecuado para el problema de la planificación de procesos tal y como se está abarcando en este trabajo.

Tras analizar el problema a resolver, es posible concluir que se trata de un problema *multimodal*, pues pueden existir varias soluciones, y no separable, ya que el valor que un gen aporta a la calidad o validez final de un individuo depende de los

valores que tomen los otros genes. Estas características y su relación con las capacidades de una serie de algoritmos evolutivos han sido estudiadas por Caamaño *et al.* en [120, 121]. Al utilizar una codificación de números enteros, los operadores de reproducción más adecuados son aquellos de tipo intercambio. Algoritmos como el DE o el CMA–ES, eficientes con codificaciones reales, requieren un mayor número de generaciones tratando de optimizar una función que, al tener codificación entera, es escalonada. Estos algoritmos se ven privados de determinada información sobre el espacio de búsqueda, que es aquella existente entre cada escalón. Esto no ocurre cuando el algoritmo aplicado utiliza operadores de tipo intercambio, como en el caso del micro–genético, por ejemplo, que sí tienen en cuenta esta característica del problema haciendo una búsqueda sobre el espacio de calidad más eficiente. Adicionalmente, en problemas de alta dimensión, es decir, con colas de muchos trabajos, el CMA–ES resulta poco eficiente pues necesita mucho tiempo de cómputo para adaptar la matriz de covarianzas.

4.5.3. Planificando con carga alta

En este apartado se comparan los resultados de los tres algoritmos que han ofrecido mejores resultados en la prueba anterior para una cola de 500 trabajos generada aleatoriamente para su ejecución en un sistema de HPC de 64 nodos de ejecución. Los trabajos generados tienen un tiempo de ejecución máximo de 36.000 segundos y utilizan hasta un máximo de 48 nodos de ejecución. La cola, de ser ejecutados los trabajos en el orden generado, arroja un *makespan* de 4.723.192. Los resultados de las ejecuciones están recogidos en la tabla 4.8. En dicha tabla se puede observar cómo el algoritmo micro–genético sigue obteniendo mejores resultados que los otros dos algoritmos.

Alg.	T. ejec. (s)	*Fitness*	Iter.	Pobl.	Parámetros
MGA	159,619	3509979 3532878	12000	4	Cruce CX (p = 0,95) Mutación *swap* de 2 genes (p = 0,65) Se supone que la población converge a las 20 generaciones
Gradiente	155,329	3618848 3673017	2500	-	Punto de comienzo aleatorio
Tabú	149,215	3965329 4029905	3300	-	*Tenure* = 250, determinado empíricamente Partiendo de cola aleatoria

Tabla 4.8: Resultados de la comparativa de técnicas para cola de 500 trabajos

Adicionalmente, con tiempos en torno a los 640 segundos, el algoritmo micro–genético logra soluciones entre los 3.480.571 y los 3.487.605 segundos de *makespan*, siendo el valor óptimo teórico de 3.410.326,95, valor calculado sumando las duraciones totales de todos los trabajos generados y dividiendo el resultado entre el número de nodos de ejecución del sistema de HPC.

4.6. Resumen y conclusiones

Los planificadores de los gestores de procesos actuales manejan una cantidad elevada de parámetros que hace complicada su configuración. Además, debido a los continuos cambios en el conjunto de usuarios así como en el comportamiento de los mismos, la frecuencia de reconfiguración del planificador para ajustarse a los objetivos del centro de supercomputación es elevada. Para lidiar con estas reconfiguraciones así como aliviar y facilitar el propio proceso de configuración algunos trabajos ya comentados [6, 7, 38, 41, 46] se han llevado a cabo por diversos investigadores. Estos trabajos, además de estudiar la manera en que afecta la modificación de determinados parámetros a las planificaciones a través de simulaciones, proponen sistemas que de manera automática modifican su configuración y políticas para adaptarse a la situación de las colas de trabajos. La principal objeción es que dicho ajuste se hace sobre un conjunto de parámetros concreto y para un conjunto limitado de políticas. Por otro lado, se ha estudiado el uso de técnicas evolutivas [8, 9, 10, 47, 55, 57] para planificación demostrando que resultan un buen recurso para la obtención de planificaciones automáticas así como para evitar las continuas reconfiguraciones, aunque algunos de estos trabajos se centran en medidas, como el *makespan*, que no llevan a buenas planificaciones bajo todos los supuestos o utilizan técnicas de balanceo de carga mediante migración de trabajos, que no siempre es fácil de implementar además de no estar soportado por todos los sistemas.

Uno de los principales objetivos de este trabajo fue el desarrollo de un planificador basado en técnicas evolutivas, el Sistema de Planificación Evolutiva o SPE, atendiendo a tres puntos claves principales:

1. Facilidad de configuración. El planificador es capaz de realizar su trabajo partiendo de una configuración muy básica de una única cola para todos los usuarios y recursos. Esto hace la configuración del sistema extremadamente sencilla. Además, se facilita la tarea del envío de trabajos a los usuarios, ya que no tienen que especificar en qué cola depositarán sus trabajos.

2. Establecimiento de políticas. El planificador permite el establecimiento de políticas de manera natural y declarativa donde los administradores del centro solo deben indicar qué quieren y no el cómo conseguirlo a través de un tedioso y complejo ajuste de múltiples parámetros. En este sentido, el planificador es abierto y está preparado para trabajar con cualquier política, como por ejemplo, aquellas que tengan en cuenta criterios subjetivos como la satisfacción de los usuarios.

3. Basado en simulaciones. Uno de los principales componentes del planificador es un simulador que permite obtener el resultado de una planificación dada para un estado de cola y una política concretos para permitir al planificador tomar sus decisiones.

En el estudio realizado se han comparado diversas técnicas de optimización sobre múltiples supuestos tomando como base la medida del *makespan*, que si bien en la práctica no resulta una medida adecuada, sí lo es a efectos de estudio. Se ha

comprobado que un algoritmo de planificación por fuerza bruta, que prueba todas las posibles permutaciones de la cola de trabajos, queda totalmente descartado para colas con más de 11 o 12 trabajos. Por encima de este valor el tiempo dedicado para la planificación es demasiado elevado. Cabe decir, que para colas que no superen ese umbral, un algoritmo de fuerza bruta resulta óptimo pues alcanza la mejor de las soluciones en un tiempo perfectamente asumible, por debajo del segundo para colas de 9 trabajos o menos. Para colas mayores, los algoritmos de optimización no informados resultan peores según el número de trabajos crece. Los algoritmos de optimización basados en población resultan adecuados, pudiendo destacar, como principal hallazgo, que el algoritmo micro–genético resulta particularmente bueno alcanzando soluciones óptimas donde los demás fallan. Para una cola de 500 trabajos, el algoritmo MGA ha resultado el mejor de la comparativa.

A la vista de estos resultados, se puede concluir que el planificador desarrollado, utilizando como tecnología de base un MGA, logra los objetivos perseguidos erigiendo los pilares para un planificador que permita el uso de criterios subjetivos en sus políticas.

Es importante destacar que un planificador basado en simulaciones resulta mejor cuanto más fieles son las simulaciones respecto de la realidad. Para lograr esta fidelidad, se propone el modelado de comportamientos, el de satisfacción y el de recursos en los siguientes capítulos.

Capítulo 5

Optimización de las solicitudes de recursos

Mentir siempre no es mentir.

Una vez abordados los problemas de los planificadores mediante la propuesta de un planificador evolutivo, que alivia la tarea de configuración y permite la definición natural de políticas para proporcionar planificaciones de manera automática y adaptativa, se pasa a tratar el principal problema de las solicitudes de recursos por parte de los usuarios con el fin de mejorar el planificador propuesto.

Las solicitudes de recursos que los usuarios hacen cuando envían sus trabajos para ser ejecutados a un centro de supercomputación suelen ser imprecisas o, incluso, inexistentes. En dichas solicitudes se especifica la cantidad de cada recurso concreto que el trabajo va a necesitar. Esta información resulta de vital importancia para el planificador de procesos ya que se basa en ella para poder realizar las planificaciones. La ausencia o vaguedad en las solicitudes causa planificaciones no óptimas con los problemas que ello conlleva, tales como la infrautilización de los recursos o el descontento entre los usuarios del sistema o los propios administradores del centro, que se ven obligados a reconfigurar el planificador y modificar las políticas. Uno de los objetivos planteados en esta tesis es el de modelar el comportamiento de los usuarios de manera individualizada para poder realizar predicciones del uso de recursos, predicciones más precisas que las propias solicitudes, y poder así mejorar las planificaciones. Este modelado es la solución que se propone al problema presentado y que se materializa en un sistema que permite la optimización de las solicitudes de recursos. Dicho sistema se detalla en el presente capítulo.

5.1. Preámbulo

La interacción de los usuarios con un gestor de colas en un centro de HPC es muy pobre en términos de posibles acciones. La principal intención de un usuario es enviar su aplicación o trabajo para su ejecución y su posterior recogida de resultados. Al margen de las consultas que se puedan realizar para comprobar el estado del sistema, en lo que respecta al envío de estos trabajos para ser colocados en una cola y posterior ejecución solo se ha de tomar una acción, que normalmente es del estilo siguiente:

```
$ comando_de_envío script_de_trabajo recursos_solicitados
```

Si se desgrana esta instrucción se puede comprobar de qué manera real se comunica el usuario con el sistema:

- `comando_de_envío` es el comando que cada JMS proporciona para el envío de los trabajos, por ejemplo, `qsub` en el caso del *Grid Engine* de Oracle o del *Open Grid Scheduler*. Su ejecución coloca el trabajo en cola para su posterior planificación.

- `script_de_trabajo` es, típicamente, un *script* con los comandos adecuados para que el trabajo se ejecute y que depende del gestor. El *script* se ejecutará cuando el planificador decida que el trabajo debe ponerse en ejecución.

- `recursos_solicitados` es una lista con los diversos tipos de recursos disponibles para solicitar y la cantidad requerida para el trabajo de cada uno de esos recursos. Es el propio usuario el que decide cómo rellenar esta lista. También es habitual encontrar la lista de recursos solicitados como parte del propio *script* de ejecución del trabajo.

A pesar de ser una única acción resulta una interacción con cierta complejidad por dos motivos. El primero de ellos es que el *script* para el envío del trabajo entraña cierta complejidad. Típicamente hay que indicar algunas opciones dependientes del JMS, como pueden ser el nombre que tomará el trabajo, el proyecto asociado o diferentes direcciones de *e–mail* para comunicar eventos relaciones con el trabajo. Algunas de estas opciones pueden ser comunes a todos los trabajos de un mismo usuario y otras de ellas pueden ser totalmente dependientes del mismo. El segundo de los motivos está directamente relacionado con la solicitud de los recursos. La lista de recursos necesarios se puede rellenar de muchas maneras y con un amplio rango de valores para cada uno de los recursos, con lo que la complejidad no es poca. Entre los posibles recursos a solicitar se deben destacar los siguientes, aunque existen más:

Número de CPUs Es el número total de CPUs o *cores* que necesitará un trabajo.

Tiempo de CPU Es el tiempo de CPU total del que podrá hacer uso el programa, dividido entre el número de procesadores del trabajo.

Tiempo *walltime* Es el tiempo real total máximo que puede durar el trabajo, desde que el primer proceso empieza a ejecutarse hasta que el último termina.

Espacio en memoria RAM (*Random Access Memory*) Es el máximo espacio en memoria del que puede hacer uso el trabajo. Suele distinguirse entre memoria física, virtual y por trabajo o por proceso.

Espacio en disco Análogamente al anterior, es el máximo espacio en disco del que puede hacer uso el trabajo.

A continuación se muestra un ejemplo aclaratorio e ilustrativo de una operación real con el sistema PBS[1]. Para lanzar un trabajo, los usuarios deben crear un *script* similar al siguiente:

```
1:  #!/usr/bin/ksh
2:  #PBS -A prname
3:  #PBS -N test
4:  #PBS -j oe
5:  #PBS -l walltime=1:00:00,size=24000,pvmem=32kb
6:
7:  cd $PBS_O_WORKDIR
8:  date
9:  aprun -n 24000 ./a.out
```

Los números a la izquierda de cada línea no forman parte del *script*, solo representan el número de las propias líneas. La primera de ellas, que es opcional, indica el *shell* (intérprete de comandos) a utilizar bajo el que se ejecutará el *script*, que en caso de no ser especificado lo hará con uno por defecto. Las siguientes cuatro líneas indican al PBS las opciones de envío del trabajo. El hecho de empezar con la etiqueta `#PBS` hace que estas líneas aparezcan como comentarios al *shell* debido al símbolo `#`. PBS buscará estas opciones de envío desde la primera línea del *script* hasta la primera línea que no sea un comentario. A partir de esa línea, cualquier otra que comience con `#PBS` será ignorada. La posterior sección será interpretada por el *shell* y puede contener múltiples líneas de llamadas a ficheros ejecutables, comandos y comentarios. Durante la ejecución normal, el *script* terminará y saldrá de la cola después de su última línea.

En el ejemplo aquí presentado se observa que en la primera línea se indica que el intérprete de comandos deseado es el *Korn Shell*[2] (`ksh`). A continuación, en las líneas 2–3, se indican los parámetros PBS de envío del trabajo. En concreto, en el *script* se está especificando que el trabajo será cargado al proyecto `prname`, que tendrá el nombre de `test` y que la salida y el error estándar serán combinados en la salida estándar (línea 4). La línea 5 indica los recursos solicitados, que en el esquema presentado previamente corresponde con la lista `recursos_solicitados`. En este ejemplo concreto se están solicitando 24.000 *cores* de ejecución durante una hora (*walltime*) donde cada uno de los procesos del trabajo podrá utilizar un máximo de 32kb de memoria virtual. A continuación aparece el bloque de comandos,

[1] Esta forma de operar es la utilizada, por ejemplo, en el superordenador *Titan* [122], que ocupa el segundo puesto de la lista TOP500 en el momento de redactar esta memoria.

[2] Desarrollado por David Korn de los Laboratorios *AT&T Bell* a comienzos de los años 80.

en el que la línea 6 se ha dejado en blanco para facilitar la lectura y será ignorada. La línea 7 cambiará el directorio al predeterminado de envío. La línea 8 ejecutará el comando `date` para conocer la hora en la que dará comienzo la ejecución del trabajo. El comando de la línea 9 lanzará el ejecutable `a.out` con los recursos solicitados en la línea 5.

Una vez generado el *script*, para enviar el trabajo bastará con ejecutar el siguiente comando indicando su nombre, que en este caso es `test.pbs`:

```
$ qsub test.pbs
```

Tras hacer esto, si no hay ningún error, el trabajo será puesto en una cola y quedará a la espera de ser ejecutado. Que los trabajos vayan a parar a una cola sucede, como se comentó previamente, debido a que los recursos son limitados y a que la demanda de sistemas de HPC es muy elevada. Y es precisamente este motivo el que hace necesaria una correcta planificación que determine el orden óptimo en el que los trabajos deben ser ejecutados. Generalmente, el recurso más crítico aquí es el tiempo de CPU y la causa principal es que, para evitar que las planificaciones resulten erróneas o, simplemente, no óptimas, y dado que se tienen en cuenta las solicitudes para llevarlas a cabo, el gestor de trabajos interrumpirá definitivamente la ejecución de aquellos trabajos que sobrepasen alguno de los valores solicitados. A su vez, este es uno de los principales problemas por los cuales la solicitud de recursos carece de la precisión necesaria para llevar a cabo una buena planificación, y es que es muy frecuente que los usuarios no pongan el cuidado suficiente en sus solicitudes. Esto hecho está bien estudiado y reflejado en trabajos como, por ejemplo, [77] o [78]. Los comportamientos típicos que aquí se observan son:

- Los usuarios siempre solicitan el máximo disponible, precisamente debido al miedo a que el gestor detenga sus trabajos.
- Los usuarios siempre solicitan la misma cantidad de recurso. Esto puede ser debido, fundamentalmente, a uno de dos motivos: que en la solicitud no se explicite ningún valor y el gestor asigne la cantidad por defecto, o que el usuario tenga construido un *script* para el envío de trabajo que es siempre igual y donde lo único que cambia es el nombre del programa que desea ejecutar.
- Los usuarios no conocen, o no pueden estimar a priori, el tiempo de ejecución de sus trabajos.

Esto ocurre, además, porque los usuarios no ven, o no conocen, las ventajas de hacer una solicitud más cuidadosa, por ejemplo, que se puedan ver beneficiados (siempre dependiendo de la política de planificación) al obtener sus trabajos prioridad sobre los demás si los valores de recursos pedidos son menores que los solicitados por otros usuarios para sus trabajos. Y es esta falta de precisión en las solicitudes de recursos, lo que lleva al gestor a realizar planificaciones que no son tan buenas como podría esperarse, como afirman Frachtenberg y Feitelson en [11]. De esta manera se crea un conflicto circular:

1. Se necesita una solicitud precisa para hacer una buena planificación.
2. Para evitar que las planificaciones sean erróneas se detienen aquellos trabajos que excedan la cantidad de recursos solicitada.
3. Por miedo a esto, las solicitudes son imprecisas, con lo que las planificaciones no son tan buenas como podrían.

De todo esto surge la necesidad de optimizar de alguna manera las solicitudes de los recursos para que el trabajo del planificador sea realmente efectivo y produzca un aumento en la satisfacción de los usuarios.

5.2. Principales ideas

Si las solicitudes de recursos por parte de los usuarios fuesen precisas, es decir, si tuviesen muy poco porcentaje de error respecto de la cantidad real de recursos utilizadas, los planificadores podrían hacer su trabajo confiando en que el resultado fuera a ser satisfactorio. Como esto no es así, resulta sumamente interesante proveer a los planificadores con más información para que estos puedan tomar sus decisiones. Estos nuevos datos pueden ser estimaciones o predicciones de las cantidades reales de recursos utilizadas por los trabajos. Así, se considera que la forma de optimizar las solicitudes de recursos es acompañarlas de estimaciones del propio uso de los recursos. Esta es la idea principal que se abarca en el presente capítulo y en la que se pretende pasar del habitual escenario en el que el usuario hace sus solicitudes y esta información se traslada directamente al planificador a otro escenario en el que el usuario actúa de la misma forma pero el planificador recibe más datos en los que basar sus decisiones, y ello gracias a un mecanismo de predicción de uso de recursos. Visualmente, se puede comprobar la diferencia en los esquemas de las figura 5.1 y 5.2.

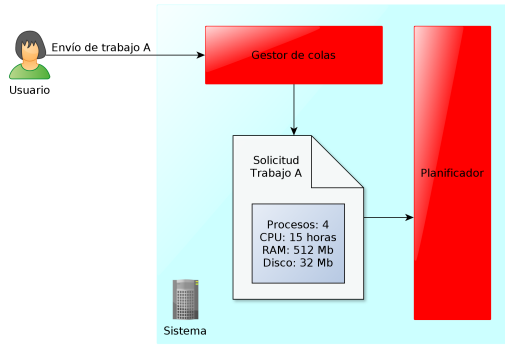

Figura 5.1: Escenario clásico de solicitud de recursos

Lo que aquí se pretende es modelar o perfilar al usuario por medio del aprendizaje de su comportamiento. Mediante este modelado se obtienen mecanismos

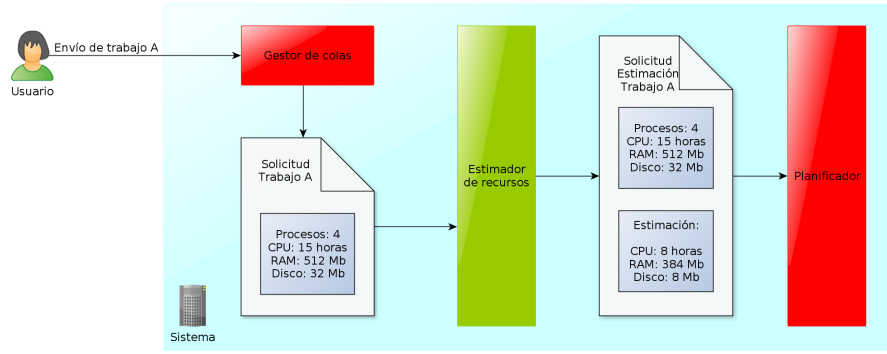

Figura 5.2: Nuevo escenario de solicitud de recursos, transparente para el usuario

capaces de ajustar los datos de las solicitudes cuando el usuario envía trabajos aportando una estimación más precisa. Para construir estos modelos, y como se verá más adelante, se hará uso de una serie de técnicas de inteligencia computacional con objeto de permitir un aprendizaje progresivo del comportamiento de los usuarios mediante *neuro–evolución*. Y es aquí donde radica lo novedoso de esta aproximación, pues como se ha visto en la sección 3.5, los trabajos hasta la fecha realizan las predicciones mediante análisis estadístico y complejos sistemas de clasificación de trabajos.

Así, con el objetivo concisamente especificado, se desarrolla en las siguientes secciones la aproximación para modelar a los usuarios y predecir el uso de recursos que será la base para el sistema de optimización de solicitudes que será integrado en el EvoProc, bautizado como SOS (Sistema de Optimización de Solicitudes).

5.3. Modelando al usuario

Se acomete en esta sección cómo abordar el problema descrito, viéndose, primeramente, la aproximación planteada. Con posterioridad se explicará una posible implementación de dicha aproximación, detallando aspectos de las técnicas concretas utilizadas así como de su ajuste y configuración.

5.3.1. Aproximación

Para intentar solucionar el problema planteado se ha desarrollado un sistema de modelado del comportamiento de los usuarios, considerando la base del comportamiento como la solicitud de recursos. Se busca que este sistema “aprenda” el comportamiento de los usuarios de tal forma que, relacionando variables tales como la solicitud de recursos, la hora del envío del trabajo y los recursos realmente utilizados, por ejemplo, sea capaz de predecir o estimar el uso real de recursos por

parte de los trabajos con el fin de conseguir planificaciones más precisas, como ya se ha establecido claramente. Lo que se pretende modelar es la forma en la que los usuarios solicitan y usan los recursos, es decir, las reglas implícitas o subconscientes que usan y relacionarla con otros parámetros como la hora de envío, para poder inferir de todo ello el uso real de recursos. Además, este modelado debe ser adaptativo para tener en cuenta los cambios de comportamiento en los usuarios y realizarse de manera automática, actualizándose de forma autónoma en cuanto exista nueva información. Así pues, el sistema que aquí se propone debe cumplir con varios requisitos:

- La forma de interacción del usuario con el sistema no debe verse modificada, es decir, la solución tiene que ser totalmente transparente al usuario. Esto evitará cualquier tipo de reaprendizaje de uso del sistema por parte de los usuarios.
- Debe considerar a cada usuario individualmente para poder capturar de manera particular la interacción de cada uno con el sistema, evitando generalizaciones. Es decir, el sistema creará un modelo único para cada usuario.
- Tiene que, a partir de los datos de solicitud de recursos de un trabajo y de información histórica perteneciente al propietario del trabajo, predecir el uso real de recursos de dicho trabajo.
- Los modelos se crearán de manera automática sin intervención humana.
- Los modelos se actualizarán de manera autónoma en cuanto el sistema disponga de nueva información para ello.

Estos requisitos deben ser atendidos considerando que el problema planteado tiene tres características principales:

- La información del comportamiento es implícita, es decir, no está reflejada de manera concisa ni se puede extraer directamente de la interacción del usuario con el sistema.
- La información llega poco a poco y de manera gradual con cada interacción.
- El comportamiento de los usuarios puede ir variando con el tiempo y cambiando con cada interacción.

Estas características hacen que las técnicas ideales para abordar el problema deban permitir un aprendizaje gradual y pueda manejar información de comportamiento implícita sin necesidad de hacerla explícita. La aproximación propuesta debe satisfacer los cinco requisitos previamente indicados. Para hacerlo, hay que prestar especial atención de manera individual a cada uno de ellos. El primero de los requisitos establecidos es la necesidad de que la interacción del usuario con el sistema permanezca igual que hasta el momento. Para satisfacer este requisito basta con no solicitar más información al usuario que la que ya proporciona. Idealmente, no es necesario en absoluto que el usuario conozca el mecanismo de estimación

de recursos, pues será información que se maneje a nivel de planificación. El segundo requisito parte de la idea de que cada usuario es diferente y, por tanto, se considera un error hacer estimaciones en función de una clasificación de trabajos como hacen algunas de las investigaciones comentadas en la sección 3.5. Así, el mecanismo a utilizar debe tratar a cada usuario de manera individual con información exclusiva proporcionada por el propio usuario y la interacción de este con el sistema. Ninguna otra información será tenida en cuenta. Si el usuario cambia de comportamiento debido a factores externos a él, el cambio será capturado en la propia interacción del usuario con el sistema. De la información relacionada con las solicitudes y los trabajos enviados al sistema se generará un modelo de usuario o comportamiento que lo representará. Así, cada usuario tendrá su propio modelo individual y diferenciado del resto. Piro *et al.* [14] y Smith *et al.* [12] llegaron a la conclusión por vías diferentes, en vista de los resultados de sus investigaciones, de que una de las principales categorías en las que se pueden clasificar los trabajos para la predicción de los recursos viene dada por el usuario dueño de los trabajos. Esto apoya la decisión de modelar individualmente a los usuarios. Para satisfacer el tercero de los requisitos, dichos modelos tendrán que ser capaces de predecir o realizar estimaciones acerca del uso real de recursos para cada trabajo que los usuarios envíen para su ejecución. Dichas estimaciones serán proporcionadas por los modelos a partir de la información recibida con las propias solicitudes. La solicitud y la estimación de recursos serán proporcionadas al planificador para que lleve a cabo la tarea de planificación, que será realizada apoyándose en las estimaciones de uso real de recursos y no solo en las solicitudes como se hace clásicamente. Para que la aproximación propuesta sea viable, los modelos han de generarse de manera autónoma sin ningún tipo de intervención humana. Esto implica que si se registra un nuevo usuario en el centro de HPC, el sistema generará automáticamente un modelo que puede empezar a ser actualizado y explotado. Y precisamente respecto de las actualizaciones de los modelos es de lo que trata el último requisito, que es uno de los más importantes. De manera general, el sistema HPC registra información relativa al usuario en dos momentos distintos: cuando recibe una solicitud de ejecución de trabajo y cuando un trabajo termina. Al igual que cuando se recibe una solicitud y el modelo del usuario correspondiente se utiliza para generar una estimación del uso de recursos, cuando un trabajo termina se registra información muy interesante, pues se puede comparar la solicitud de recursos del usuario, la propia estimación del modelo y el uso real de recursos del trabajo. La aproximación propuesta debe tomar dicha información y actualizar el modelo para, primero, capturar el comportamiento del usuario respecto de la forma de solicitar recursos, y segundo, ajustar el modelo a cualquier cambio de comportamiento del usuario que, implícitamente, se ve reflejado en el par solicitud–uso. Es decir, se hace un ajuste gradual del modelo según se va disponiendo de nueva y actualizada información. Adicionalmente, este mecanismo debe poder realizar la operación de actualización por sí mismo y, nuevamente, sin necesidad de intervención humana.

Para abordar estos requisitos se ha desarrollado un algoritmo iterativo que permite modelar el comportamiento de los usuarios basado en dos conceptos fundamentales: que la información llega al sistema de manera gradual y que el comportamiento de los usuarios varía con el tiempo. Dado que el objetivo es realizar un modelado continuo en el tiempo del comportamiento del usuario, se ha denomi-

nado a este algoritmo precisamente como Modelado Continuo de Comportamiento (MCC). El MCC, inspirado en parte en la manera de operar de algunos mecanismos cognitivos, como el MDB (*Multilevel Darwinist Brain*) [123, 124] de Bellas *et al.*, permite construir un modelo por usuario, manteniendo la individualidad del mismo y huyendo de clasificaciones o categorías. Así, y de manera general, el algoritmo de MCC funciona de la siguiente manera para cada usuario:

- Se recibe un elemento de información relacionada con información de comportamiento del usuario.
- Se almacena el elemento en un histórico de elementos. Este histórico tiene un tamaño máximo. De esta manera, si el número de elementos en él alcanza dicho tamaño y llega uno nuevo, el elemento más antiguo es desechado. Esta operación es fundamental porque permite descartar información antigua que ya no es de relevancia para el modelado del comportamiento, permitiendo al MCC actualizar el modelo con información reciente. Si el usuario cambia su comportamiento, este estará recogido en el histórico y tendrá cierta relevancia entre los elementos en él almacenados, otro motivo para limitar el tamaño de dicho histórico.
- En cuanto el histórico es actualizado con un nuevo elemento, un mecanismo de aprendizaje adapta, o genera si no existe, un modelo ajustado al contenido del histórico. Es decir, que se "aprende" el comportamiento implícito en el histórico. Es fundamental que no se produzca un ajuste excesivo al contenido para favorecer la generalización, interpolación y extrapolación en la etapa de predicción.

Lo que permite el algoritmo es mantener un modelo que siempre está actualizado con el contenido del histórico. Así, en la iteración i del algoritmo, es decir, cuando el elemento i ya está disponible, el modelo m_{i-1} de la iteración previa, se adapta levemente para considerar el elemento i y descartar la información del elemento desechado del histórico, dando lugar a una nueva versión actualizada, el modelo m_i. De esta manera se va logrando el aprendizaje gradual y continuo según va llegando la información.

Para materializar el MCC es necesario determinar los elementos de información a utilizar y seleccionar técnicas o mecanismos tanto para la representación de los modelos como para el ajuste de los mismos. Así pues, la información disponible en un primer momento, cuando el usuario envía un trabajo es la relativa a los datos del propio trabajo, como la fecha y hora de envío y la cantidad de recursos solicitados. La segunda pieza de información relevante es la cantidad de recursos realmente utilizados, que se conoce tras la ejecución del propio trabajo. De esta manera, el elemento de información está compuesto de los datos del trabajo, la solicitud de recursos y los valores de uso real de esos recursos. Como se indicó, lo que se busca es un mecanismo que pueda, en función de los datos del trabajo y de la solicitud, predecir o estimar el uso real de recursos, tarea que llevará a cabo el modelo de usuario. En cualquier aproximación al modelado de un comportamiento, ya sea estadística, heurística, etc., se hace necesario un conjunto de datos representativos del citado comportamiento. Estos datos se podrían obtener en tiempo real durante

la ejecución de los comportamientos y/o de registros pasados o ficheros históricos. Para usuarios ya existentes, se obtendrán los datos de los ficheros históricos, mientras que para nuevos usuarios se irán obteniendo durante sus interacciones con el JMS y tras la finalización de la ejecución, los datos de los trabajos irán conformando el histórico. En este histórico de trabajos se tendrán recogidos, por tanto, datos actuales del comportamiento reciente del usuario y no de comportamientos pasados que ya no sean relevantes o lo sean en menor medida. Para la representación de los modelos se utilizan redes de neuronas artificiales (RNA) gracias a sus características de aprendizaje en base a muestras sin necesidad de hacer explícito el conocimiento y a su capacidad de generalización. La RNA que representa un modelo de comportamiento toma como entradas las distintas variables del envío de un trabajo y ofrece como salida una estimación de la cantidad de recursos que en realidad va a necesitar el trabajo. Desde este punto de vista, se puede ver el modelo de usuario como un modulador del comportamiento, pues corrige la petición del usuario optimizándola. Gracias al uso de redes, no es necesario hacer explícito el conocimiento (las reglas, por ejemplo) que puedan seguir los usuarios. Esto sumado a los hechos de que pueda haber reglas no obvias y de que cada usuario sigue las suyas personales, las redes resultan un mecanismo adecuado para este fin. Como mecanismo de ajuste de las RNA se utiliza un algoritmo evolutivo para realizar una búsqueda global frente a la búsqueda local que proporcionan los algoritmos clásicos de entrenamiento de redes como el *back propagation* [125]. La idea principal es mantener una población de redes que sea evolucionada en cada iteración del MCC. Manteniendo la población de redes entre iteraciones del MCC y utilizando unas pocas generaciones en la evolución se logra el pequeño ajuste necesario para adaptarse al nuevo elemento. Todo esto hace que, para la creación y actualización de un modelo, se necesiten los siguientes tres elementos principales:

1. La red de neuronas que se considera la pieza fundamental del modelo y que es la encargada de hacer las estimaciones. Es la red que representa al propio modelo del usuario en un momento dado.

2. Histórico de trabajos, donde está recogido el comportamiento del usuario en una lista ordenada de los n últimos trabajos enviados por el usuario y ejecutados por el sistema.

3. Población de redes de neuronas, de la que, tras ser evolucionada para ajustarse al contenido del histórico, se tomará la mejor red.

Resumiendo, para abordar el objetivo planteado se han utilizado las siguientes ideas básicas:

- El modelo de comportamiento depende de los datos contenidos en el histórico de corto plazo de trabajos, que va cambiando con el tiempo. Un algoritmo evolutivo tiene la responsabilidad de evolucionar la población de redes buscando minimizar el error en las estimaciones sobre los trabajos del histórico. Esto permite un aprendizaje gradual según la información (los trabajos) se va conociendo progresivamente y en tiempo real.

- Para evitar la convergencia prematura de la población de redes a un contenido particular del histórico, se puede controlar el número de generaciones de la evolución que tiene lugar entre actualizaciones del histórico manteniendo la población entre iteraciones. De esta forma, el modelo se adapta rápidamente a los cambios que puedan surgir en el comportamiento del usuario.

Una vez decidido y descrito el mecanismo de predicción se debe establecer el proceso para construir y actualizar los modelos. Concretando la forma de operar, el proceso de creación de un modelo resulta muy sencillo en esencia. Asumiendo que no existen datos históricos de trabajos, para cada usuario nuevo se crea, inicialmente, un histórico de tamaño máximo t vacío y una población de redes de neuronas aleatoria. Se toma una de las redes al azar como red principal de la población. Una vez con el modelo creado, cada vez que termina un trabajo y sus datos de ejecución pasan a estar disponibles es posible actualizar el modelo. Para esta actualización se siguen los siguientes pasos:

1. Cuando un trabajo del usuario *A* termina, este se añade, junto con los valores reales de uso de recursos, al principio del histórico de trabajos de *A*, lo que hace que el resto de trabajos, si existen, se desplacen en el histórico una posición expulsando al último, que además es el más antiguo, es decir, se utiliza una política *first in, first out* (FIFO). También sería posible el uso de otra estrategia de reemplazo como las explicadas en [124], como sustituir a aquel que aporte menos información al conjunto y no al más antiguo.

2. Con el histórico de trabajos renovado, se evoluciona durante unas pocas generaciones la población de redes de neuronas que predicen el uso real de recursos de los trabajos del histórico.

3. Tras la evolución, la red que resulte mejor es la que se adopta como la red principal del modelo de usuario, y por tanto, es esta la que se utiliza para predecir el uso de recursos de nuevos trabajos.

4. La población de redes de neuronas, que está adaptada (aunque no en exceso, manteniendo la capacidad de generalización) a los contenidos concretos del histórico, no se desecha, pues será utilizada como población inicial para futuras actualizaciones.

Este proceso se repite cada vez que termina un trabajo, manteniendo así actualizado en todo momento el modelo de cada uno de los usuarios. La figura 5.3 muestra los tres componentes de los modelos de usuario y como interactúan entre sí para la actualización de modelos.

Una alternativa para la inicialización de un modelo para un nuevo usuario que, por tanto, no tiene trabajos en el histórico, es la utilización del modelo de otro usuario. De esta manera, si los usuarios se pareciesen entre sí, la adaptación del modelo al nuevo usuario sería más rápida que partiendo de un modelo vacío. Otra posibilidad es la de utilizar un prototipo dado o, simplemente, no utilizar las estimaciones del modelo en las primeras iteraciones, hasta que haya datos suficientes, y utilizar tal cual los valores proporcionados por el usuario.

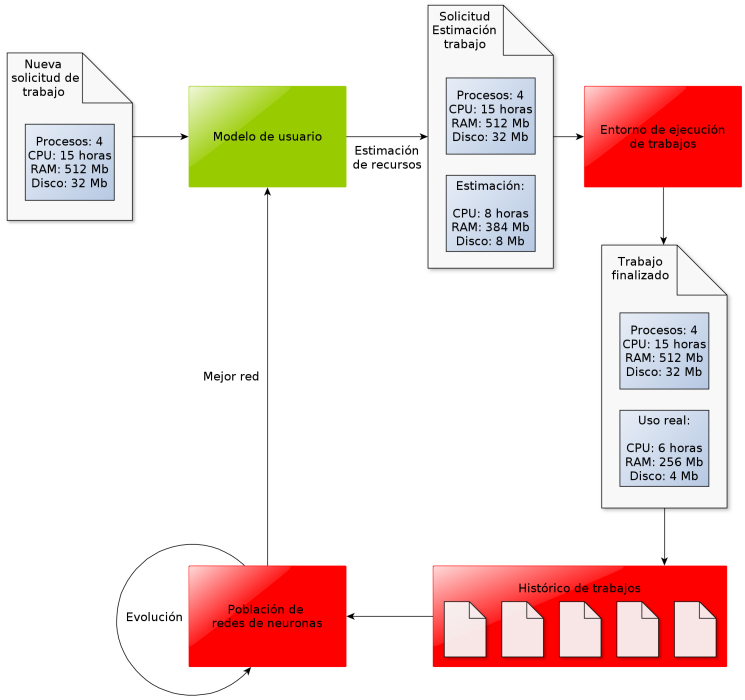

Figura 5.3: Operación general del modelado de usuarios

5.3.2. Implementación

Se presentan a continuación algunos detalles de una implementación concreta de los objetos más relevantes del subsistema de modelado de usuarios, haciendo especial hincapié en los algoritmos utilizados para la ejecución de la red de neuronas y para su obtención mediante evolución. Esta implementación es la que ha sido utilizada para realizar las pruebas comentadas más adelante.

5.3.2.1. La red de neuronas

Se utilizan redes *feed forward* en configuración de *perceptrón* multi-capa debido a su potencia como mecanismo universal de aproximación [126]. Esta implementación se ha centrado en la estimación del uso de tiempo de sistema por un trabajo (es decir, el tiempo que transcurre desde que el trabajo se pone en ejecución hasta que termina y sale de los nodos de ejecución), conocido como (*walltime*). La red utilizada posee cuatro entradas y una salida, que se detallan a continuación junto con su codificación. Tiene dos capas ocultas de tres neuronas cada una, lo que hace un total de veinticuatro pesos sinápticos.

Tanto las entradas como las salidas se codifican con valores reales entre 0 y 1,

y todas representan datos del envío de un trabajo y de la solicitud de recursos. Las entradas son las siguientes:

Hora del envío Es un conjunto discreto de valores que representa la hora del día (h) a la que se ha enviado el trabajo al sistema. Como es lógico, h varía entre 0 y 23. La codificación que se ha seguido cumple la siguiente fórmula:

$$i_1 = \begin{cases} 0 & \text{si } 8 \leq h < 12 \\ 1/4 & \text{si } 12 \leq h < 16 \\ 2/4 & \text{si } 16 \leq h < 19 \\ 3/4 & \text{si } 19 \leq h < 22 \\ 1 & \text{en otro caso} \end{cases} \tag{5.1}$$

La idea tras esta codificación es la de diferenciar las primeras horas de las últimas tanto en una jornada de mañana como de tarde, así como las horas de la noche.

Día de la semana La segunda entrada codifica el día de la semana (d, donde 1 representa el lunes y 7 el domingo) diferenciando entre días laborables (lunes a viernes) y fines de semana:

$$i_2 = \begin{cases} 0 & \text{si } 1 \leq d \leq 5 \\ 1 & \text{en otro caso} \end{cases} \tag{5.2}$$

Tiempo real solicitado Es el *walltime*, w, y codifica la cantidad de tiempo, en segundos, solicitado para la ejecución del trabajo, independientemente del número de procesadores a usar. La codificación se hace respecto de un máximo de tiempo M establecido por el centro de computación:

$$i_3 = \frac{w}{M} \tag{5.3}$$

Número de procesadores La última entrada codifica el número de procesadores solicitado, n, por el usuario, parámetro que tiene repercusión en el tiempo de ejecución del trabajo (debido, por ejemplo, al tiempo usado por los protocolos de comunicación y sincronización entre los procesos de un trabajo en paralelo). El máximo de tiempo que se puede solicitar estará determinado por el número de procesadores solicitados. El número máximo de procesadores P también es determinado por el centro.

$$i_4 = \frac{n-1}{P-1} \tag{5.4}$$

La salida de la red, o_1 corresponde con el tiempo estimado de ejecución del trabajo. El valor w_e, tiempo estimado del trabajo, se encuentra en el mismo dominio que la entrada de tiempo solicitado, i_3, y por tanto, la decodificación del valor se realiza teniendo en cuenta el valor máximo de tiempo que se puede solicitar, M, mediante la siguiente ecuación:

$$w_e = o_1 \cdot M \tag{5.5}$$

Los pesos sinápticos de la red caen dentro de un rango definible mediante un parámetro. En general ha sido utilizado el rango de -10 a 10. Como función de transferencia para la red se utiliza la función *sigmoidal* definida por la ecuación $y = \frac{1}{1+e^{-x}}$.

Como resumen se han recogido la estructura y diferentes características de la red utilizada en la tabla 5.1. Algunos de estos parámetros se han determinado empíricamente en un proceso de barrido de los diferentes valores que pueden adoptar. Se han probado redes con una y dos capas ocultas. Para cada capa oculta se han barrido valores entre dos y seis neuronas. Se ha considerado que, dada la función de transferencia típica utilizada, el rango de los pesos sinápticos de -10 a 10 era adecuado, ya que en esos extremos la función toma valores muy cercanos a 0 y a 1, respectivamente.

Número de neuronas de entrada	4
Número de neuronas de la primera capa oculta	3
Número de neuronas de la segunda capa oculta	3
Número de neuronas de salida	1
Rango de los pesos sinápticos	De -10 a 10
Rango de la codificación de entradas y salidas	De 0 a 1
Función de transferencia	$1/(1+e^{-x})$

Tabla 5.1: Estructura y características de la red de neuronas

5.3.2.2. El algoritmo evolutivo

Se ha elegido como algoritmo evolutivo un algoritmo genético debido no solo a su sencillez, sino a su amplia difusión en toda clase de problemas de búsqueda y optimización, por lo que resultan un mecanismo profundamente probado.

La función del algoritmo genético utilizado en este subsistema es, como ya se ha dicho, la obtención de la red de neuronas. Para ello, el algoritmo evoluciona una población de redes para que se adapten al histórico de trabajos, es decir, que cometan el menor error posible en la estimación para todos los registros almacenados en dicho histórico. Por tanto, la función de *fitness* o de calidad utilizada para evaluar a los individuos, que son cada una de las redes, a lo largo de la evolución, devuelve un valor indicativo del error obtenido en la estimación. El algoritmo genético utilizado aquí sigue un esquema canónico.

Como se puede comprobar, el proceso de evolución se lleva a cabo a través de una serie de generaciones, cuyo número viene determinado por un parámetro. Un paso previo al comienzo de la evolución es la determinación de la calidad (el error de estimación, en este caso) de todos los individuos de la población. Por cada generación, existe un número de pasos que se realizan para llevar a cabo la evolución, que son los pasos clásicos de evolución: selección de individuos, reproducción, evaluación de los nuevos individuos y reemplazo de viejos individuos por los nuevos.

El proceso de selección elige los individuos más adecuados para llevar a cabo la reproducción. Para generar una nueva población se usa la técnica del torneo con una ventana de dos individuos. Esto alivia la presión evolutiva, favoreciendo la exploración a costa de hacer que la solución tarde más en encontrarse. Al usar esta forma de selección se descarta por lo menos el peor individuo, que nunca pasa a reproducirse con otro evitando individuos poco útiles.

Tras la selección comienza la reproducción destinada a originar nuevos individuos. Se ha optado por una estrategia muy frecuente en la reproducción, el crecimiento cero, con lo que de dos padres nacen dos hijos. Los individuos se cruzan entre sí con una probabilidad p_c. Se utiliza un operador de cruce clásico, que consiste en elegir aleatoriamente dos puntos de corte, intercambiándose entre los padres el conjunto de cromosomas que queda entre los puntos de corte, como se puede observar en la figura 5.4.

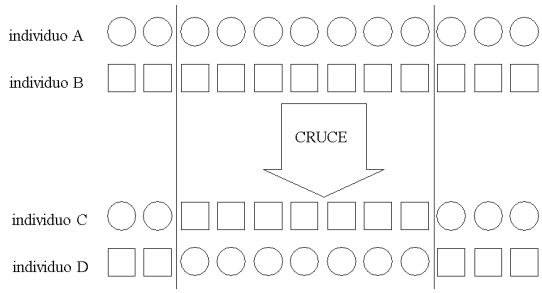

Figura 5.4: Operación de cruce del algoritmo genético

Una vez finalizados los cruces, los genes pueden sufrir mutaciones con una cierta probabilidad p_m. La mutación consiste en aplicar una pequeña alteración a un gen seleccionado al azar. Con esto se promueve la diversidad genética.

La codificación de los individuos resulta sencilla. Es una colección de valores reales representando los pesos de las conexiones y los sesgos (*bias*), que también se evolucionan, de las redes de neuronas. De esta forma, los cromosomas tendrán un número de genes n_g que se calcula mediante

$$n_g = n_c + n_b = n_i \cdot n_1 + n_1 \cdot n_2 + n_2 \cdot n_o + n_1 + n_2 + n_o \tag{5.6}$$

siendo n_c, n_b, n_i, n_1, n_2 y n_o respectivamente, el número de conexiones, el número de *bias*, el número de neuronas de la capa de entrada, de la primera capa oculta, de la segunda y de la de salida.

La nueva población se evalúa haciendo uso de la función de calidad, tras lo cual tiene lugar el reemplazo, que consiste en sustituir un gran porcentaje r de padres, los peores, con los mejores hijos, en la misma cantidad. Los individuos destituidos se eliminan.

Con estos pasos termina una generación volviendo de nuevo al proceso de selección. La evolución termina al alcanzarse un número concreto de generaciones

o cuando se alcanza un valor umbral de error, ambos definidos a través de sendos parámetros.

Previamente a la evaluación de cada individuo mediante la función de calidad es necesario realizar un proceso de decodificación del individuo que implica calcular su fenotipo a partir de su genotipo. Una vez realizado este paso puede comenzar la evaluación, donde la función de calidad actúa de la siguiente manera:

1. Para cada trabajo i existente en el histórico se hace una estimación de los recursos utilizados a partir de los valores de la solicitud. Para ello: aplica una función $f(s_i, u_i, e_i)$ a los parámetros tiempo (*walltime*) solicitado, tiempo utilizado y tiempo estimado del trabajo i, que devuelve un valor indicativo del error de estimación para ese trabajo.

2. Realiza una suma ponderada con los valores de error provenientes del paso anterior, es decir, los valores de error de estimación para cada uno de los n trabajos del histórico. Se hace de manera que se prima a los trabajos más recientes sobre los más antiguos y se logra por medio de una sencilla función exponencial. La función utilizada es:

$$\epsilon = \frac{\sum_{i=0}^{n-1} (2/3)^i \cdot f(s_i, u_i, e_i)}{\sum_{i=0}^{n-1} (2/3)^i} \tag{5.7}$$

3. Devuelve este valor de error ϵ, para que pueda ser asociado al individuo que representa la red evaluada.

La función $f(s,u,e)$ (en la figura 5.5 se puede ver representada la función $1 - f(s,u,e)$, que se ha hecho así por claridad, donde s es $solicitado$, u, $usado$ y e, $estimado$) pretende penalizar más una estimación por defecto respecto del tiempo real utilizado que una estimación por exceso, pues se considera la primera como un falso negativo. Hay que tener en cuenta que, normalmente, los sistemas de HPC evitan que un trabajo utilice más recursos que los que haya solicitado el usuario, deteniéndolo. Además, una subestimación repercute negativamente en la planificación en el sentido de que al considerar un trabajo más corto de lo que es, puede pasar a ejecución antes que otros, retrasando así la hora de entrada a ejecución de estos, que pueden ser muchos, haciendo aumentar el descontento general. En caso contrario, una sobreestimación puede hacer retrasar ese trabajo lo que causa el descontento en un solo usuario. Si la estimación de la red coincide con el valor real, el error es nulo, es decir, es una estimación perfecta.

En esta forma de operar se usa un algoritmo genético sobre una población de redes de neuronas tratando de encontrar una red que se ajuste al histórico. Es necesario un control en el número de generaciones con el fin de evitar una convergencia excesivamente rápida, haciendo perder a la red capacidad de generalización, y posibilitando el aprendizaje gradual. Es decir, un número muy alto de generaciones derivaría en una población de redes de neuronas excesivamente adaptada al contenido del histórico de trabajos haciendo que, al ser utilizadas para realizar estimaciones de recursos con trabajos que difieran de los del histórico, la predicción fuese muy poco precisa. Además, esto dificultaría el aprendizaje gradual, ya

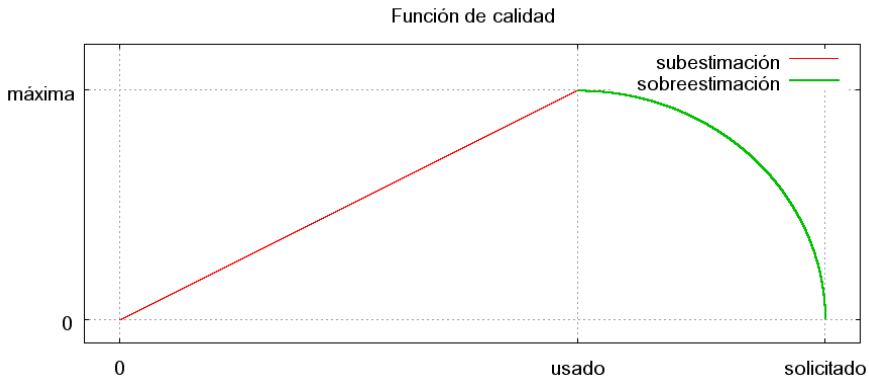

Figura 5.5: Función de evaluación de una estimación

que al introducir un nuevo trabajo en el histórico habría que volver a evolucionar bastante la población para adaptarse de nuevo al histórico, un histórico para el que la población de redes de neuronas se adapta muy bien a todos los trabajos excepto a uno, el nuevo, que les resultará especialmente difícil de estimar. Si el número de generaciones es bajo y la población se adapta un su justa medida al histórico, se mantiene la capacidad de generalización a la hora de estimar recursos para un nuevo trabajo y, además, al introducir los datos de este nuevo trabajo en el histórico, al haber mantenido la población esa capacidad de generalización, solo habría que aplicar de nuevo un número bajo de generaciones para aprender esta nueva información que, presumiblemente, ya era predecible.

5.4. Planificando con solicitudes optimizadas

Una vez establecido el mecanismo de optimización de solicitudes hay que determinar cómo se utiliza durante el ciclo de vida de envío de trabajos al HPC, la planificación y la ejecución de los mismos y cómo y cuándo se disparan los procesos principales del SOS, que son la explotación de los modelos para realizar la estimación del uso de recursos y la actualización de dichos modelos. La figura 5.6 refleja el citado ciclo de vida donde, nuevamente, se presentan en azul las acciones realizadas por el administrador, en verde las realizadas por los usuarios y en rojo las realizadas por los sistemas propuestos en esta tesis. En gris se puede apreciar la acción de ejecución de trabajos, por parte de los nodos de ejecución.

La tarea principal del administrador es la declaración de políticas tal y como se vio en el capítulo 4. Con las políticas establecidas, los usuarios se conectan al sistema y realizan sus envíos de trabajos a través de solicitudes. Las solicitudes no llegan tal cual al planificador, sino que pasan por una etapa de optimización en la que se hace uso de los modelos de usuario correspondientes. Los modelos de usuario realizan estimaciones del uso de recursos que se entregan al planificador, que dispara

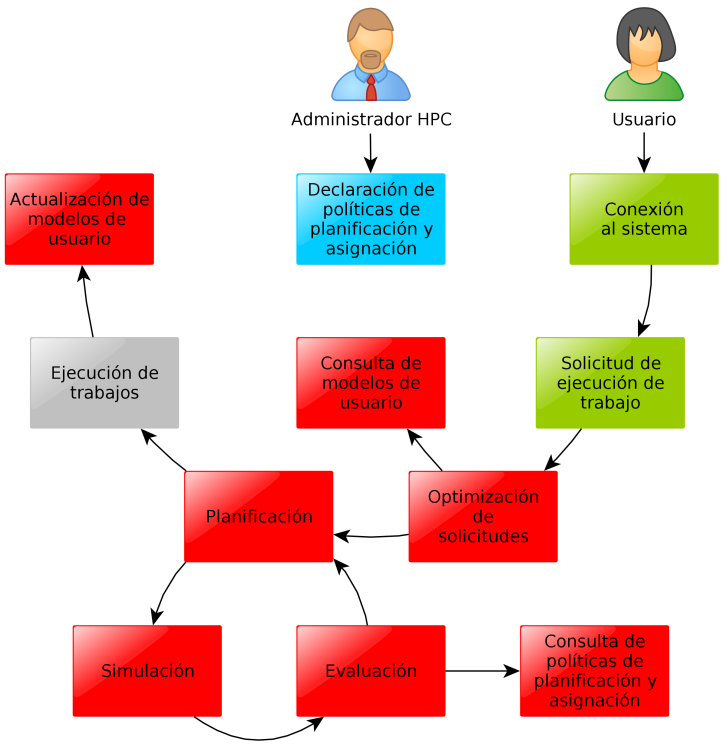

Figura 5.6: Eventos en el proceso de planificación con solicitudes optimizadas

un proceso evolutivo para realizar la planificación. Una vez concluida, algunos trabajos serán puestos en ejecución. Cuando estos trabajos terminen, nuevos datos de ejecución estarán disponibles para formar parte de los históricos de trabajos, y por tanto de las memorias a corto plazo, de los modelos de usuario correspondientes y tendrá lugar el proceso de actualización de modelos, descrito en la sección 5.3.1, y cuyo esquema se puede ver en la figura 5.3. Como se puede observar, las etapas necesarias para la optimización de solicitudes se introducen en el proceso general de manera natural y transparente al usuario.

A continuación se presenta un capítulo con un estudio con diversos experimentos que muestran la viabilidad del SOS para finalizar con un resumen acerca de los resultados obtenidos para dicho sistema.

5.5. Pruebas y resultados

Para probar el subsistema de modelado de usuarios, se han tomado ficheros de *log* proporcionados por el Centro de Supercomputación de Galicia con datos reales

relativos a la ejecución de procesos de todo un año en el supercomputador SVG [19]. Se implementó un analizador sintáctico para extraer los datos relativos a los usuarios de los ficheros citados. Además, se han creado algunos usuarios ficticios con los que poder comprobar posteriormente el funcionamiento del planificador, y sobre los que también se comprobará el comportamiento del subsistema de modelado.

5.5.1. Usuarios reales

Las pruebas realizadas con tres usuarios reales, usuarios *A*, *B* y *C* en adelante, han sido llevadas a cabo con diferentes tamaños de histórico de trabajos y número de generaciones por evolución en la etapa de actualización de los modelos. Esto permite comprobar el comportamiento del SOS en base a distintos parámetros. Otros parámetros, que se han mantenido constantes a lo largo de todas las pruebas, relativos tanto a la evolución como a la red de neuronas, se presentan en la tabla 5.2.

Parámetro	**Valor**
Número de individuos	50
Rango de pesos sinápticos y *bias*	De -10 a 10
Participantes en la selección por torneo	2
Probabilidad de cruce	0.65
Probabilidad de mutación	0.08
Padres que permanecen tras una generación	1 %

Tabla 5.2: Parámetros constantes durante las pruebas de modelos de usuario

Las probabilidades de cruce y mutación, así como el porcentaje de padres que permanece en la población tras cada evolución y la ventana de individuos para la selección por torneo se han elegido de manera empírica mediante una batería de pruebas con barrido de los parámetros resultando en el conjunto de valores típicos.

Los parámetros que más van a influir en los modelos son dos. Uno de ellos es el tamaño del histórico de trabajos del usuario, que es un indicativo de la memoria que va a tener el modelo. El otro es el número de generaciones que tienen lugar en una evolución. Para cada usuario se han realizado múltiples pruebas combinando varios posibles valores para estos dos parámetros.

Para la realización de estas pruebas se ha simulado el lanzamiento de trabajos por parte de los usuarios tal y como están registrados estos lanzamientos en los ficheros de *log* utilizados. Se han ejecutado cinco baterías de pruebas en las que cada prueba se ejecuta cinco veces) con el fin de obtener mejores valores de medidas estadísticas como la media de error y la varianza. Esto es así porque en el proceso de modelado tienen cabida elementos aleatorios, como la inicialización de la población de redes o los cruces y mutaciones durante la evolución. Se parte en todos los casos de un modelo de usuario vacío, es decir, sin trabajos en el histórico. El proceso, que

involucra la estimación de recursos y la aplicación del algoritmo MCC, para cada usuario ha sido el siguiente:

1. Se lee un trabajo del fichero de *log*.
2. Se estima la duración del trabajo con el modelo de usuario.
3. Con la ayuda de la misma función de calidad utilizada para el proceso de actualización de los modelos (figura 5.5) y el valor real de duración del trabajo se calcula el error de la estimación.
4. Se incorporan los datos a un fichero de resultados.
5. Con los datos de uso real de recursos se actualiza el modelo de usuario en cuestión mediante el MCC.
6. Se vuelve al paso 1 hasta acabar con todos los trabajos.
7. Se obtienen las gráficas.
8. Se hacen los cálculos del error medio y la varianza del error con los datos del fichero obtenido en el paso 4.
9. Se elimina el modelo de usuario y se vuelve al paso 1, repitiendo todo el proceso 4 veces más.

Para cada uno de los usuarios presentados a continuación se muestran gráficas de los resultados de estimación de los modelos, gráficas de evolución del error durante la actualización del mejor modelo y una tabla comparativa del error de cada modelo con diversas configuraciones. La gráficas de resultados muestran, para cada uno de los trabajos lanzados, el tiempo realmente usado y el tiempo estimado por el modelo. En el eje de ordenadas se indica la escala del tiempo en segundos, mientras que el eje de abscisas representa los trabajos. Estas gráficas corresponden a los tres mejores modelos encontrados para los casos en que el histórico toma valores de 5, 20 y 50 trabajos.

Usuario A

El usuario A tiene un patrón muy claro en sus pautas de comportamiento, que es el hecho de solicitar casi siempre el máximo de tiempo disponible, que son 200 horas, es decir, 720.000 segundos. Además, cuenta con las siguientes características:

- El máximo tiempo real de ejecución de sus trabajos se encuentra en torno a las 110 horas.
- Se dan bastantes oscilaciones.
- Hay trabajos lanzados los fines de semana.

- Siempre se solicita un único procesador.

En la figura 5.7 se puede observar el comportamiento de los tres mejores modelos encontrados con los distintos valores de tamaño de histórico de trabajos (5, 20 y 50), donde se ve como se ajusta bien al tiempo realmente usado.Al observar detenidamente las imágenes se aprecia como un tamaño de histórico mayor repercute en una adaptación más lenta a los cambios de comportamiento, por ejemplo, en la bajada del tiempo de ejecución en torno al trabajo 330 o en las grandes oscilaciones entre los trabajos 50 y 100.

La figura 5.8, que es un detalle de la gráfica con histórico de 5 trabajos y 32 generaciones, permite comprobar más de cerca el comportamiento del modelo. Teniendo en cuenta que se penaliza más la subestimación que la sobreestimación, como se puede ver en la gráfica de la figura 5.5, el modelo tiende a estimar para el trabajo n un valor cercano al real del trabajo $n-1$ si este era mayor, como se aprecia con los trabajos 57, 63 y 82, resultando más conservador en caso contrario, aspecto deseable, ya que no resulta adecuado adaptarse demasiado a las oscilaciones, cosa que puede causar más errores de estimación de los deseados, tanto por exceso como, y sobre todo, por defecto. Este fenómeno se aprecia entre los trabajos 91 y 95, por ejemplo, donde, si bien es cierto que la estimación difiere (por encima) en algunas horas del valor real, resulta mucho más acertado que el valor de la solicitud del usuario. Cabe destacar la precisión en la estimación desde el trabajo 65 al 80, con poca sobreestimación, así como desde el 85 al 90, que a pesar de caer mínimamente por debajo del tiempo real utilizado, se pueden considerar estimaciones muy precisas, teniendo en cuenta, además, que no parece haber ninguna tendencia en los trabajos anteriores o si la hay, corresponde con algún patrón local de comportamiento, donde quizá sea mas claro para los picos dibujados por los trabajos entre el 63 y el 72. Sin duda pasan a jugar aquí las otras entradas de la red no reflejadas en las gráficas, como la hora o el día de envío del trabajo.

Para el mejor modelo encontrado en este batería de pruebas, que es aquel que utiliza un tamaño de histórico de 20 trabajos y 1 generación por evolución, se muestra en la imagen 5.9 cómo evolucionan, con el paso de las iteraciones, el error medio de la población de redes neuronales y el error de la mejor red. La primera observación que se hace, y teniendo en cuenta que las pruebas parten de un modelo de usuario con histórico de usuarios vacío, es la evolución que sufre la población de redes, que poco a poco se adaptan al comportamiento del usuario según el histórico se va poblando. Así, durante los primeros 20 trabajos, el error medio de la población se va decrementando hasta el momento en que el histórico está lleno. A partir de ese momento, el error permanece más estable. Entre el trabajo 20 y el 200, el usuario A mantiene un cierto patrón de comportamiento que queda reflejado en que, a medida que se actualiza el modelo, la media de la población sigue una tendencia de decremento del error, adaptándose al comportamiento cada vez más. A partir del trabajo 200 y, sobre todo, del 380 el comportamiento se vuelve más irregular haciendo que la población oscile pero que mantenga una cierta media de error, mientras que, de media, el mejor modelo empeora levemente.

Se ha calculado una tabla de error a partir de las baterías de pruebas, para todas las estimaciones de trabajos del usuario. No se tuvieron en cuenta los primeros

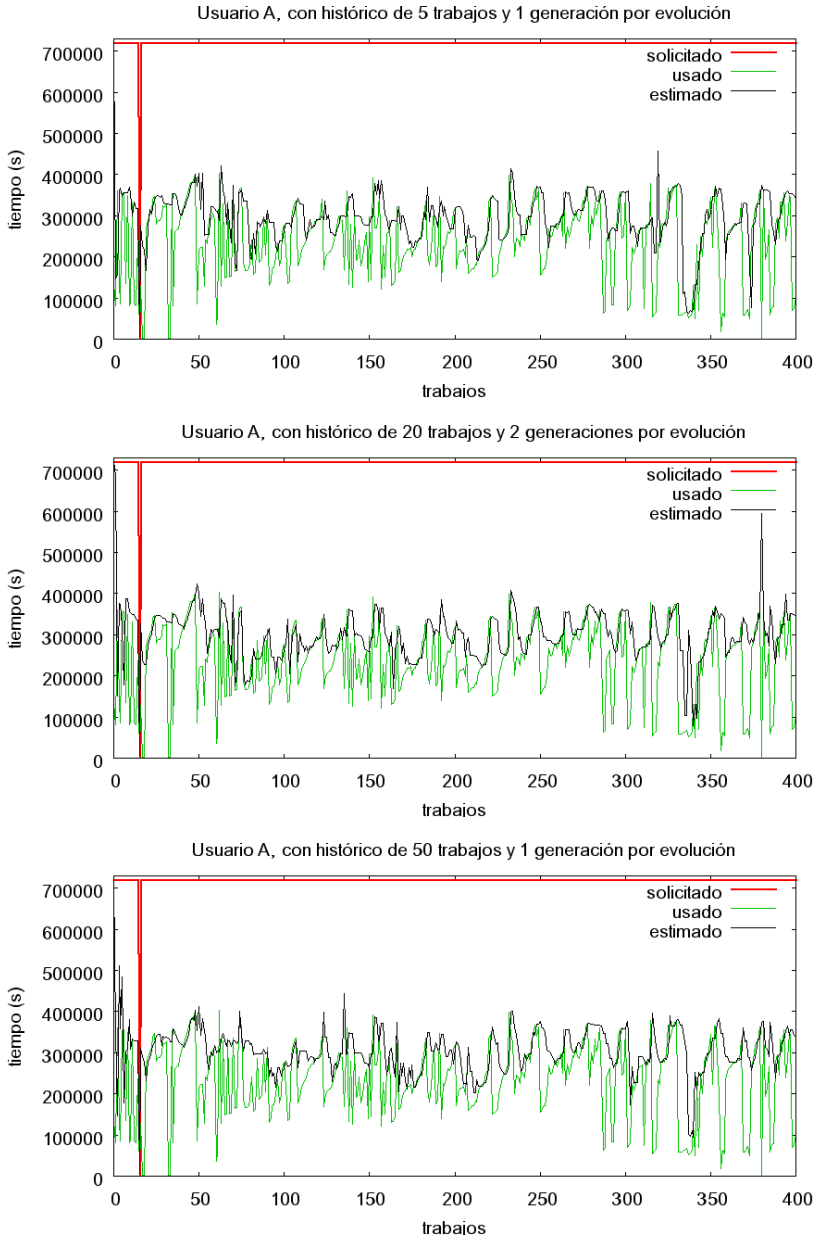

Figura 5.7: Pruebas del modelo del usuario A

trabajos de forma que el modelo siempre tuviera lleno el histórico, es decir, se ha prescindido de las primeras estimaciones. Las columnas mantienen la información del número de generaciones por evolución, mientras que el tamaño del histórico

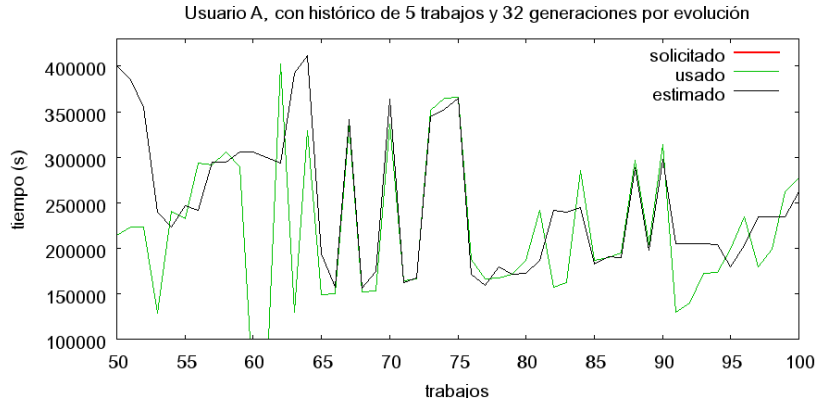

Figura 5.8: Detalle de las pruebas del modelo del usuario A

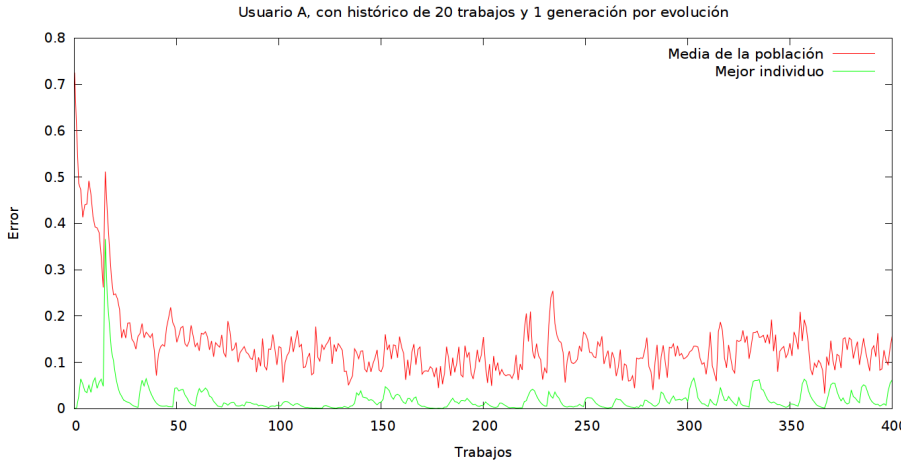

Figura 5.9: Evolución del mejor modelo para el usuario A

de trabajos se ha organizado por filas. El caso concreto de la columna etiquetada con cero generaciones indica que no ha habido evolución tras la incorporación de trabajos al histórico, utilizándose la población existente sin modificaciones. Cada casilla de la tabla contiene dos valores, donde el superior indica la media del error cometido y el inferior, la varianza. Se han resaltado con fondo gris aquellas casillas con el mejor valor de error para un tamaño de memoria dado y usado letra cursiva en aquellas que tienen el mejor valor para un número concreto de generaciones. Además, se ha subrayado el valor de error más bajo logrado. A la vista de la tabla 5.3, es evidente que para este usuario, el mejor número de generaciones a usar en el MCC para la actualización del modelo es 1 con independencia del tamaño del histórico. Respecto de este, y como muestran las celdas en cursiva, el número de

trabajos a utilizar se puede concluir entre 20 y 40. El mejor modelo ha resultado ser el de 1 generación y 20 trabajos.

H/G	0	1	2	4	8	16	32
5	0.08659	0.05782	0.06104	0.06648	0.06981	0.07900	0.08122
	0.02072	0.01497	0.01548	0.01812	0.01952	0.02515	0.02610
10	0.06313	0.04953	0.05133	0.05184	0.05742	0.06235	0.06736
	0.01142	0.00957	0.00950	0.00971	0.01217	0.01446	0.01686
20	0.05384	*0.04308*	*0.04548*	0.04904	*0.05083*	0.05641	0.06230
	0.00762	*0.00552*	*0.00634*	0.00836	*0.00833*	0.01120	0.01397
30	*0.04860*	0.04526	0.04679	0.04892	0.05415	0.05770	*0.06081*
	0.00779	0.00588	0.00671	0.00793	0.01050	0.01105	*0.01286*
40	0.06791	0.04525	0.04828	*0.04884*	0.05436	*0.05571*	0.06307
	0.00870	0.00639	0.00741	*0.00774*	0.00913	*0.00996*	0.01405
50	0.06248	0.04471	0.04748	0.04979	0.05470	0.05857	0.06217
	0.00914	0.00523	0.00739	0.00763	0.00981	0.01117	0.01254
60	0.06414	0.04616	0.04712	0.04853	0.05540	0.05671	0.06470
	0.01130	0.00710	0.00626	0.00671	0.01061	0.00998	0.01386

Tabla 5.3: Error medio y varianza de estimaciones del modelo del usuario A

Mientras que, de media, el error cometido por el usuario en la solicitud es de 476.022,94 segundos por trabajo (calculado para una ventana de 100 trabajos y con el histórico de trabajos lleno), el error medio en las estimaciones ha sido de 62.818,76 por trabajo. Así, el modelo comete, de media, un error de un 13,20 % en sus estimaciones respecto del error cometido por el propio usuario en su solicitud. Desde el punto de vista del tiempo solicitado, el usuario solicita 720.000 segundos por trabajo y consume una media de 243.977,06 lo que implica un error del 195,09 %. El modelo estima una media de 294.119,58, y teniendo el cuenta que el error de estimación puede ser por exceso o por defecto, el error cometido por el SOS en este caso es del 25,75 %. La tabla 5.4 recoge estos datos.

Medida	**Valor**
Tiempo medio usado	243.977,06
Tiempo medio solicitado por el usuario	720.000,00
Tiempo medio estimado por el modelo	294.119,58
Error medio de las solicitudes	476.022,94
Error medio de las estimaciones	62.818,76
% de error de las estimaciones frente a las solicitudes	13,20
% de error de las solicitudes	195,09
% de error de las estimaciones	25,75

Tabla 5.4: Medidas de resultados para el usuario A

Usuario B

El usuario B varía respecto del A en cuanto al tiempo solicitado, que tiene algunos cambios, como se aprecia en las gráficas. Características:

- Es bastante irregular en el tiempo real de uso, que en ocasiones se acerca al máximo solicitado.
- Otras veces, los trabajos se ejecutan durante muy poco tiempo.
- Casi ningún trabajo es lanzado el fin de semana.

La figura 5.10 muestra las gráficas de resultados para los tres mejores modelos encontrados para tamaños de histórico 5, 20 y 50. La figura 5.11 muestra la evolución del error medio de la población de redes y de la mejor red para el mejor modelo encontrado para el usuario B, que es el de 50 trabajos como tamaño de histórico y 1 generación por evolución. Al igual que sucede en el caso del usuario A, mientras el histórico no está lleno la población se va adaptando poco a poco con lo que su error decrementa hasta el trabajo 50. Alrededor del trabajo 35, los trabajos del usuario comienzan a durar poco tiempo. Al no estar lleno el histórico, este cambio de comportamiento brusco confunde al modelo y a la población, que sufren un alto incremento en el error que se subsana según siguen entrando trabajos en el histórico. A partir del trabajo 50, el modelo se estabiliza y la mejor red resulta muy buena a pesar de que la población sufre algunas subidas en el error, sobre todo cerca del trabajo 100, donde el comportamiento cambia en lo referente a tiempo solicitado. Desde ahí y hasta el trabajo 170 la población se estabiliza algo más notándose una cierta tendencia a disminuir el error medio. Un nuevo cambio en el comportamiento, visible en las duraciones de los trabajos, causa de nuevo una pequeña crisis en el modelo, que pierde precisión que recupera, no obstante, con unas pocas iteraciones más. Cabe destacar que para un usuario irregular como este, se consiguen unos resultados muy aceptables, lo que invita a pensar que usuarios que se ciñan más a patrones de comportamiento concretos serán mejor modelados.

De la tabla de errores 5.5 se concluye que el usuario B necesita un modelo con un tamaño de histórico en torno a los 50 o 60 trabajos. Puede ser debido a que necesita información suficiente para capturar los citados cambios de comportamiento y que los 20 trabajos que necesitaba el usuario A se quedarían algo cortos, un tamaño más indicado para usuarios más estables. El número de generaciones para la actualización de modelo no se aprecia tan claramente, pareciendo, aunque de manera no muy concluyente, que existe una cierta tendencia a necesitarse más generaciones cuanto más grande sea el histórico. No obstante, el mejor modelo ha resultado el de 1 generación y 50 trabajos de tamaño de histórico.

La tabla 5.6 recoge distintas medidas sobre los resultados obtenidos con el mejor modelo para el usuario B, donde se puede observar que el error cometido por el modelo en las estimaciones es del 70,03 % frente al 566,18 % proveniente de las solicitudes del propio usuario.

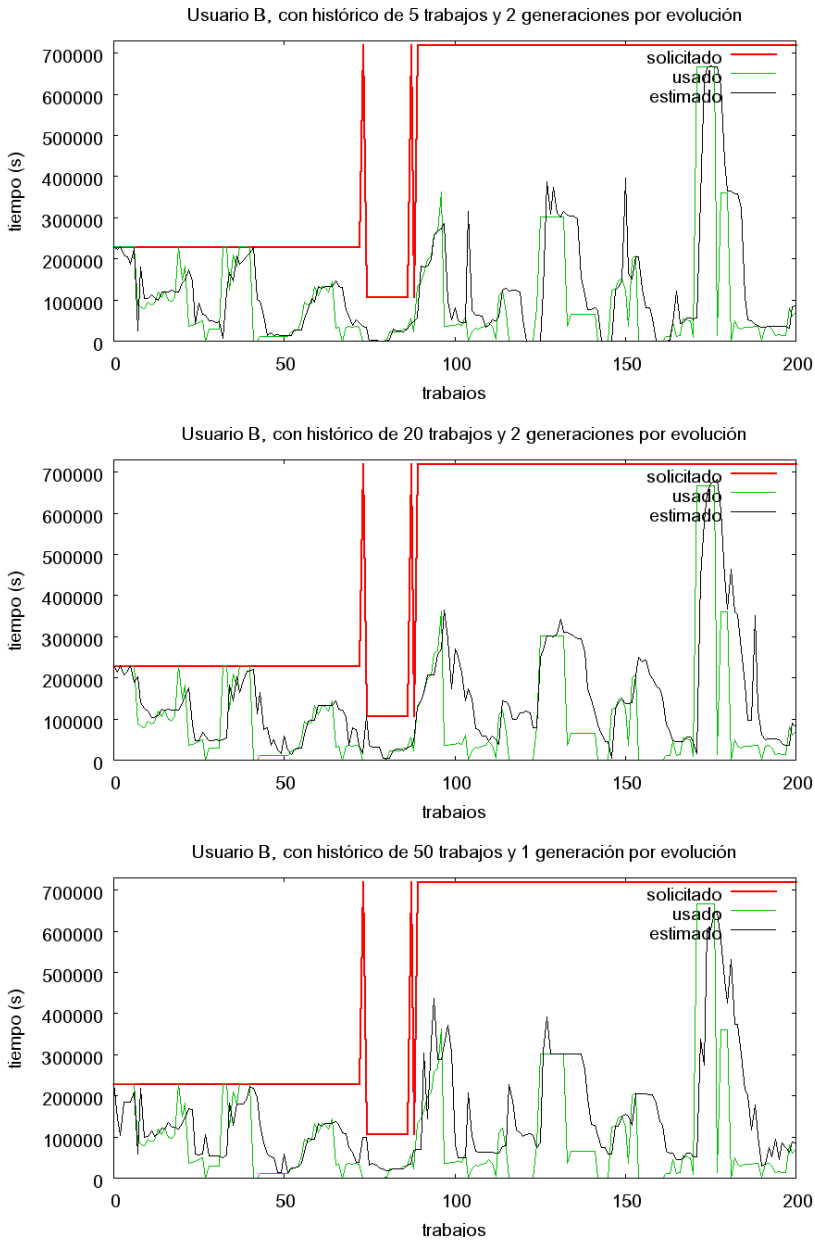

Figura 5.10: Pruebas del modelo del usuario B

Usuario C

Este usuario casi siempre solicita el máximo de tiempo. Tiene un comportamiento aparentemente irregular hasta el trabajo número 200, a partir del cual, el tiempo

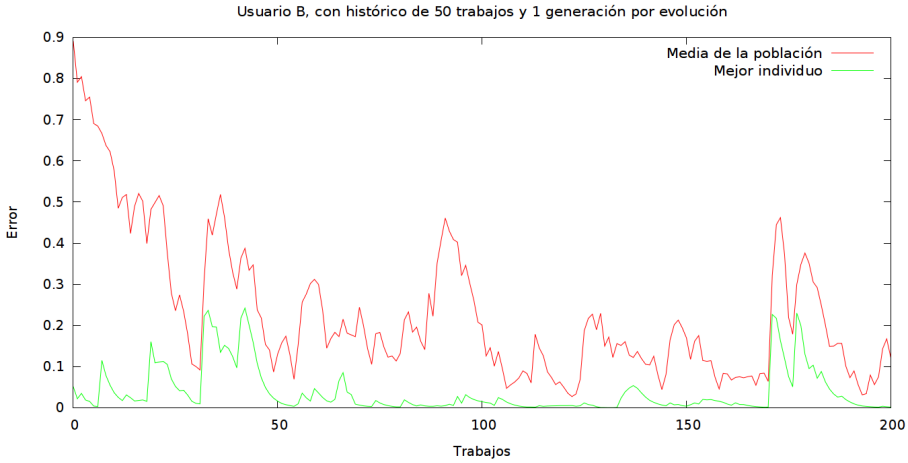

Figura 5.11: Evolución del mejor modelo para el usuario B

H/G	0	1	2	4	8	16	32
5	0.13554 0.06140	0.13009 0.05951	0.12459 0.05729	0.12836 0.05923	0.12683 0.06335	0.12896 0.06733	0.13347 0.06909
10	0.12687 0.05034	0.11721 0.04549	0.12215 0.05321	0.12148 0.05381	0.12623 0.06226	0.12608 0.05962	0.12616 0.06305
20	0.11891 0.04714	0.10922 0.03988	0.10746 0.04383	0.11732 0.04799	0.11337 0.04753	0.12479 0.05894	0.11839 0.05422
30	0.10582 0.03957	0.11160 0.04524	0.11324 0.04636	0.11321 0.04992	0.12045 0.05418	0.12212 0.05769	0.11351 0.05312
40	0.11118 0.05035	0.11229 0.04748	0.11181 0.04849	0.10916 0.04944	0.10807 0.04782	0.10909 0.05110	0.10917 0.05173
50	*0.10868* *0.04519*	*0.09485* *0.03439*	*0.10097* *0.04287*	0.11271 0.05133	0.11185 0.05205	0.11076 0.05123	0.11340 0.05468
60	0.11404 0.05230	0.10625 0.04675	0.10741 0.04975	*0.09635* *0.04252*	*0.10014* *0.04627*	*0.09600* *0.04556*	*0.10146* *0.04974*

Tabla 5.5: Error medio y varianza de estimaciones del modelo del usuario B

de ejecución de todos los trabajos se encuentra entre las 55 y las 100 horas. Algunos trabajos son lanzados los fines de semana.

Al igual que en los casos anteriores, se presentan gráficas con los resultados de estimación para los tres mejores modelos encontrados para los tamaños de histórico 5, 20 y 50 en la figura 5.12. En el detalle presentado en la figura 5.13 se hace patente de nuevo, la tendencia a una corrección brusca del modelo cuando hace una subestimación, dejando claro que son los errores más graves que cometen los modelos. Al no ajustar con tanta velocidad los errores de sobreestimación, se evita

Medida	**Valor**
Tiempo medio usado	78.314,53
Tiempo medio solicitado por el usuario	521.712,00
Tiempo medio estimado por el modelo	116.350,84
Error medio de las solicitudes	443.397,47
Error medio de las estimaciones	54.842,57
% de error de las estimaciones frente a las solicitudes	12,37
% de error de las solicitudes	566,18
% de error de las estimaciones	70,03

Tabla 5.6: Medidas de resultados para el usuario B

volver a cometer errores por defecto en fluctuaciones inmediatas, como queda claro entre los trabajos 360 y 390. También es posible adivinar que, tras este período de fuertes oscilaciones, al modelo le resulte difícil ajustarse a este comportamiento, por lo que intenta estimar una media de las duraciones, reflejado en la caída en el valor de estimación cerca del trabajo 390, lo que ayuda a hacer una estimación aceptable en los 3 o 4 trabajos sucesivos, que, tal vez por casualidad, tienden a durar ese valor medio.

En la figura 5.14 se muestra la evolución del error medio de la población de redes y de la mejor red para el mejor modelo encontrado para el usuario C, cuyo tamaño de histórico es de 50 trabajos y con 2 generaciones por evolución. A partir de la primera etapa en la que el histórico todavía no está lleno, el modelo se estabiliza con pocas fluctuaciones. Lo más destacable es el valle observable en torno al trabajo 100, donde algunos trabajos del usuario duran muy poco tiempo, haciendo al modelo muy bueno. A partir de ahí, aunque la mejor red sigue teniendo un error muy bajo, unas pequeñas subidas de error medio en la población delatan cambios de comportamiento en los usuarios, errores que se recuperan enseguida debido al uso de 2 generaciones en las actualizaciones frente a la generación utilizada para los modelos de los usuarios A y B. A partir del trabajo 200 el comportamiento del usuario se estabiliza, permitiendo observar un pequeña tendencia en el error medio de la población, que se va decrementando poco a poco ajustando el modelo cada vez más a esa estabilidad en el comportamiento.

En la tabla 5.7 se observa que, para los casos estudiados, el valor óptimo de tamaño de histórico se encuentra entre 40 y 60 trabajos. El número óptimo de generaciones varía entre 1 y 4. El mejor modelo para este usuario resulta ser el de 50 trabajos y 2 generaciones.

La tabla 5.8 presenta las medidas sobre los resultados obtenidos con el mejor modelo para este usuario. El error cometido por el modelo en las estimaciones es del 30,77 % frente al 541,51 % cometido por el usuario, es decir, el error del SOS es un 5,68 % del proveniente de las solicitudes.

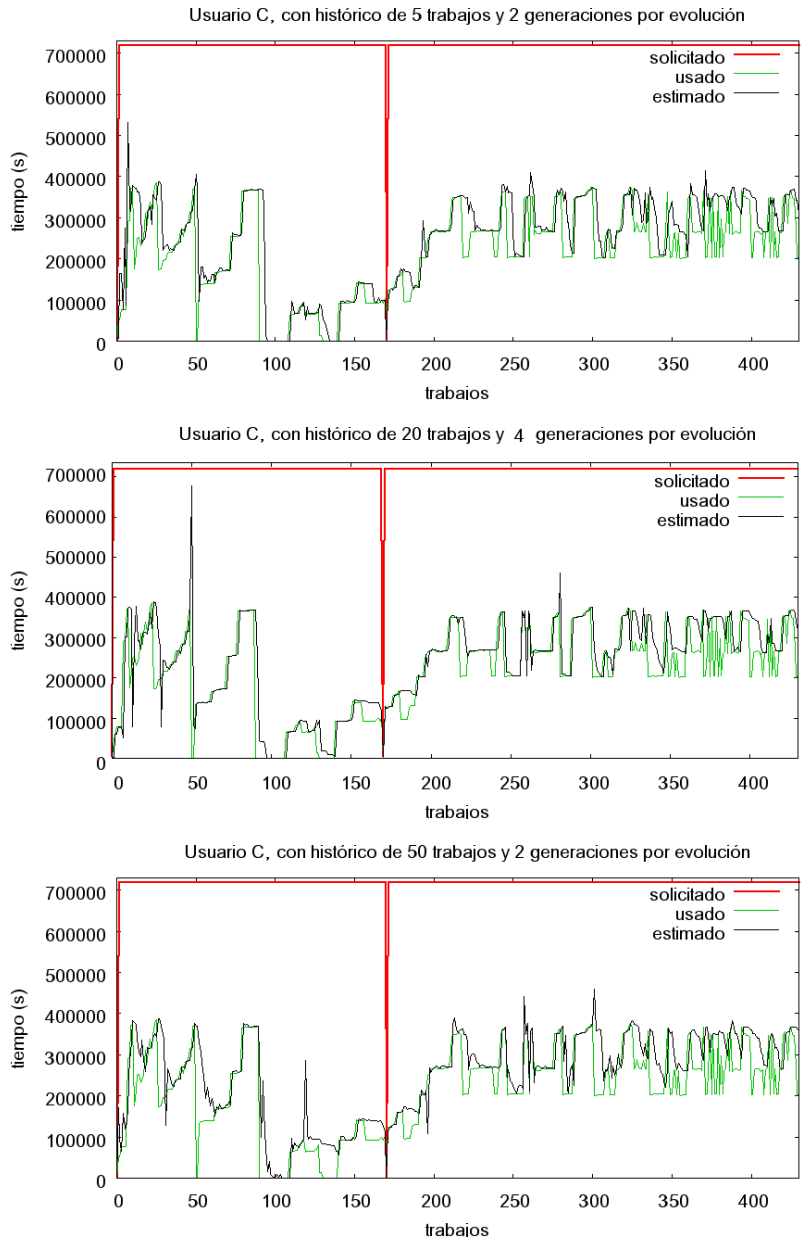

Figura 5.12: Pruebas del modelo del usuario C

Observaciones

Independientemente del tamaño de memoria utilizado, cuanto mayor es el número de generaciones, la corrección que hace la red en errores fuertes de estimación

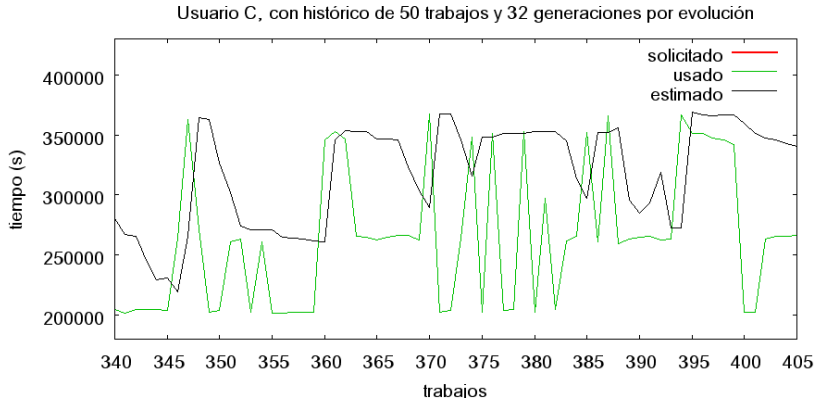

Figura 5.13: Detalle de las pruebas del modelo del usuario C

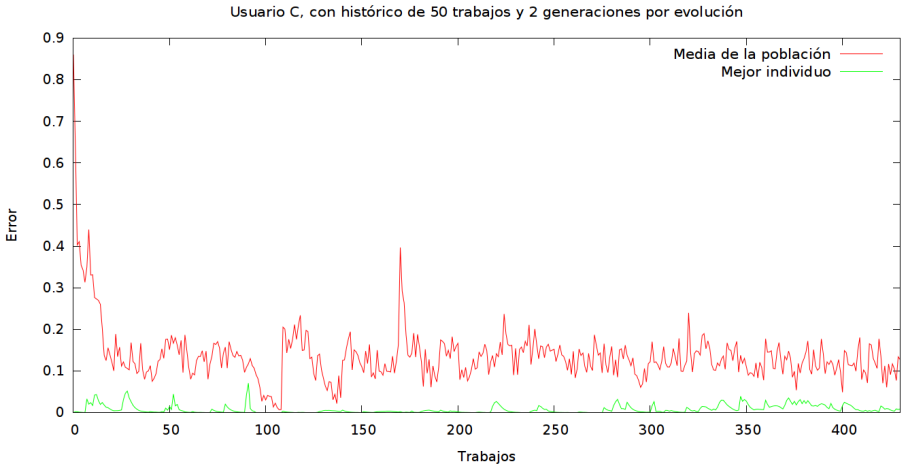

Figura 5.14: Evolución del mejor modelo para el usuario C

es muy brusca, igualando para la estimación de n el valor real en $n-1$. Esto hace que la red cometa más errores debido a la pérdida de capacidad de generalización sufrida por este sobreaprendizaje. Esto, además, aumenta la probabilidad de que los modelos caigan en máximos locales. En contraposición, con pocas generaciones, estas correcciones son más suaves, el modelo se ajusta de forma muy aceptable al histórico de trabajos y, en general, el error cometido es menor, como se puede ver en las tabla de errores. No obstante, el uso de un número bajo de generaciones aumenta la probabilidad de la presencia de picos y oscilaciones. Téngase en cuenta que esta medida de error no corresponde con una función lineal y que una pequeña variación en el valor puede significar una diferencia grande en la estimación.

Por otro lado, el tamaño de la memoria repercute de la siguiente manera: con

H/G	0	1	2	4	8	16	32
5	*0.06258*	0.05446	0.05366	0.05492	0.05404	0.05603	0.05625
	0.01907	0.02102	0.02094	0.02276	0.02173	0.02419	0.02394
10	0.06467	0.05021	0.04991	0.04969	0.05103	0.05305	0.05357
	0.02112	0.01795	0.01757	0.01779	0.01954	0.02047	0.02144
20	0.08089	0.04834	0.04774	0.04577	0.04880	0.05031	0.05406
	0.02207	0.01675	0.01557	0.01574	0.01865	0.01944	0.02263
30	0.07654	0.04566	0.04798	0.04818	0.05127	0.04890	0.05128
	0.02127	0.01416	0.01635	0.01730	0.02005	0.01905	0.02041
40	0.06778	0.04576	0.04732	0.04665	*0.04744*	*0.04804*	0.05208
	0.01828	0.01379	0.01585	0.01573	*0.01676*	*0.01739*	0.02062
50	0.06563	0.04618	*0.04339*	*0.04544*	0.04884	0.05089	0.05056
	0.01835	0.01400	*0.01311*	*0.01477*	0.01758	0.01989	0.01944
60	0.06841	*0.04499*	0.04598	0.04739	0.04850	0.04881	*0.04938*
	0.01838	*0.01340*	0.01491	0.01634	0.01758	0.01784	*0.01847*

Tabla 5.7: Error medio y varianza de estimaciones del modelo del usuario C

Medida	**Valor**
Tiempo medio usado	112.235,19
Tiempo medio solicitado por el usuario	720.000,00
Tiempo medio estimado por el modelo	138.475,38
Error medio de las solicitudes	607.764,81
Error medio de las estimaciones	34.536,99
% de error de las estimaciones frente a las solicitudes	5,68
% de error de las solicitudes	541,51
% de error de las estimaciones	30,77

Tabla 5.8: Medidas de resultados para el usuario C

poca memoria los modelos aprenden patrones locales de comportamiento pero reaccionan mal con tendencias a largo plazo, como incrementos graduales en las duraciones de los trabajos. Al aumentar el tamaño de la memoria, estos patrones pasan más desapercibidos, pero los modelos son mas conscientes de las tendencias de los usuarios.

Se puede concluir que con un comportamiento estable lo ideal es tomar un tamaño de histórico pequeño, donde pueda quedar reflejado dicho comportamiento. Además, incrementar algo el número de generaciones, permite un mejor ajuste a este comportamiento y mejores estimaciones. Al ser un comportamiento estable y no existir demasiado riesgo de cambios, este ajuste puede ser mayor. Por el contrario, si el comportamiento es inestable, debe bajarse el número generaciones para dejar las puertas abiertas a cambios. Aunque el ajuste no sea tan bueno y el aprendizaje sea más lento, al no mostrar un ajuste excesivo la adaptación al cambio se hará de forma que se comete menos error en sucesivas estimaciones. También es

recomendable en este caso un tamaño de histórico mayor pues permite almacenar el propio cambio y la estabilidad que pueda haber antes y después del mismo.

Se puede concluir que los parámetros óptimos de tamaño de histórico y número de generaciones dependen del usuario a modelar, pero que se puede llegar a una solución de compromiso tomando pocas generaciones, entre 1 y 4, y un tamaño de memoria entre 20 y 50.

5.5.2. Usuarios ficticios

Se han creado tres usuarios ficticios (*evoproc1*, *evoproc2* y *evoproc3*) que siguen patrones de comportamiento muy característicos. Se han utilizado los mismos parámetros que para los usuarios reales, los indicados en la tabla 5.2 y se ha considerado únicamente un tamaño de histórico de 30 trabajos y 4 generaciones por evolución, que si bien no han resultado óptimos en el caso de los usuarios reales si han aportado buenos resultados. El tamaño 30 en el histórico se ha mantenido comedido debido a la estabilidad en el comportamiento creada para los usuarios ficticios, motivo por el cual también ha sido incrementado el número de generaciones respecto de los usuarios reales.

Usuario *evoproc1*

Este usuario corresponde con un perfil que cumple las siguientes características:

- Siempre solicita la misma cantidad de tiempo.
- Los trabajos son lanzados por las mañanas.
- De vez en cuando envía trabajos los fines de semana.
- La duración de los trabajos se mantiene en intervalos de tiempo y hay pocas oscilaciones.
- Solamente ejecuta trabajos de un único procesador.

Como se puede ver en la gráfica de la figura 5.15 el modelo comete, generalmente, errores de estimación de pocos segundos y no realiza cambios bruscos, de forma que dificulta caer en errores de mayor peso. Los valores estimados por el modelo de este usuario son mucho mejores que los solicitados por el propio usuario.

Usuario *evoproc2*

Con el usuario *evoproc2* se ha pretendido representar a aquellos en cuyas solicitudes se guarda una relación más directa entre el tiempo solicitado y el realmente utilizado. Así, *evoproc2* cumple:

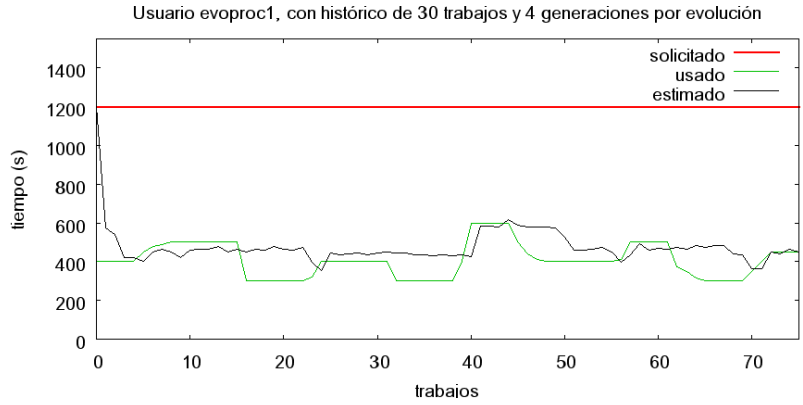

Figura 5.15: Pruebas del modelo del usuario *evoproc1*

- Sus trabajos siempre duran alrededor de un 50 % del tiempo solicitado (existe una pequeña componente aleatoria).
- Los trabajos son lanzados por las mañanas.
- Algunos de sus trabajos requieren 2 o 4 procesadores.
- No se envían trabajos los fines de semana.

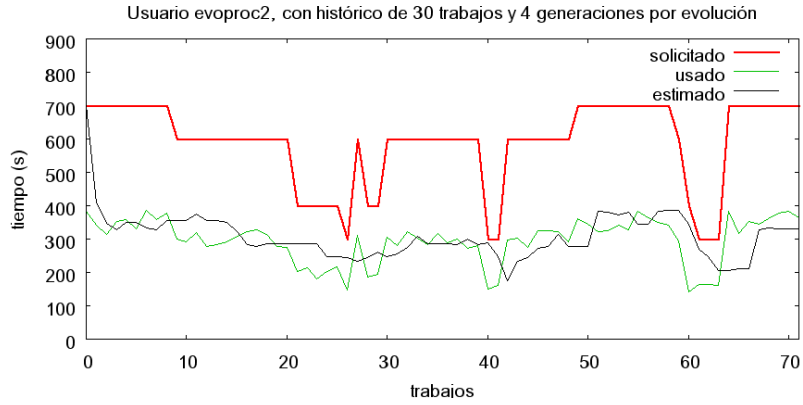

Figura 5.16: Pruebas del modelo del usuario *evoproc2*

Se ve en la figura 5.16 que el modelo tiende a hacer una media en el tiempo real de ejecución, en parte debido a esa componente aleatoria en la duración de los trabajos. Resultan, al igual que con el usuario *evoproc1*, estimaciones muy aceptables, sobre todo comparando con el tiempo solicitado.

Usuario *evoproc3*

El último de los usuarios ficticios creados también sigue un comportamiento regular:

- Siempre solicita la misma cantidad de tiempo.
- Envía trabajos durante todo el día, incluso algunas noches.
- Todos los trabajos utilizan 2 o 4 procesadores.
- La duración de los trabajos es siempre similar (y con mucha diferencia respecto de lo solicitado) para trabajos del mismo número de procesadores, necesitando alrededor del doble de tiempo para aquellos trabajos que solo utilizan 2.

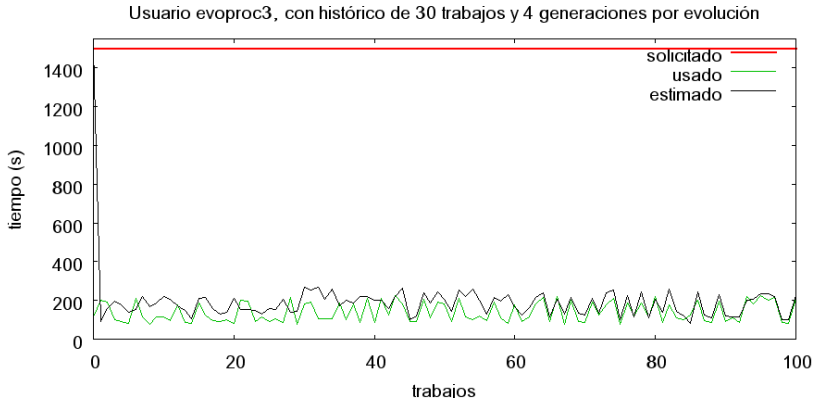

Figura 5.17: Pruebas del modelo del usuario *evoproc3*

Como se aprecia en la figura 5.17, este modelo resulta muy bueno. No solo por lo bien que se ajusta al tiempo real, sino porque hay una clara ventaja en el hecho de poder conocer el tiempo aproximado de uso respecto del solicitado cuando la diferencia es tan grande. Un planificador incurriría en errores de planificación al basarse en el tiempo solicitado. Nótese que es a partir del trabajo 40, cuando el histórico ya está lleno, cuando se empiezan a hacer las mejores estimaciones.

5.6. Resumen y conclusiones

Al principio de este capítulo se dejaba claro que la principal información que utilizan los planificadores para poder tomar sus decisiones son las solicitudes de recursos realizadas por los usuarios al enviar un trabajo a un centro de HPC. El problema fundamental estriba en la poca precisión de estas solicitudes o, incluso,

en su ausencia. Como ya se vio, algunos investigadores han trabajado para salvar esta deficiencia [13, 14, 15, 16, 84]. Las principales objeciones a estos trabajos es que se basan en análisis estadístico de datos históricos que no permiten adaptarse rápidamente a cambios en el comportamiento de los usuarios y que utilizan categorizaciones o clases de trabajos, impidiendo distinguir entre usuarios individuales, cayendo de nuevo en el problema de la generalización.

Como parte de los objetivos de este trabajo, se desarrolló un sistema de optimización de solicitudes, el SOS, cuyo fin es realizar una estimación de los recursos realmente necesarios y proporcionar al planificador valores más precisos. Para ello, el sistema permite capturar el comportamiento de los usuarios de manera continua y modelarlo por medio del algoritmo creado a tal efecto, el MCC. El sistema propuesto atiende a dos requisitos esenciales:

1. Se considera a cada usuario por separado e independiente del resto con el fin de modelar única y exclusivamente su comportamiento.

2. Los modelos se adaptan dinámicamente para ajustarse a cualquier cambio en el comportamiento de los usuarios y lo hacen en cuanto hay nueva información disponible para mantenerse actualizados en todo momento.

La aproximación aquí aportada proporciona un mecanismo fiable para la construcción de modelos de usuario hábiles para la estimación de recursos, cuyos resultados son muy satisfactorios, pues proporcionan estimaciones del uso de recursos mucho más precisas y ajustadas que las propias indicaciones de los usuarios. Según los resultados recogidos, se puede concluir que, aunque no existe una configuración concreta que produzca siempre el mejor resultado, sí hay un rango de valores para los que siempre se observa un buen resultado. En dicha configuración, parámetros como el tamaño del histórico de trabajos y el número de generaciones entre actualizaciones del modelo, difieren de un usuario a otro, pero no tanto como para no poder determinarlos de manera *on–line* y así poder escoger automáticamente los mejores para cada uno de los usuarios. Para los casos estudiados se ha encontrado que los valores de histórico que mejor funcionan se hallan entre 10 y 50 trabajos aproximadamente, mientras que el número de generaciones varía de 1 a 4, aproximadamente. No obstante, hay que considerar el efecto de estos parámetros pues, de manera general, un histórico más largo permite reflejar mejor las tendencias de los usuarios, mientras que uno más corto permite ajustarse más a patrones locales de comportamiento. De manera general, se prefieren tamaños de histórico mayores y menor número de generaciones para comportamientos cambiantes, mientras que en el caso de comportamientos estables, resulta ideal el uso de tamaños pequeños de histórico y un número de generaciones mayor.

El uso de una función de evaluación que penalice más las subestimaciones favoreciendo algo más las sobreestimaciones ligeras respecto del uso real de recursos, produce un buen resultado, pues siendo conservador para proporcionar valores al planificador que eviten la interrupción prematura de los trabajos, como normal general producen estimaciones que se mantienen levemente por encima de ese uso real, manteniendo un buen compromiso entre precisión y margen para el error.

Capítulo 6

Optimización de la satisfacción

Nada satisface más que haber llegado a tu destino tras recorrer un largo camino.

Como se indicó en la introducción a este trabajo, la fase de identificación de necesidades en el proceso general de desarrollo de un producto causa una generalización en las características deseables del propio producto. Esto lleva a que no todos los usuarios estén todo lo satisfechos que cabría esperar. Así, uno de los principales objetivos aquí establecidos es el de considerar el concepto subjetivo de la satisfacción como un criterio fundamental no solo para evaluar el rendimiento, sino para guiar el funcionamiento de un sistema informático, en este caso concreto, el planificador de un HPC, huyendo de la mencionada generalización y permitiendo que el sistema se adapte a cada uno de sus usuarios de manera particular. En este capítulo se recogen estas ideas y, además de proponer varias aproximaciones para hacer esto, se introduce el desarrollo del sistema que permita a un planificador como el ya presentado considerar la satisfacción como criterio a ser usado en sus políticas de planificación.

6.1. Preámbulo

Según la teoría del desarrollo de productos, y como afirma Ulrich en [5], existen múltiples criterios a tener en cuenta a la hora de diseñar el funcionamiento y la interacción con el usuario de un sistema informático, dispositivo electrónico o servicio virtual. De manera general, lo que las compañías buscan es maximizar las ventas de sus productos, para lo cual tienen que prestar especial atención a diversos factores con el fin de llegar a la mayor cantidad posible de público, factores que están orientados, principalmente, a satisfacer a los clientes en múltiples niveles. Como es lógico, y sin entrar en el terreno de la mercadotecnia, el diseño del producto o servicio debe ser consecuente con este principio. Algunas de las principales

características generales perseguidas, tanto por las compañías como por los propios consumidores, son las siguientes:

Cumplir su objetivo Aunque parezca obvio, es totalmente indispensable que el producto diseñado cumpla el objetivo por el cual los consumidores lo adquieren de manera completa y no parcial, como sucede de vez en cuando. En este sentido, los productos acometen su principal funcionalidad en un grado u otro y cuanto más se acerquen a las expectativas ideales, mejor producto resultará.

Facilidad de uso No solo el hecho de cumplir el objetivo hará del producto o servicio un bien destacable. Se hace necesario en la misma medida una facilidad de uso que permita al consumidor una experiencia intuitiva y agradable.

Fiabilidad Es interesante también que el proceso se complete sin errores siempre o la mayoría de las veces creando una sensación de disponibilidad absoluta, un producto con el que se pueda contar cuando se necesite.

Seguridad La seguridad es otro factor indispensable, sobre todo en aquellos servicios que manejen datos sensibles, como pueden ser las operaciones bancarias o el acceso a datos personales o privados, como el correo electrónico.

Tiempo de respuesta inmediato El tiempo de respuesta también es un factor que incide directamente en el éxito de un producto. Ningún consumidor quiere tener que esperar los resultados de sus acciones demasiado tiempo, buscando la inmediatez.

Accesorios En un segundo plano, los accesorios o funcionalidades secundarias que aportan valor añadido a los productos también son considerados. De hecho, en la actualidad, la mayoría los consumidores descartan de inmediato un producto si no realiza más funciones que la imprescindible. Este fenómeno se puede observar en el campo de la telefonía móvil.

Apariencia física Otro aspecto a tener en cuenta se centra exclusivamente en la apariencia física del producto o interfaz del servicio y, dado el caso, sus embalajes. Está claro que ante productos de características similares los consumidores preferirán aquel que les resulte más atractivo físicamente y que esté construido con los mejores materiales.

Tendencias Un factor subjetivo que no depende del producto o servicio en sí, sino del momento actual y de los movimientos sociales y de mercado, son las llamadas “modas”, que pueden determinar el éxito o el fracaso de un producto concreto.

Obsolescencia Levemente relacionado con el aspecto anterior, el consumidor espera de un producto por el que acaba de pagar un período de vigencia mínimo para evitar que quede anticuado excesivamente pronto.

Precio adecuado Uno de los principales factores es el coste del servicio o producto que, además, de alguna manera, repercute en la percepción del consumidor respecto del resto de factores.

Diferenciado El producto o servicio debe proporcionar características distintivas que lo puedan hacer atractivo frente a otros productos similares en el mercado. Adicionalmente, un producto diferenciado gana en visibilidad.

Ecológico Las cualidades ecológicas, tan presentes en la conciencia social hoy en día, de un producto refuerzan las posibilidades de éxito que este puede tener. Asimismo, un producto con un bajo consumo tendrá otros consecuencias beneficiosas a mayores, como por ejemplo, una mayor autonomía en caso de alimentación mediante baterías.

Por supuesto, estos son solamente algunos de los factores que determinan el éxito de un producto, dejando de lado otros relacionados con, por ejemplo, la política de garantías o los servicios de posventa y atención al cliente. Todas estas características deben darse a la vez para lograr un producto de éxito, aunque a un público determinado le basten solamente un subconjunto de ellas. Llevando esta afirmación al límite, cada individuo le dará mayor peso a un factor u otro obviando al resto, incluso por no habérselos planteado nunca. La idea de querer llegar al mayor público posible se trata, por tanto, de una generalización que pretende que un producto competitivo a todos los niveles mantenga satisfechos a los consumidores. En este sentido, un producto o servicio se considera de éxito cuando llega a la mayor parte posible de la población a la que está dirigido y la satisface en la mayor medida posible. No obstante, un producto que no goce de suficiente alcance también puede mantener satisfecho a su pequeño grupo de consumidores, principio básico para aumentar la clientela. Y para mantener la satisfacción general es necesario prestar atención a todos y cada uno de los factores durante el diseño y la construcción del producto. No obstante, el problema de la generalización es que hay que llegar a un equilibrio entre los factores, albergando el problema de no cumplir al cien por cien las expectativas individuales de cada cliente. Es obvio que, una vez puesto en venta un producto no se pueden modificar determinadas características del mismo hasta que se reemplace con una nueva versión, como por ejemplo, su objetivo principal o la apariencia física, mientras que otros sí pueden modificarse, a través de actualizaciones de software, por ejemplo, tales como el tiempo de respuesta, los accesorios o la fiabilidad. Esta clasificación varía en función del producto o servicio, por supuesto. Nótese que un servicio virtual accesible por internet es susceptible de ser modificado en la forma y medida que se requiera y en cualquier momento. De esta manera, si se localiza una carencia en alguno de estos factores que pueda estar minando el éxito del producto, la compañía podría trabajar e incidir directamente sobre esos factores.

En la sección 3.6 se ha hecho un breve estudio sobre la construcción y utilización de métodos para medir la satisfacción de los usuarios de sistemas informáticos. El pilar fundamental de la mayoría de ellos se basa en la determinación de factores relativos a la interacción del usuario con la aplicación concreta. Mediante un sistema de puntuación, generalmente una encuesta, se obtiene una medida estimativa de la calidad de la aplicación en función de la satisfacción global de los usuarios y que, por otro lado, permiten determinar los puntos débiles de productos y servicios. Nuevamente se está hablando de una generalización en la satisfacción. De todas formas, el conjunto de factores generales que determinan el éxito de un producto, algunos

de los cuales se acaban de presentar, junto con todos aquellos determinados por los diferentes autores citados configuran un listado de aspectos a tener en cuenta en el desarrollo de productos o servicios con el fin de maximizar la satisfacción de los usuarios o consumidores. Es evidente que es necesario un proceso de estudio para determinar cuáles son los aspectos importantes para un producto concreto. Una vez hecho esto, y para dejar de lado el problema de la generalización del éxito y la satisfacción, habría que encontrar la manera de poder obtener medidas de satisfacción por usuario o consumidor que, como parte de una retroalimentación permitiera al producto o servicio modificarse dinámicamente para adaptar su operativa a los usuarios concretos y mantenerlos lo más satisfechos posible. De esta manera, se podría incrementar el éxito general de un producto teniendo en cuenta a los usuarios de manera individual.

Resumiendo, la idea principal que de esto se desprende es que un producto o servicio puede ser mejorado proporcionándole cualidades de adaptación individual a los usuarios de forma que se mejore su percepción y se incremente, por tanto, la satisfacción, promoviendo de esta manera el éxito de dicho producto o servicio.

Volviendo al ámbito abarcado en este trabajo, hay que destacar que el proceso de planificación es susceptible de ser mejorado mediante la búsqueda de características de adaptación a los usuarios de un centro de HPC. No se ha encontrado en la literatura sistemas de planificación que tengan en cuenta medidas, objetivas o subjetivas, directamente relacionadas con la satisfacción y la percepción del servicio particulares de los usuarios finales de este tipo de sistemas. En este sentido, los planificadores pecan del mismo problema de generalización y establecen políticas (ver sección 4.1.4) destinadas a satisfacer de manera general a todo el conjunto de usuarios. Por otro lado, también hay que considerar la satisfacción del propio centro de HPC o de sus administradores, pues muchos de sus intereses dependen directamente del uso que los sistemas se haga y de su rendimiento, y, aunque es evidente que los propios administradores buscan el contento general de los usuarios, la satisfacción de aquellos viene determinada por otros factores tales como el económico, el energético o la facilidad de uso y configuración. Mientras que este último factor ha sido tratado en el capítulo 4, otros que dependen directamente del uso de recursos se podrán afrontar con la aproximación al modelado de recursos propuesto en el capítulo 7. Además, los administradores tienen la posibilidad de poder establecer directa y explícitamente, a través de las políticas de planificación y asignación de recursos, objetivos que incrementen su satisfacción. Es por eso que, sin dejar de lado al centro de HPC, la atención de este capítulo se centra en la medida y optimización de la satisfacción de los usuarios del centro.

6.2. Principales ideas

Considerar la satisfacción que produce en cada uno de los usuarios la utilización de un sistema de HPC para mejorar el servicio requiere varios pasos:

1. Definir el término “satisfacción” desde el punto de vista del uso de un centro de HPC.

2. Determinar cuáles de los factores que repercuten en el éxito del servicio son modificables.

3. Determinar cuáles de los factores modificables están relacionados directamente con la satisfacción.

4. Estudiar la manera de medir la satisfacción de los usuarios en base a los factores relevantes.

5. Estudiar la forma de utilizar las medidas de satisfacción en favor del servicio.

El primer paso es determinar clara y concisamente qué es la satisfacción de un usuario del centro de HPC y de qué depende. Bajo este contexto y en términos generales, la satisfacción se puede ver como el cumplimiento de un deseo. Simplificando, lo que un usuario quiere cuando envía un trabajo a un *cluster* es que su trabajo se ejecute en forma y tiempo adecuados, amén de que el propio hecho del envío resulte una tarea fácil de acometer. Pero la sencillez de esta afirmación oculta un entramado especialmente complejo que obliga a desgranar los factores que intervienen en el éxito del servicio. Se trata de modificar el funcionamiento del planificador teniendo en cuenta las particularidades de cada usuario. Esto hace que muchos de los factores no tengan relevancia, bien porque no son modificables dinámicamente, porque no influyen en la percepción de los usuarios o porque no tienen aplicación en este ámbito concreto. Hay que tener presente que los factores introducidos por Bailey y Pearson en [88] o Doll *et al*. en [90] están muy orientados a sistemas clásicos de información donde las principales funciones son el acceso y manejo de la propia información. En este sentido, el ámbito de la HPC y los planificadores, debido a que su función es otra, son muy limitados. De hecho, además del estado del sistema, la documentación de uso y manuales propios, los resultados de ejecución de los trabajos es la única información accesible, y aunque influyan de una forma u otra en la percepción del servicio por parte del usuario lo hará en una medida casi despreciable, pues forma parte del propio proceso de uso del sistema. Sin embargo, sí es cierto que modificando esta forma de uso del sistema y mejorando los métodos de consulta de esta información se puede mejorar la experiencia del usuario con el fin de incrementar su satisfacción. No obstante, no es el objetivo de este trabajo el estudiar la mejora del sistema en este sentido, ya que para ello existe multitud de trabajos derivados de los de Bailey y Doll hasta la actualidad. Considerando tanto los factores propuestos por estos autores, así como los esbozados en la anterior sección, entre aquellos no relevantes para la maximización de la satisfacción de los usuarios se pueden destacar los siguientes:

Cumplimiento del objetivo Es evidente que el objetivo de un servicio tan concreto no debe cambiar y mucho menos hacerlo para adaptarse al usuario.

Seguridad Es un requisito que no debe variar pues siempre ha de cumplir unos mínimos previamente establecidos. Desde este punto de vista, no tiene sentido modificar los aspectos relativos a la seguridad pues el usuario queda, o debería quedar, completamente satisfecho al garantizar la privacidad de sus datos y un cifrado suficiente en sus conexiones con el sistema.

Fiabilidad Al igual que el objetivo del servicio es siempre el mismo, su fiabilidad debe permanecer en un punto óptimo resumido en que el servicio debe funcionar según sus especificaciones. Y así será salvo fallo técnico. Tampoco tiene sentido pensar en variar este factor en función del usuario.

Facilidad de uso Es posible enviar trabajos, generalmente, a través de dos sistemas alternativos equivalentes. El más habitual es por medio del uso de comandos en un interprete de comandos en modo texto (consultar sección 5.1). Para facilitar esta tarea a usuarios poco experimentados es posible el uso de un interfaz gráfico que internamente hace uso de los comandos mencionados. Ambos sistemas están especialmente diseñados para, además de resultar flexibles, minimizar el número de pasos para el envío del trabajo y eliminar operaciones superfluas. Desde este punto de vista, la facilidad de uso se puede considerar alta, con lo que no resulta necesario una adaptación de este factor.

Idioma Aunque algunos sistemas no permiten adaptar el idioma a la nacionalidad o lenguas del usuario, la breve y escueta interacción del usuario con el sistema hace que este factor no sea demasiado relevante. No obstante, aunque el sistema sí se adaptara, no tiene interés una adaptación dinámica pues se considera el idioma como una característica no cambiante.

Precisión de la información La información que devuelve el sistema a las acciones del usuario es muy concreta, pues se trata del estado de las colas y de los propios trabajos del usuario. En este sentido, la precisión de la información no es modificable pues se espera de ella que sea máxima.

Suficiencia en la información Por el mismo motivo comentado en el factor anterior, este factor tampoco resulta relevante, pues se muestra toda la información necesaria. No obstante, cierto es que un sistema más avanzado como el que aquí se presenta puede dar nueva información al usuario, como el tiempo estimado que sus trabajos tendrán que esperar en la cola antes de pasar a ejecución, información que no proporcionan todos los sistemas.

Claridad en la información La sencillez de la información mostrada hace que la claridad sea óptima, por tanto este factor carece de relevancia.

Información actualizada El sistema presenta información recogida en tiempo real o cerca con lo que la información presentada no está obsoleta. Así, este factor no puede, en principio, ser optimizado, cosa que, además, no influiría demasiado.

Comunicación con el personal del centro Aunque sea este un factor que puede afectar en gran medida a la satisfacción de los usuarios, al ser algo que se mantiene por una vía ajena al sistema en sí, no influye en la propia interacción con el usuario.

Tendencias Un cambio de tendencia en el ámbito de la supercomputación pudiera llevar al uso de nuevos paradigmas. No obstante, esto implicaría una inversión gradual y a largo plazo, entre otros motivos, para evitar la obsolescencia

prematura de los actuales métodos. Por tanto, aún con la posible aparición de estos supuestos cambios, el uso del actual sistema no se vería en exceso perjudicado.

Servicio diferenciado Los usuarios utilizan el servicio que se ajusta a sus necesidades. En este sentido, carece de interés el buscar características que lo distingan del resto siempre y cuando se cumpla con los requisitos establecidos.

Apariencia física Este término hace referencia aquí a las cualidades estéticas del interfaz gráfico de usuario (*Graphical User Interface*, GUI). Es evidente que las modificaciones que pudieran hacerse en el GUI no afectarían en exceso a la percepción del servicio, pues no deja de ser una simple vía para comunicarse con él. Esto se ve enfatizado por el hecho de que un simple GUI basado en comandos parece ser una solución estandarizada para este tipo de sistemas. A su vez, la sencillez de uso y el mínimo conjunto de instrucciones utilizables no generan la necesidad de un rediseño o mejora del interfaz.

Formato de salida adecuado Debido, nuevamente, a la breve y concisa información que otorga el sistema al usuario, el formato de salida no es una característica relevante, ya que con un mínimo de diseño se alcanza un nivel adecuado. Por otro lado, es algo a los que los usuarios se acostumbran rápidamente siendo, incluso, contraproducente el variarlo.

Accesorios Es evidente que un servicio orientado a un fin tan específico cualquier tipo de accesorio ajeno a dicho fin sea totalmente prescindible, por tanto, este factor carece de interés alguno.

Obsolescencia Aunque las tendencias en computación cambien a lo largo del tiempo, ya se ha dicho que es un proceso lento. Esto hace que un sistema concreto pueda quedar obsoleto con el paso del tiempo en favor de otros sistemas de vanguardia, no obstante, es muy probable, como así sucede en la actualidad, que sistemas más antiguos y de menores prestaciones sigan siendo de gran utilidad, pues aún bajo estas circunstancias siguen resultando sistemas de gran rendimiento.

A la vista está que muchos de los factores no son relevantes o carecen de sentido en el ámbito abarcado en este trabajo. No obstante, sí existe un factor fundamental que incide muy directamente en la satisfacción, si no la determina casi por completo. Se trata del tiempo, pero no del tiempo de respuesta del sistema, que puede ser más o menos inmediato y despreciable sino al lapso entre los momentos en que un usuario envía un trabajo al sistema y en el que el trabajo comienza su ejecución, lo que se ha llamado tiempo de espera del trabajo. Es evidente que, por ejemplo, si un usuario envía un trabajo y este empieza a ejecutarse de manera instantánea, el usuario estará plenamente satisfecho con el servicio recibido. Así, simplificando y con matices, a medida que el tiempo de espera se hace mayor, la satisfacción se va decrementando. Y esta es la idea principal y básica manejada aquí para incorporar criterios de satisfacción a un planificador de un centro de HPC. Además, la insatisfacción causada por el tiempo de espera puede ser distinta para distintos usuarios, hecho que puede ser explotado durante la planificación. Pero asociados al tiempo

existen otros factores a considerar, como son las expectativas del usuario o la sensación de control. Las expectativas que tenga el usuario modulan la idea básica de que con el paso del tiempo disminuye la satisfacción. La sensación de control sobre el sistema disminuye cuando el tiempo de espera aumenta, lo que crea en el usuario la sensación de que no es él, sino el sistema, el que pone sus trabajos en ejecución, cosa que es cierta, pero cuando el tiempo de espera es despreciable, el usuario no lo percibe así, sino que la ejecución del trabajo es vista como una consecuencia directa del acto del envío y no del estado ni de las decisiones del sistema.

En resumidas cuentas se puede decir que es el tiempo el que determina en primera instancia el modo en el que el usuario percibe la calidad del servicio y por ende, su satisfacción.

6.3. Midiendo y estimando la satisfacción de los usuarios

Una vez establecido el principal factor determinante de la calidad el siguiente paso es encontrar la manera de medir la satisfacción, que podría pasar por presuposición o medidas reales, teniendo en cuenta su inherente subjetividad. El motivo fundamental que sustenta la necesidad de medir la satisfacción es el de poder evaluar cuantitativamente una planificación de trabajos en términos de la percepción del usuario. Como se dijo, esto permitirá al planificador aplicar políticas enfocadas a la satisfacción individual y general de los usuarios de un centro de HPC. Para esto es imprescindible un sistema de predicción o estimación de la satisfacción, que ha de diseñarse y desarrollarse considerando en primer término el sistema de medición de la satisfacción, dado que, al ser las planificaciones, lógicamente, en tiempo futuro, no se puede medir la satisfacción, sino estimarla.

No es la intención de este trabajo encontrar la manera perfecta de medir la satisfacción con la máxima precisión y transparencia al usuario posibles, sino sugerir algunas formas de hacerlo aliviando algunos de los problemas que implica el considerar la satisfacción de los usuarios de sistemas informáticos y, a la vez, poder contar con un mecanismo que tenga en cuenta la satisfacción proporcionando nuevos factores a considerar en las políticas de satisfacción para que el sistema pueda adaptar de manera dinámica y sin supervisión su comportamiento para mantener el contento general. Es por esto que cualquier sistema de medida y estimación de satisfacción que cumpla en mayor o menor medida estas condiciones sería válido de cara a un sistema global de planificación de procesos en un centro de HPC. Así, podrían usarse los modelos aquí presentados o podría valer una combinación de ellos. La propuesta principal que aquí se presenta se materializa en un sistema que, tras medir o presuponer la satisfacción que ha provocado la ejecución de cada uno de los trabajos de un usuario en un sistema de HPC, “aprendiera” la cantidad de satisfacción aportada al usuario por cada uno de esos trabajos. Una vez asimilada esta información, el sistema podrá utilizarse como mecanismo de estimación de satisfacción para los trabajos de un usuario en una determinada planificación. Por supuesto, esto es posible teniendo en cuenta como entradas a todo este proceso

las estimaciones de tiempos de comienzo y de fin de cada trabajo que proporciona el planificador evolutivo que hace, a su vez, uso del sistema de optimización de solicitudes, ambos presentados en este trabajo.

Antes de buscar la manera de medirla la satisfacción es imprescindible caracterizar la forma de cuantificarla, considerándose tres aproximaciones principales:

1. Establecer una escala lineal de satisfacción entre dos extremos, un mínimo y un máximo a partir de los cuales no es definible la satisfacción y en donde un nivel de satisfacción de $2n$ implica un valor de satisfacción que duplica a otro nivel de n. Por ejemplo, esta escala podría definirse entre los valores 0 y 1, significando 0 un satisfacción nula y 1 una satisfacción plena.

2. Directamente relacionado con el anterior pero haciendo un cambio al ámbito de la psicología sería posible establecer una escala de satisfacción representable mediante etiquetas que determinan de manera ordenada un nivel determinado de satisfacción. Según Osgood en su estudio de la naturaleza y medida del significado [127], bastarían siete etiquetas semánticas para lograr una buena representación del concepto de satisfacción que podrían ser: totalmente satisfecho, muy satisfecho, bastante satisfecho, satisfecho, algo insatisfecho, muy insatisfecho, totalmente insatisfecho. No obstante, una escala puede expandirse (o contraerse) con la incorporación de nuevas etiquetas si surgiese la necesidad.

3. El principal problema de las anteriores aproximaciones es precisamente el hecho de estar acotadas que implica que no se pueda representar de manera continua la satisfacción en una escala infinita. Esto se logra eliminando una de las cotas (o ambas) permitiendo que la satisfacción crezca o decrezca indefinidamente. De esta manera, y por ejemplo, no se alcanzaría un estado de satisfacción total, sino que la satisfacción se incrementaría de manera continua resultando más alta cada vez.

Las diferentes alternativas presentadas para la representación y caracterización de la satisfacción permiten analizar y desarrollar los sistemas de medida y predicción introducidos. Por supuesto, no es menester limitarse a una de estas opciones pudiendo utilizarse un híbrido entre ellas así como aplicar diversas técnicas de computación y representación del conocimiento como los sistemas difusos.

Se considera el sistema de estimación de la satisfacción, llamado a partir de ahora SES (Sistema de Estimación de la Satisfacción), y subsistema de EvoProc, como un mecanismo para estimar la satisfacción por trabajo y no por usuario, es decir, la satisfacción del usuario vendrá determinada por una suma o una media de la satisfacción aportada por cada uno de los trabajos enviados por dicho usuario. Nótese que, no obstante, ambas medidas pueden ser utilizadas durante el proceso de planificación. De esta manera, el SES puede ser visto como un módulo o caja negra que permita al proceso de planificación obtener medidas de satisfacción por medio de la provisión de datos referentes al trabajo y a sus tiempos de espera.

A continuación se recogen las diferentes alternativas desarrolladas para un sistema de medida y estimación de la satisfacción de diversa complejidad. La determina-

ción de varias estrategias ha sido motivada por la dificultad inherente a la medición de un parámetro subjetivo y del requisito de construir un sistema tan transparente como sea posible buscando siempre minimizar la intrusión al usuario. Para cada una de estas alternativas, de las cuales se presentan resultados en la sección 6.6, se ha tenido en cuenta una idea común: la satisfacción se decrementa según el tiempo de espera aumenta, y esto se considera así para cualquier usuario. Se puede dar la excepción, en casos concretos, de que el nivel de satisfacción se mantenga durante un periodo concreto de tiempo, pero, lógicamente, jamás se incrementará. No obstante, es necesario matizar esta afirmación, ya que, como se vio, el tiempo no es el único factor determinante de la satisfacción. En el caso de un cambio de expectativas, por ejemplo, la estimación de la satisfacción puede sufrir un incremento, pues dicho cambio puede alterar la percepción del usuario.

Otra idea común aplicada es la de presuponer una satisfacción máxima bajo el supuesto de que un trabajo enviado por un usuario pasa a ejecución de manera instantánea sin permanecer en una cola de espera un tiempo perceptible.

6.3.1. Perfil único de decremento continuo de la satisfacción

Esta manera de estimar la satisfacción se puede considerar una de las más directas. Tiene la ventaja de no necesitar la intervención del usuario, sin embargo, no está exenta de problemas, pues se trata de presuponer la satisfacción del usuario y cómo esta se decrementa a medida que sus trabajos permanecen en espera. Se considera, además, que todos los usuarios del sistema poseen el mismo perfil de satisfacción. Teniendo en cuenta que la satisfacción se mide por cada uno de los trabajos enviados por el usuario, hay que reparar en la idea de que no es lo mismo esperar una cantidad de tiempo determinada por dos trabajos de distintas características, en concreto al hablar de su tiempo de ejecución. Como ejemplo, si un trabajo necesita cien horas de CPU para su ejecución es factible esperar diez horas en la cola, pero la misma espera para un trabajo de diez minutos sería prácticamente inadmisible. Esto lleva a diseñar una función continua que toma como parámetros no solo la espera estimada a la que tendrá que verse expuesto un trabajo, sino también diversas características que modulen el decremento de satisfacción, tales como la cantidad de tiempo de CPU necesitada o la cantidad de RAM.

Si se tiene en cuenta una escala de satisfacción continua entre un máximo dado (satisfacción plena) y un mínimo en el infinito, lo que en realidad se está considerando es una escala de insatisfacción que podría estar definida entre 0 y $-\infty$. De esta manera, el sistema mide la insatisfacción causada por la espera para cada uno de los trabajos. Sirva de ejemplo para este sistema de medida la función expuesta a continuación. Se trata de una función continua que decrementa la satisfacción exponencialmente con el aumento del tiempo de espera. De esta manera se considera que tiempos de espera aceptables producen un decremento lógico y natural de satisfacción, pero que tiempos de espera mayores producen una disminución mucho más acusada. De esta manera, se puede denominar tolerancia a la espera al tiempo máximo de espera admisible, que es definido por los administradores del sistema. Las ideas de que la tolerancia es mayor cuanto mayor sea el tiempo de ejecución

del trabajo y de que no es proporcional a este se adoptan mediante el uso de una función de tolerancia que tiene el aspecto de la figura 6.1, resuelta mediante la función de una elipse, aunque podría considerarse cualquier otra curva:

$$T(t_r) = D\sqrt{1 - \frac{(t_r - M)^2}{M^2}} \tag{6.1}$$

$T(t_r)$ es el valor de tolerancia para un trabajo de duración t_r. M es el tiempo de ejecución máximo permitido por el sistema y D es el tiempo de espera máximo admisible para un trabajo de duración M.

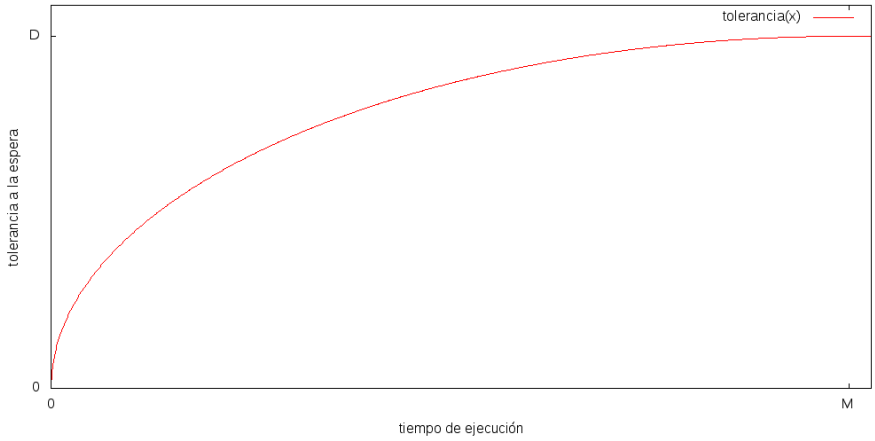

Figura 6.1: Tolerancia de un trabajo a la espera

Las curvas de satisfacción presentadas en la figura 6.2 quedan determinadas por la siguiente función matemática basada en una parábola:

$$S_c(j) = -\frac{t_w^2}{T(t_r)^2} \tag{6.2}$$

donde $S_c(j)$ representa el nivel de satisfacción para un trabajo de duración t_r que sufre una espera de t_w. $T(t_r)$ es el valor de tolerancia a la espera para un trabajo de duración t_r que viene dado por la función T de tolerancia recién presentada en la ecuación 6.1. El valor máximo de satisfacción es de 0 y puede decrementarse indefinidamente.

En ellas se aprecia como la satisfacción disminuye de diferente manera dependiendo de la duración estimada del trabajo (T y $T/2$ en la gráfica). La función ha sido diseñada para tener en cuenta la cantidad de CPU solicitada, pero puede ser modificada para considerar otras variables tales como la cantidad de CPU estimada y las cantidades de RAM solicitadas y estimadas por el SOS.

A favor de esta aproximación se encuentran características como la sencillez del cálculo y por tanto la velocidad de cómputo, factor interesante desde el punto de

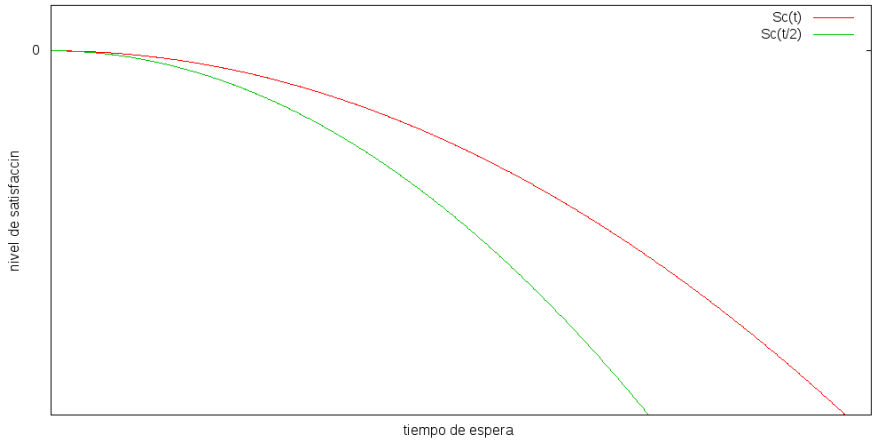

Figura 6.2: Decremento continuo de la satisfacción

vista de la eficiencia del proceso de planificación. La característica más importante es la ya comentada de transparencia al usuario. No obstante, los problemas que acarrea esta solución son varios:

- Todos los usuarios tienen el mismo perfil. Esto modela de manera muy simplificada y, en cierta forma, trivializa la percepción que los usuarios tienen del servicio. El hacer esto descarta de manera categórica cualquier otro parámetro con influencia sobre la satisfacción o, visto de otra manera, generaliza hasta las últimas consecuencias el concepto de satisfacción de los usuarios.
- La forma de medir la satisfacción no es dinámica, con lo que para reflejar un cambio en dicha forma es necesario modificar de manera explícita la función.
- La función ha de ser diseñada asumiendo que la satisfacción guarda una relación directa con esta, que en realidad no es más que una función más o menos compleja para representar la cantidad de espera que ha sufrido un trabajo.
- Se asume que la satisfacción se decrementa de manera continuada cuando en realidad esto no tiene por qué ser así.

6.3.2. Perfil único de decremento discreto de la satisfacción

Para abordar el último problema comentado de la aproximación anterior se propone una leve modificación. Se basa en la idea de que el usuario no es consciente de manera continuada de que sus trabajos están esperando para pasar a ejecución y que, solo de manera puntual, se consulta el estado de las colas para conocer el de sus trabajos. Por ejemplo, un usuario podría hacer una consulta al sistema cada día por la mañana, o cada semana, y de esta manera, su satisfacción se decrementaría

en cierta cantidad en esos momentos determinados. Dicho de otra forma, no importa demasiado que un trabajo entre una hora antes o después si alguien consulta el sistema con una frecuencia mucho menor, por ejemplo cada día.

Esta modificación se basa en la función 6.2 pero con una leve modificación. La idea es definir un valor de intervalo durante el cual la satisfacción se mantiene constante. Una vez superado ese intervalo, el valor de satisfacción cae hasta el indicado por la función, momento en el cual comienza un nuevo intervalo. La longitud del intervalo puede ser única o variar con el tiempo o cualquier otro parámetro. Además, los extremos de los intervalos pueden venir dados por el día o la semana naturales, por ejemplo, o depender directamente del momento de envío de los trabajos, lo que haría que cada 24 horas, por ejemplo, a partir del momento de envío del trabajo, la satisfacción decaiga en una determinada cantidad. A modo ilustrativo se presenta en la figura 6.3 una curva de satisfacción con decremento discreto junto a otra de decremento continuo para su comparación, ambas para un trabajo de la misma duración. La curva se construye basándose en la ecuación 6.2 tomando para cada valor de espera el valor de satisfacción del extremo del último intervalo, es decir, de la última vez que se consultó el sistema, por ejemplo. De esta manera, y suponiendo una función $L(u)$ que para el usuario u devuelve la diferencia entre el momento actual y el de la última consulta, se puede expresar una nueva ecuación para este perfil de satisfacción de la siguiente manera:

$$S_d(j) = -\frac{(t_w - L(u))^2}{T(t_r)^2} \tag{6.3}$$

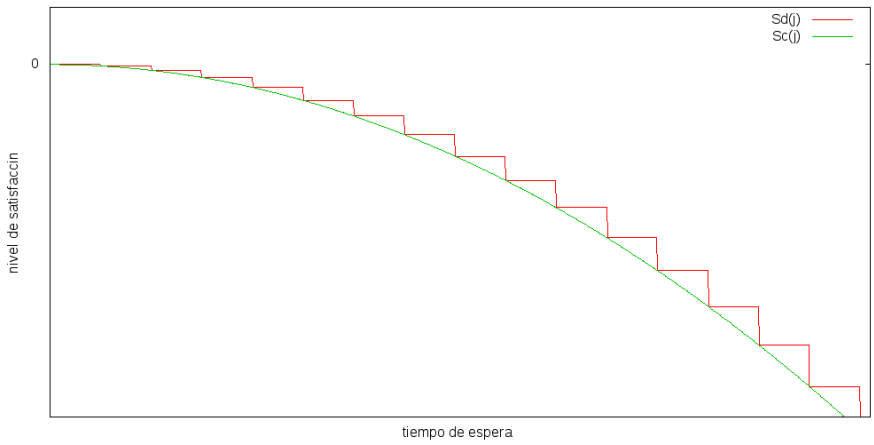

Figura 6.3: Decremento discreto de la satisfacción

Otra variante consiste en considerar un decremento de satisfacción, y por tanto un intervalo, única y exclusivamente en el momento en que el usuario hace una consulta al sistema acerca del estado de sus trabajos, asumiendo que su satisfacción decrece si sus trabajos no han entrado en ejecución y siguen esperando. Sin embargo, esto implica el desarrollo de un mecanismo que permita predecir los momentos

en los que los usuarios realizan dichas consultas al sistema para poder calcular la pérdida de satisfacción para cada uno de sus trabajos.

A excepción del problema resuelto ya comentado, esta aproximación sigue adoleciendo de los problemas de la anterior. El uso de perfiles, mientras no sean individuales, mantiene el inconveniente de las categorizaciones en el conjunto de usuarios. No obstante, unos perfiles así diseñados resultan útiles para la realización de pruebas y ajustes del planificador o para comparaciones con otros métodos de medida. Además, vienen a formalizar, en cierta forma, las asunciones realizadas por los administradores de los sistemas en cuanto al decremento de la satisfacción en el tiempo. Con esto, de querer ser incorporados a las políticas de planificación, podrá hacerse desde la óptica del planificador evolutivo de manera más natural e intuitiva. Una última ventaja sería el uso de perfiles como modelo inicial de satisfacción cuando todavía no hay datos en el sistema para construir un modelo personalizado.

6.3.3. Diversos perfiles de decremento de satisfacción

Llegados a este punto resulta bastante natural la resolución de otro de los problemas planteados. Es posible asignar un perfil de satisfacción diferente de entre los dos propuestos y sus posibles variantes a cada uno de los usuarios, permitiendo además una configuración y ajuste de parámetros individualizada. No obstante, surge otro inconveniente y es el hecho de tener que asignar y ajustar un perfil diferente por usuario y todo lo que ello conlleva. Esto implicaría hacer un estudio de cada uno de los diferentes usuarios para poder extraer un perfil de satisfacción. Aún así, y suponiendo una solución que permitiera realizar dicha asignación de manera automática, sería necesario otro componente que posibilitara su reajuste dinámicamente para adaptarse a los cambios del usuario. Precisamente para esto se propone el mecanismo presentado a continuación, que permitirá caracterizar al usuario y modelar su satisfacción de manera individual.

6.3.4. Encuestas

Es posible lidiar con el problema de la necesidad de suponer los niveles de satisfacción en los usuarios a costa de sacrificar la deseable característica de la no intrusión. Como se vio en la sección 3.6, algunos de los autores, en los trabajos con más relevancia, realizan encuestas con el fin de conocer el nivel de satisfacción en los usuarios de diversos sistemas. El principal problema de estos trabajos es la longitud y complejidad de las encuestas, que requieren un esfuerzo del usuario que no siempre es posible, máxime si se tiene en cuenta un sistema altamente dinámico que varíe en gran medida con el tiempo. Esta característica puede hacer que los resultados de las encuestas queden invalidados. Bajo estas circunstancias, es necesario repetir la encuesta para poder extraer, nuevamente, conclusiones acerca del estado del sistema a través de la satisfacción de los usuarios. Sin embargo, en el actual trabajo, lo que se pretende es incrementar la satisfacción global de los usuarios por medio de la alteración del comportamiento del sistema, en concreto, del proceso de planificación. No se necesita para dicha empresa un análisis pormenorizado

de los diferentes factores del sistema que inciden en la satisfacción pues, como se ha visto, el principal factor es el tiempo. Esto permite la construcción de encuestas concretas, breves y muy dirigidas. La idea principal es no perturbar demasiado al usuario. Resulta este un tema de vital importancia ya que es necesario incluir un paso en la interacción normal del usuario con el sistema. Lógicamente, y al igual que puede resultar una molestia para usuarios antiguos, los usuarios que se registren en el sistema asumirán este paso como parte normal de la interacción. Este nuevo paso no es otra cosa que la respuesta a la encuesta, que debe ser diseñada para que sea lo más inocua posible. Por ejemplo, la encuesta podría constar de una única pregunta en la que se solicita al usuario que puntúe de cero a diez, o mediante etiquetas semánticas, como se comentó previamente, el nivel de satisfacción para sus últimos trabajos puestos en ejecución desde la última sesión en el sistema y tras presentarle los resultados. Es evidente que, si ningún trabajo hubiera entrado en ejecución desde la última sesión no sería necesario molestar al usuario con ningún tipo de encuesta. Estas encuestas así construidas permiten conocer el nivel de satisfacción que ha producido un trabajo o un conjunto de ellos en cuanto el usuario se conecta al sistema.

La idea de utilizar encuestas breves y sistemas de evaluación similares se ha utilizado con éxito en sitios *web* como pueden ser Facebook o Amazon. En el primer caso, un sencillo mecanismo de evaluación de las distintas publicaciones a los que los usuarios tienen acceso permite a los desarrolladores de Facebook y a terceros conocer si el contenido publicado es del agrado de los usuarios. El mecanismo se basa en que se permite a los usuarios marcar cada una de las publicaciones, propias o ajenas, con una señal que indica un gusto especial por la publicación. El mecanismo es tan sencillo que está aceptado de manera global por la comunidad de usuarios, que lo consideran una funcionalidad básica del propio sitio. Dichas señales son las llamadas "me gusta" o "*like*". A partir del momento en que un usuario marca una publicación con un "me gusta", el resto de usuarios son conscientes de ello y además, la propia plataforma, así como otro público ajeno (si así lo permite la configuración de privacidad), disponen de valiosa información. Por ejemplo, en [128] Kosinski *et al.* utilizan dicha información para conocer distintas características de los usuarios. En concreto son capaces de discernir entre la orientación sexual de los hombres, la procedencia caucásica o africana de los norteamericanos o su inclinación política hacia republicanos o demócratas con tasas de acierto 88 %, 95 % y 85 %, respectivamente. Un trabajo que se asemeja un poco más al tratado en esta tesis es el propuesto por Timian *et al.* en [129], en el que se intenta medir a través de Facebook y su mecanismo de "me gusta" la calidad del servicio proporcionado a los pacientes de diversos hospitales para compararla con otras medidas clásicas de evaluación de calidad. Por medio del uso de Facebook para comunicarse con los pacientes se obtienen una serie de "me gusta" que sirven como aproximación a la satisfacción de los pacientes. En el caso de Amazon, aquellos clientes que hayan adquirido un producto tienen la posibilidad de asignarle una puntuación entre uno y cinco. La propia compañía utiliza las puntuaciones de los productos en sus algoritmos de recomendación de productos, como ellos mismos explican en [130].

Osgood sentó las bases en [127] de las escalas de diferencial semántico. Estas escalas son un instrumento de evaluación psicológica que consiste en presentar

diversos conceptos al usuario que este debe evaluar relacionándolo con una lista de adjetivos. Los adjetivos se presentan de manera bipolar en forma de un par de elementos opuestos. El sujeto debe marcar en una escala graduada entre ambos elementos opuestos qué le sugiere el concepto presentado, o dicho de otra forma, cómo ubica el concepto en relación a ambos polos, que representará la dirección y la intensidad de su juicio. Osgood determina y utiliza, además, escalas de siete pasos o puntos. Por ejemplo, al sujeto se le presentaría el concepto "pacifista" y una escala de siete pasos entre los adjetivos opuestos "amable" y "cruel" para que pudiera emitir su juicio. Bradley *et al.* [131] utilizan este sistema para la medida de emociones tales como el placer, la excitación y el dominio ante una variedad de estímulos en forma de imágenes demostrando que es posible utilizarlo para determinar la emoción causada en un sujeto ante un determinado estímulo. De esta manera, se mide la satisfacción de los usuarios ante el propio estímulo proveniente de las sensaciones que el sistema le cause al presentarle los resultados de sus trabajos, y los adjetivos "satisfacción" e "insatisfacción" para poder emitir su evaluación.

Utilizando estas ideas para medir la satisfacción, un ejemplo de escenario permitirá aclarar cómo es la nueva interacción del usuario con el sistema, qué nueva información se presenta al usuario y cómo se llevan a cabo las encuestas.

Primera sesión

- El usuario se conecta al sistema de HPC.
- El sistema le da paso y puede comenzar la interacción normal.
- El usuario envía un trabajo *A* al sistema y es colocado en una cola de espera.
- El usuario se desconecta del sistema.

Segunda sesión

- El usuario se conecta al sistema un día después de su última sesión.
- El sistema le da paso y puede comenzar la interacción normal.
- El usuario consulta el estado de las colas y de sus trabajos y comprueba que el trabajo *A* lanzado el día anterior sigue en espera.
- El usuario envía un trabajo *B* y es colocado en una cola de espera.
- El usuario se desconecta del sistema.

Tercera sesión

- El usuario se conecta al sistema dos días después de su última sesión.
- Como el trabajo *A* ha entrado en ejecución desde la última sesión, el sistema presenta esta información (o similar) al usuario:

```
Desde su última sesión ha pasado a ejecución uno de sus trabajos.
A continuación se presenta la información relativa:
 - Trabajo: A.
 - Solicitudes de recursos:
    * Número de núcleos de ejecución:      16
    * Tiempo de CPU:                       60:00:00 (hh:mm:ss)
```

```
    * Cantidad de RAM:                    2 Gb
    * Cantidad de disco:                  128 Mb
 - Fecha y hora de envío del trabajo:     18:14:33, 15/05/2012
 - Fecha y hora de puesta en ejecución:   03:54:35, 17/05/2012
 - Sesiones iniciadas en el intervalo:    1
 - Tiempo de espera del trabajo:          33:40:02 (hh:mm:ss)
```

- Posteriormente, y antes de dar paso al usuario, el sistema realiza una encuesta acerca de la satisfacción al usuario.

```
En vista de los resultados mostrados, indique su nivel de
satisfacción con el tiempo de espera que ha tenido que sufrir
para el trabajo A:
  1.- Plenamente satisfecho.
  2.- Muy satisfecho.
  3.- Bastante satisfecho.
  4.- Satisfecho.
  5.- Algo insatisfecho.
  6.- Muy insatisfecho.
  7.- Totalmente insatisfecho.
Por favor, introduzca a continuación su nivel de satisfacción: _
```

- El usuario introduce su respuesta y el sistema, tras agradecer su tiempo y su respuesta, da paso al usuario.
- El usuario continúa con la interacción normal.

Lo que se logra con encuestas es conocer de primera mano el nivel de satisfacción que el sistema causa en el usuario. Mediante la recopilación de información de este tipo se logran pequeñas bases de datos actualizadas por usuario. Por medio del procesado de estas bases de datos, el sistema puede "aprender" la manera en que los usuarios perciben la calidad del sistema para poder así estimar la satisfacción que causará una determinada espera para un determinado trabajo. Así, y de la misma manera que se hace con el SOS, explicado en la sección 5.3.1, sería posible utilizar una red de neuronas artificiales como mecanismo para la generalización y por tanto de estimación de satisfacción y construir con ella un modelo de satisfacción. Una red de neuronas construida a este efecto tendría las entradas necesarias relacionadas con la información de la solicitud de recursos del trabajo y del tiempo de espera soportado por el trabajo. La salida de la red, que podría ser continua o discreta, indicaría el nivel de satisfacción producido por la espera para ese trabajo, es decir, por el servicio ofrecido por el sistema para la solicitud concreta. Con cada nuevo dato acerca de la satisfacción extraído de nuevas encuestas se podría actualizar una red de neuronas por usuario de manera dinámica. De esta forma, el mecanismo de estimación de satisfacción se mantendría actualizado y se iría adaptando dinámicamente según los cambios que ocurriesen en el sistema o en los propios usuarios. Es decir, un esquema muy similar al planteado en el algoritmo de MCC. No obstante, en este trabajo se han utilizado otro tipo de técnicas, como la definición de una superficie de satisfacción determinada por polinomios. Estos polinomios se aproximan mediante una regresión en la que los coeficientes son ajustados mediante algoritmos evolutivos, como se explica en la sección 6.6.1.

Existe un inconveniente añadido a esta aproximación. Si el usuario se percatara de que el sistema intenta ajustarse para satisfacerle lo más posible podría darse el caso de que el propio usuario falseara sus respuestas para mostrarse al sistema

menos satisfecho de lo que realmente está con el fin de lograr un servicio todavía mejor. No obstante, dada la propia naturaleza de la solución, podría utilizarse el propio modelo de satisfacción para detectar un desvío en la conducta del usuario a fin de descubrir dicha trampa, aunque no se pueda asumir que realmente así suceda. La idea sería contrastar la respuesta del usuario con la propia estimación del modelo de satisfacción. De existir un gran desvío en la respuesta se puede entender que, de alguna manera, la respuesta proporcionada por el usuario es errónea. Este hecho podría tenerse en cuenta para actualizar el modelo de satisfacción.

6.4. La satisfacción del HPC

Como se ya se indicó en la introducción a este trabajo y en la de este capítulo, al tratar el concepto de satisfacción no hay que dejar de lado al centro de HPC, es decir, a los administradores del centro. El objetivo de este trabajo es el de introducir criterios subjetivos, en forma de satisfacción, en las políticas de planificación. Pero, para ello, no solo hay que considerar la satisfacción de los propios usuarios pues, aunque uno de los objetivos de un centro de HPC es el de mantener un alto nivel de satisfacción en sus usuarios, existen otros intereses que influyen directamente en la satisfacción del centro. Se podría decir que esta última es una suma o combinación de la satisfacción de los usuarios y el grado en que se alcanza cada uno de estos intereses, que suelen ser económicos, energéticos e, incluso, sociales y políticos.

La principal diferencia entre los administradores y los usuarios, en lo que a satisfacción se refiere, estriba en que los administradores saben de manera exacta cómo debe comportarse el sistema para lograr un alto grado de satisfacción desde su punto de vista, pero desconocen qué es lo que satisface a cada uno de los usuarios, con el agravante de ser un gran número, y cómo trasladar esto a las políticas de planificación. Es por esto que el principal esfuerzo de este capítulo se centra en la medida y uso de la satisfacción de los usuarios, pues los criterios que satisfacen a los administradores pueden ser incluidos de manera explícita en las políticas de planificación del sistema. Además, desde el punto de vista del factor de facilidad de uso, comentado previamente y considerado uno de los factores principales determinantes de la satisfacción en trabajos como [88], [90] o [94], la satisfacción de los administradores se ve incrementada al haber reducido la complejidad de la configuración de los sistemas y la definición de políticas por medio del planificador evolutivo propuesto y planteado en el capítulo 4, que sirve como base al resto de soluciones propuestas en este trabajo. Adicionalmente, la propuesta de un sistema de modelado de recursos que se hace más adelante, en el capítulo 7 y la posibilidad de poder estimar parámetros como el coste económico de uso o de energía utilizado, entre otras cosas, facilita todavía más la configuración de las políticas de cara a cumplir con aquellos requisitos que determinan la satisfacción del centro de HPC.

6.5. Planificando para maximizar la satisfacción

En la sección 6.3 se ha profundizado en la forma de implementar y utilizar nuevas políticas de manera sencilla para el planificador evolutivo propuesto y se ha adelantado la posibilidad de considerar la satisfacción de los usuarios en alguna de ellas.

La idea fundamental es mantener un nivel de satisfacción general lo más alto posible, es decir, intentar satisfacer lo máximo posible a todos los clientes por igual. Por supuesto, esto admite matices. Es posible que se quiera maximizar la satisfacción de usuarios importantes o con tareas que requieran prioridad, como por ejemplo las predicciones meteorológicas, a costa de disminuir la satisfacción del resto de usuarios. Considerando estas ideas, pueden configurarse diferentes políticas de planificación. Algunas destacables se presentan a continuación.

6.5.1. Maximizando la satisfacción general

Es la política más natural y la que resulta más equitativa, pues intenta mantener la satisfacción general lo más elevada posible. La idea es medir la satisfacción proporcionada por cada uno de los trabajos en espera por medio de los modelos de satisfacción de los usuarios propietarios de dichos trabajos para, posteriormente, maximizar dicha suma. Durante el proceso de simulación, explicado en la sección 4.3.2.4, y tras la optimización de las solicitudes de recursos, se extraen los datos de espera estimados necesarios para el cálculo de la satisfacción. Una vez obtenidos los valores de satisfacción de cada uno de los trabajos se calcula una media aritmética proporcionándose de esta manera un valor general de satisfacción, que es el que se trata de maximizar. Esto, en términos matemáticos, se puede reflejar de la siguiente manera:

$$SG_T(P) = \frac{\sum_{i=1}^{n} s_{u_i}(t_i)}{n} \tag{6.4}$$

donde P es una planificación dada y n es la cantidad de trabajos en P. s_{u_i} es la función que representa al modelo de satisfacción del usuario u_i propietario del trabajo t_i, de forma que $s_{u_i}(t_i)$ es el valor de satisfacción proporcionado por el trabajo t_i en la planificación P.

Una variante de esta política pasa por considerar un solo valor de satisfacción para cada usuario independientemente del número de trabajos que tengan en la planificación. Esto se hace calculando la media de la satisfacción de los trabajos para cada usuario y después considerando este valor para el cálculo de satisfacción general. Así, la expresión se ve modificada como sigue:

$$SG_U(P) = \frac{\sum_{i=1}^{n_u} \frac{\sum_{j=1}^{n_{t_u}} s_u(t_j)}{n_{t_u}}}{n_u} \tag{6.5}$$

donde n_u es el número de usuarios con trabajos en la planificación P. s_u representa el modelo de satisfacción del usuario u, n_{t_u} es el número de trabajos que el usuario u tiene en P y t_j representa cada uno de los trabajos del subconjunto de trabajos que u tiene en P.

6.5.2. Priorizando usuarios y trabajos

En determinadas ocasiones, o tal vez de manera continuada, será necesario primar a ciertos usuarios o trabajos concretos por algún motivo determinado. Esto se traduce en que hay que considerar la satisfacción de dichos usuarios o trabajos más importante que la del resto por lo que hay que ponderarlas en las políticas de planificación para darles más peso. De esta manera, una política de planificación que tenga en cuenta la satisfacción ponderada por trabajo podría tener la siguiente forma:

$$SGP_T(P) = \frac{\sum_{i=1}^{n} \alpha_i s_{u_i}(t_i)}{\sum_{i=1}^{n} \alpha_i} \tag{6.6}$$

donde α_i es el peso asignado al trabajo t_i. De la misma manera, sería posible priorizar al usuario y no al trabajo individual:

$$SGP_U(P) = \frac{\sum_{i=1}^{n_u} \alpha_i \frac{\sum_{j=1}^{n_{t_u}} s_u(t_j)}{n_{t_u}}}{\sum_{i=1}^{n_u} \alpha_i} \tag{6.7}$$

donde, en esta ocasión, α_i es el peso asignado a cada uno de los usuarios con trabajos en P. Incluso sería posible la creación de una única política que priorizara usuarios y trabajos.

6.5.3. Esquema general de planificación con optimización de la satisfacción

Ya con las aproximaciones propuestas para medir y estimar la satisfacción durante el proceso de planificación con el fin de poder optimizarla, se plasma aquí el proceso general de dicha planificación considerando el ciclo de vida completo desde que un usuario lanza un trabajo al sistema y este termina, y que permite ver dónde y cuándo tienen lugar las tareas asociadas al sistema de estimación de la satisfacción presentado, el SES.

La figura 6.4 refleja dicho ciclo de vida donde, nuevamente, la diversas tareas realizadas por diversas entidades se destacan en un color o en otro: se presentan

en azul las acciones realizadas por el administrador, en verde las realizadas por los usuarios y en rojo las realizadas por los sistemas propuestos en esta tesis. En gris, nuevamente, se indica la acción de ejecución de trabajos que tiene lugar en los nodos de ejecución.

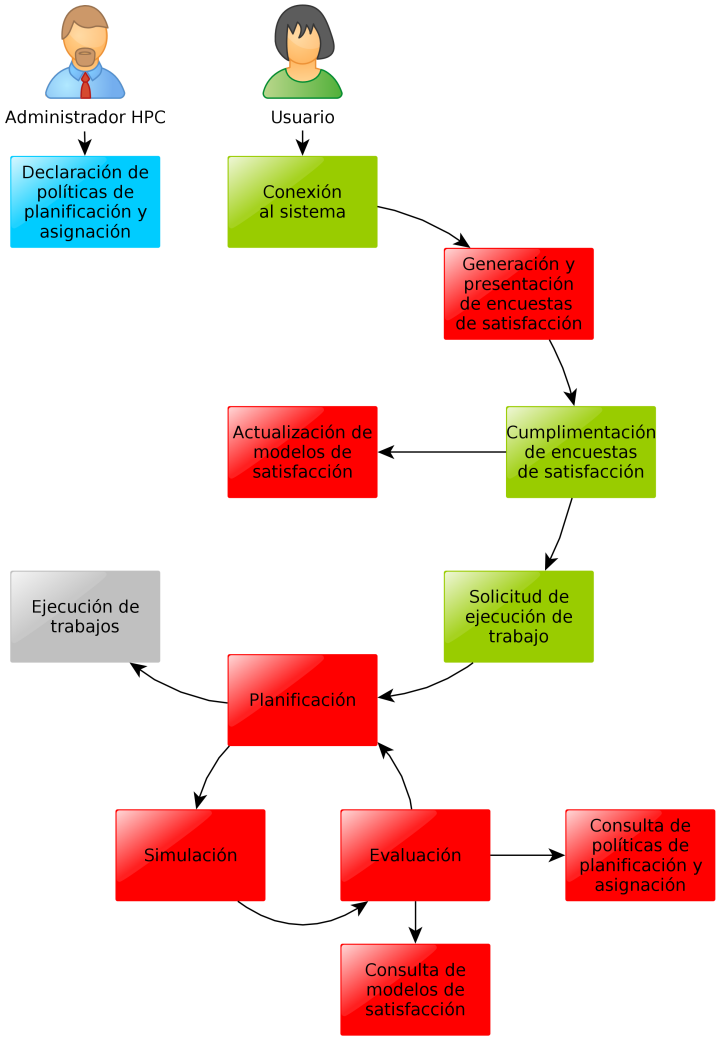

Figura 6.4: Eventos en el proceso de planificación para la optimización de la satisfacción

La tarea principal del administrador es la declaración de políticas tal y como se vio en el capítulo 4. Con las políticas ya determinadas, los usuarios se conectan al sistema para realizar sus solicitudes. Si otros trabajos del usuario han sido puestos en ejecución desde su última conexión, el sistema genera una encuesta que

presenta al usuario para que este la responda. Una vez respondida, los datos son proporcionados al sistema que actualiza el modelo de satisfacción del usuario, que es actualizado en ese momento para poder ser explotado durante la etapa de planificación. Tras la cumplimentación de la encuesta, el usuario puede enviar trabajos al sistema. Cuando el proceso de planificación evolutiva comienza y, durante la etapa de evaluación, se utilizan los modelos de satisfacción actualizados para realizar estimaciones y guiar la evolución hacia la planificación óptima. Cuando termina la planificación algunos de los trabajos de la cola son puestos en ejecución, completando así el ciclo de vida de la planificación con optimización de la satisfacción.

En el capítulo que sigue se presentan diversos experimentos que consideran la medida de la satisfacción y su estimación durante la planificación. En ellos se puede comprobar cómo varían los resultados con diferentes perfiles de satisfacción asignados a distintos usuarios. Para finalizar este capítulo se resumen brevemente los resultados obtenidos.

6.6. Pruebas y resultados

En esta sección se mostrarán los resultados obtenidos de pruebas directamente relacionadas con el sistema de optimización de satisfacción. Primeramente, se aborda el modelado de satisfacción generando modelos para determinados usuarios. Posteriormente, se crea un escenario de planificación en el que se usan dichos modelos para comprobar cómo su percepción del servicio y, por ende, su satisfacción incide directamente en las planificaciones arrojadas por el sistema.

6.6.1. Modelando la satisfacción

Son varias las opciones que se han propuesto en el capítulo 6, en concreto en la sección 6.3. Dado que la aproximación al modelado de satisfacción por encuestas permite evitar el hacer asunciones acerca de la satisfacción de los usuarios se presentan aquí resultados adoptando esta solución. Para esta prueba, además, se mantienen los supuestos ya comentados de que la satisfacción de un usuario es máxima para un trabajo si el tiempo de espera para dicho trabajo es nulo y que, además, la satisfacción se decrementa, o al menos se mantiene constante, con el incremento del tiempo de espera para la puesta en ejecución.

Se han generado datos de un supuesto usuario, usuario A en adelante, que ha cubierto diversas encuestas de satisfacción sobre sus trabajos ejecutados. Estas encuestas se mantienen tan sencillas como las presentadas en la sección 6.3.4, en las que el usuario solamente ha de calificar su satisfacción acerca de cada uno de sus trabajos con una nota numérica entera entre 1 y 7, simbolizando el 1 un estado de plena satisfacción y el 7 uno de insatisfacción máxima. Para la construcción del modelo se ha optado por realizar una regresión polinomial múltiple con dos términos independientes y uno dependiente. Se considera aquí que un polinomio de dicha forma es suficiente para expresar el modelo de satisfacción de usuario y de estimar, mediante interpolación, el nivel de satisfacción para un trabajo bajo unos supuestos

determinados. Uno de los términos independientes es el tiempo de espera del trabajo, definido por la diferencia entre el instante de puesta en ejecución del trabajo y el instante de envío al sistema, denotado por t_w. El otro término es el tiempo de CPU del trabajo, denotado por t_c. Cabe destacar que en lugar del tiempo de CPU utilizado por el trabajo podría usarse el valor proporcionado por el usuario para este parámetro en el momento de hacer la solicitud. El término dependiente es el nivel de satisfacción y es denotado por s. De esta manera, para un trabajo j, existen valores de espera, de CPU y de satisfacción, $t_w(j)$, $t_c(j)$ y $s(j)$, respectivamente. De esta manera, la el nivel de satisfacción del trabajo se puede ver como una función f_s del tiempo de espera y de ejecución de la siguiente forma: $s(j) = f_s(t_w(j), t_c(j))$. Para aproximar dicha función se utiliza un polinomio de grado 2 de forma que, la curva que aproxima la función tiene la forma:

$$f_s(t_w(j), t_c(j)) \approx a_1 t_w(j)^2 + a_2 t_c(j)^2 + a_3 t_w(j) t_c(j) + a_4 t_w(j) + a_5 t_c(j) + c \quad (6.8)$$

En la anterior aproximación, el problema consiste en encontrar las términos a_i y la constante c, cosa que puede hacerse mediante técnicas analíticas o de optimización, por ejemplo. Una vez establecidos, el polinomio permitirá realizar las citadas estimaciones de valores de satisfacción, denotadas por $s_e(j)$. Para comprobarlo, se ha diseñado un caso en el que el usuario ficticio responde encuestas que son recogidas por el sistema. La tabla 6.2 ofrece los valores de tiempo de CPU, $t_c(j)$, tiempo de espera ,$t_w(j)$, (en horas) y el resultado de la encuesta para cada trabajo ($s(j)$) para doce trabajos dados. Con dichos valores se lleva a cabo la regresión para encontrar los términos y la constante necesarios para aproximar la función f_s. Para este caso concreto se han calculado dichos valores mediante un algoritmo *differential evolution*, cuyos parámetros se pueden consultar en la tabla 6.1. El algoritmo, que en este caso ha mostrado mejor comportamiento frente a otros algoritmos como el genético canónico o el micro–genético, proporciona como resultado la siguiente función:

$$\begin{aligned} f_s(t_w(j), t_c(j)) \approx & -0{,}0000199596\ t_w(j)^2 - 0{,}0000002389\ t_c(j)^2 \\ & - 0{,}0000064111\ t_w(j) t_c(j) - 0{,}0000064111\ t_w(j) \\ & + 0{,}0233376170\ t_c(j) + 0{,}8805725189 \end{aligned}$$

La curva resultante se muestra en la imagen 6.5 donde se puede comprobar como varía la satisfacción en función del tiempo de espera y del tiempo de CPU de los trabajos.

Al usar la función calculada para estimar algunos valores de satisfacción para algunos trabajos, $s_e(j)$, se obtienen los resultados ofrecidos en la tabla 6.3.

En estos resultados se puede comprobar como, por ejemplo, y para este usuario concreto, a igual tiempo de CPU la satisfacción es menor (el valor es más alto) si el tiempo de espera es mayor (trabajos 13 y 14), o que, a igual tiempo de espera, la satisfacción es ligeramente mayor si el tiempo de CPU es mayor (trabajos 17 y 19).

Por medio del uso de este mecanismo de estimación de la satisfacción se pueden implementar políticas de maximización de la misma en un planificador de trabajos. En el caso del planificador evolutivo, dicho mecanismo podrá ser utilizado

Parámetro	**Valor**
Longitud del cromosoma	6
Tamaño de la población	100
Estrategia de mutación	Aleatoria
F	0,9
Estrategia de cruce	Cruce binomial
CR	0,9
Número de generaciones	15000

Tabla 6.1: Parámetros utilizados en la evolución del polinomio de satisfacción mediante el algoritmo DE

Trabajo j	$t_c(j)$	$t_w(j)$	$s(j)$
1	10	20	2
2	20	20	1
3	100	70	1
4	50	100	3
5	20	10	1
6	10	5	1
7	15	15	1
8	2000	300	2
9	150	200	3
10	20	100	6
11	10	400	7
12	40	150	4

Tabla 6.2: Resultados de encuestas de satisfacción de un usuario ficticio

Trabajo j	$t_c(j)$	$t_w(j)$	$s_e(j)$
13	10	30	1,564
14	10	60	2,209
15	30	0	1
16	30	120	3,381
17	120	300	5,894
18	200	120	3,302
19	200	300	5,793

Tabla 6.3: Resultados de estimación de satisfacción para un usuario ficticio

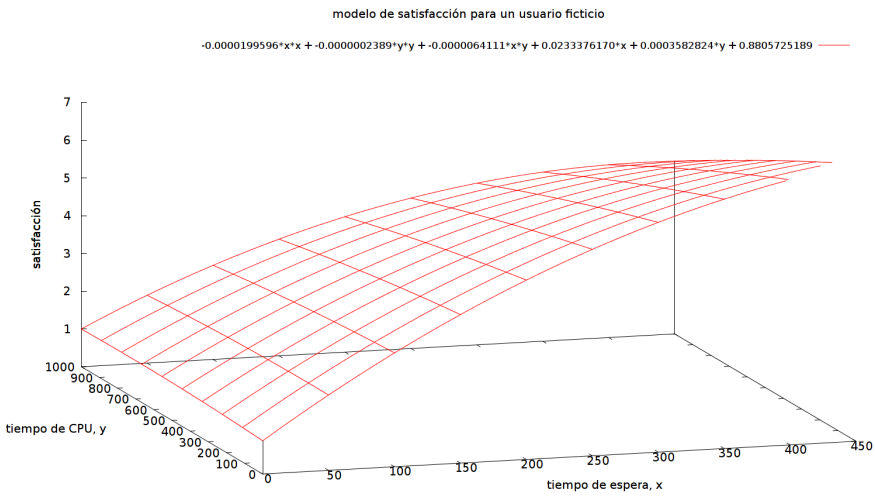

Figura 6.5: Curva del perfil de satisfacción de un usuario ficticio A

como parte de la función de calidad que guiará la evolución hacia una planificación óptima. Para comprobar su funcionamiento y el tipo de resultados que pueden obtenerse se ha diseñado la siguiente prueba.

6.6.2. Planificando para optimizar la satisfacción

La idea principal de este experimento es la de planificar colas con trabajos de diversos usuarios con distintos perfiles de satisfacción y comprobar diversas cosas:

- Qué resultados ofrece el planificador cuando se quiere maximizar la satisfacción global.
- Cómo se comporta el planificador al pretender primar a un usuario sobre el resto.
- Cómo varía la planificación frente a cambios en el perfil de satisfacción.
- Cómo la medida del *makespan* puede quedar descalificada si lo que se pretende es maximizar la satisfacción.
- Adicionalmente, que todas las pruebas llevadas a cabo se hacen sin la necesidad de reconfigurar el planificador, siendo necesario únicamente, y solo en algún caso, el ajuste de la política de planificación.

Para la realización de las pruebas cuyos resultados se presentan a continuación, y de la misma manera que en el experimento previo, se han generado respuestas a encuestas de satisfacción para dos usuarios adicionales con el propósito de construir

Trabajo j	$t_c(j)$	$t_w(j)$	$s(j)$
1	10	50	1
2	20	100	1
3	30	300	5
4	500	100	1
5	500	300	4
6	800	200	1
7	800	400	3
8	1000	500	3
9	1000	200	1

Tabla 6.4: Resultados de encuestas de satisfacción del usuario ficticio 2

Trabajo j	$t_c(j)$	$t_w(j)$	$s(j)$
1	10	5	2
2	10	10	3
3	10	20	5
4	10	25	6
5	30	10	1
6	30	50	4
7	30	70	6
8	100	50	2
9	120	240	5
10	200	100	2
11	200	600	7
12	250	250	3

Tabla 6.5: Resultados de encuestas de satisfacción del usuario ficticio 3

sus perfiles de satisfacción. Uno de los usuarios es el ya presentado en la sección anterior (tabla 6.2) mientras que las respuestas para el resto de usuarios se presentan en las siguientes tablas (6.4 y 6.5).

Las curvas de los perfiles de satisfacción para estos usuarios se muestran en las figuras 6.6 y 6.7 donde se puede apreciar, ya de manera visual, que las curvas que representan los perfiles difieren entre sí de manera notable.

6.6.2.1. Maximizando la satisfacción

En esta prueba se lanzan trabajos de tres usuarios diferentes cuyos perfiles de satisfacción son los recién presentados. Para que la prueba sea fácilmente interpretable por un humano se ha buscado una situación de igualdad para los tres usuarios en el punto de partida. Para ello, se simula que en el mismo momento (el momento 0) llegan cuatro trabajos de cada uno de los usuarios con duraciones estimadas de

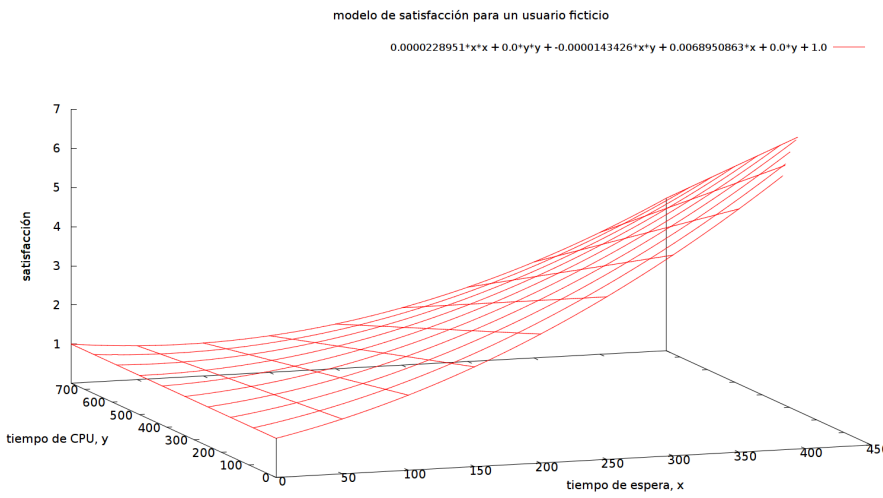

Figura 6.6: Curva del perfil de satisfacción de un usuario ficticio B

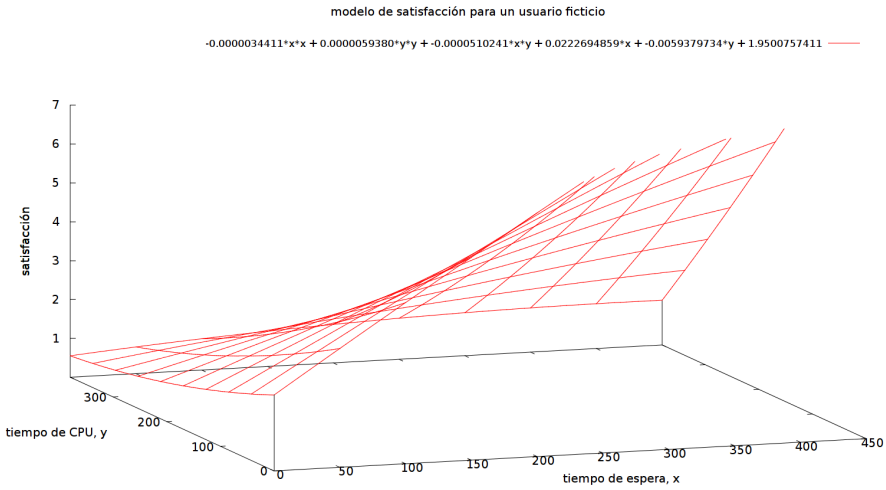

Figura 6.7: Curva del perfil de satisfacción de un usuario ficticio C

20, 40, 70 y 120 horas para cada uno de esos cuatro trabajos por usuario. De esta manera, en la cola de espera existen doce trabajos de idénticas características si se compara un usuario con otro. Los trabajos son de un solo proceso y se utiliza un *cluster* de dos nodos de ejecución de las mismas características que está vacío en el momento 0. Para mejor comprensión se presenta la tabla 6.6, donde para cada trabajo j se muestra su tiempo de CPU ($t_c(j)$), el usuario al que pertenece y el instante de llegada.

Trabajo j	$t_c(j)$	**Usuario**	**Llegada (h)**
1	20	A	0
2	40	A	0
3	70	A	0
4	120	A	0
5	20	B	0
6	40	B	0
7	70	B	0
8	120	B	0
9	20	C	0
10	40	C	0
11	70	C	0
12	120	C	0

Tabla 6.6: Características de los trabajos planificados durante la prueba de maximización de la satisfacción

La planificación se dispara en el momento 0 y se considera instantánea a efectos prácticos con lo que en el mismo momento pueden pasar los primeros trabajos a ejecución. Como política de planificación se utiliza aquella basada en la fórmula 6.4 que persigue la maximización global de la satisfacción de los usuarios, es decir, se minimiza la medida $SG_T(P)$ (ecuación 6.4). Tras el proceso de planificación se logran los resultados presentados en la tabla 6.7. En ella se observa el instante de puesta en ejecución de cada trabajo, el tiempo de espera (estos dos valores coinciden pues el instante de entrada a la cola de espera para todos los trabajos es el momento 0) y la satisfacción estimada $s(j)$ para cada trabajo usando, para su cálculo, el perfil de satisfacción perteneciente al propietario del trabajo. Ha de recordarse que la escala de satisfacción abarca desde el 1 hasta el 7, siendo un 1 la satisfacción máxima y el 7 la insatisfacción máxima. A pesar de que la escala es manejada por los usuarios como etiquetas lingüísticas que llevan asociadas números enteros, el modelo matemático que representa los perfiles es continuo, motivo por el cual se arrojan resultados reales y no enteros. Además, debido a que dichos resultados son interpolaciones pueden suceder situaciones como la del trabajo 1, cuyo valor de satisfacción para una espera de 0 es de 0,888 debiendo ser 1,0. Pudo optarse por redondear los valores por debajo del 1 a 1,0 y por truncar valores por encima del 7 a 7,0, pero para este ejemplo concreto se han tomado las estimaciones de satisfacción tal cual las ofrecieron los perfiles.

En la figura 6.8 se observa el gráfico de la planificación obtenida en el cual se han diferenciado los trabajos de los diferentes usuarios con distintos colores, perteneciendo el color azul al usuario A, el verde al B y el rojo al C. Además, para identificar cada trabajo se etiquetan con el número asociado en las tablas. De su lectura se desprenden varias conclusiones asumiendo que todos los trabajos llegan a la cola en el instante 0 simultáneamente:

- En un primer momento el planificador concede preferencia a los trabajos de

Trabajo j	$t_c(j)$	**Usuario**	**Inicio (h)**	**Espera (h)**	$s(j)$
1	20	A	0	0	0,888
2	40	A	20	20	1,348
3	70	A	60	60	2,206
4	120	A	290	290	5,786
5	20	B	60	60	1,479
6	40	B	130	130	2,208
7	70	B	150	150	2,399
8	120	B	220	220	3,246
9	20	C	0	0	1,834
10	40	C	20	20	2,125
11	70	C	80	80	3,037
12	120	C	170	170	3,968

Tabla 6.7: Resultados de planificación por trabajo al maximizar la satisfacción global

los usuarios A y C frente a los B. Se aprecia en la curva de perfil de dicho usuario (figura 6.6) como su satisfacción decrece de manera muy lenta cuando el tiempo de espera se mantiene por debajo de cierto límite.

- De manera constante, y para cada usuario, los trabajos más cortos son los primeros en finalizar. Esto es debido a que, tal y como están construidos los perfiles, la satisfacción depende del tiempo de espera desde que el trabajo entra en la cola hasta que entra a ejecución, y no hasta que termina. De esta manera, si el planificador colocase antes los trabajos más largos, los trabajos subsiguientes sufrirían esperas más largas bajando el nivel de satisfacción. Además, según se desprende de los perfiles, los trabajos largos tienen más tolerancia a la espera, en otras palabras: cuanto más largo es el trabajo, más lento disminuye el nivel de satisfacción con el incremento del tiempo de espera.
- A pesar de que al comienzo se priorizan los trabajos del usuario A, pues su satisfacción decrece rápidamente desde el primer momento, se puede ver como su último trabajo, el 4 se planifica para su ejecución como último trabajo de los doce. Es debido a que la satisfacción del usuario A decrece más lento con el tiempo respecto de los otros usuarios. Se podría concluir que se trata de un usuario más paciente a partir de cierta espera.

Finalmente, en la tabla 6.8 se presentan las medias aritméticas de satisfacción global (por trabajo) y por usuario, donde se aprecia que, aun con pequeñas variaciones, los niveles de satisfacción por usuario se mantienen en torno a la media.

6.6.2.2. Priorizando a los usuarios

La prueba presentada a continuación consiste en planificar la misma cola que el caso anterior. En esta ocasión se sustituye la política de maximizar la satisfacción

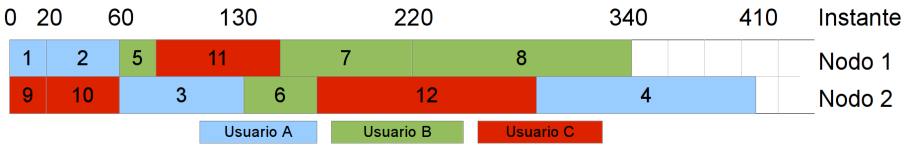

Figura 6.8: Planificación para maximizar la satisfacción global

Usuario j	**Satisfacción media**
A	2,557
B	2,333
C	2,741
Todos	2,544

Tabla 6.8: Características de los trabajos planificados durante la prueba de maximización de la satisfacción priorizando al usuario A

global por otra en la que se pueden priorizar los trabajos de usuarios concretos. Para ello solamente es necesario seleccionar en el planificador evolutivo una política basada en la ecuación 6.7. En dicha política debe maximizarse la media ponderada de las satisfacciones por usuario según las prioridades asignadas, por tanto, el algoritmo evolutivo debe minimizar la medida $SGP_T(P)$ (ecuación 6.6). Hay que tener en cuenta que el administrador del HPC debe establecer, previamente, dichas prioridades a los usuarios.

En este caso concreto se le asigna mayor prioridad al usuario A, fijando su peso α_i a 1,5 mientras que los de los usuarios B y C permanecen a 1, que es el valor por defecto. De la misma manera que en el caso anterior, se presenta la tabla 6.9 con los resultados de planificación y satisfacción estimada por trabajo, el esquema de la planificación en la figura 6.9 y la tabla 6.10 con las medias aritméticas de satisfacción para cada uno de los usuarios.

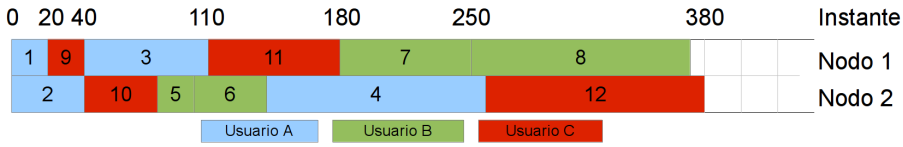

Figura 6.9: Planificación para priorizar al usuario A

Las conclusiones que se extraen en comparación con el caso anterior, en el que se maximizaba la satisfacción por igual de todos los usuarios son:

- Obviamente, al priorizar los trabajos del usuario A, según la planificación, todos sus trabajos entran a la vez o antes a ejecución que en el caso anterior. El

Trabajo j	$t_c(j)$	**Usuario**	**Inicio (h)**	**Espera (h)**	$s(j)$
1	20	A	0	0	0,888
2	40	A	0	0	0,895
3	70	A	40	40	1,788
4	120	A	140	140	3,688
5	20	B	80	80	1,675
6	40	B	100	100	1,861
7	70	B	180	180	2,802
8	120	B	250	250	3,724
9	20	C	20	20	2,257
10	40	C	40	40	2,526
11	70	C	110	110	3,579
12	120	C	260	260	5,289

Tabla 6.9: Resultados de planificación por trabajo al maximizar la satisfacción global

Usuario j	**Satisfacción media**
A	1,815
B	2,516
C	3,413
Todos	2,581

Tabla 6.10: Características de los trabajos planificados durante la prueba de maximización de la satisfacción priorizando al usuario A

caso más evidente es el del trabajo 4, que se adelanta del momento de inicio 290 al 140. Esta prioridad lleva a que el resto de usuarios se vean perjudicados en mayor o menor medida, pues algunos de sus trabajos se retrasan. Es destacable, no obstante, el caso del trabajo 6 perteneciente al usuario B, que pasa del instante de inicio 130 en la planificación previa al instante 100 en la actual. De esta manera, aunque de manera global el usuario B ha sido perjudicado, ha obtenido mejores resultados de satisfacción para ese trabajo en concreto (de 2,208 a 1,861).

- Como se desprende de la tabla 6.10, la satisfacción global es ahora algo menor que en caso anterior, pues al primar a un usuario, y como se acaba de decir, los demás se ven perjudicados. Con esta planificación se logra ahora una media de satisfacción de 1,815 para el usuario A, que es una mejora de 0,742 frente al caso anterior. El usuario B se ve perjudicado levemente pues de un valor de satisfacción de 2,333 pasa a uno de 2,516. El peor caso resulta el del usuario C, pues pasa de un valor de 2,741 a uno de 3,413.

- El *makespan* de esta planificación resulta 380 frente a los 410 del caso anterior. Como ya se ha dicho, el *makespan* no representa una medida adecuada a minimizar para obtener buenas planificaciones y en este ejemplo se puede observar como, al buscar una cosa u otra, esta medida toma diferentes valores. A continuación se verá un ejemplo más relevante de este hecho.

6.6.2.3. Planificación "ciega" o aleatoria

A efectos de comparación con las pruebas anteriores en términos de satisfacción y con un planificador que utilice otra política de planificación, se ha repetido de nuevo la misma prueba con los mismos usuarios y trabajos. La política de planificación, en este caso, es una política "ciega" o aleatoria simulando el hecho de que el planificador no dispone de información suficiente. Por ejemplo, simulando que el momento de llegada de los trabajos es igual para cada uno de ellos, si el planificador no utiliza más información que el instante de llegada no se podría decidir qué trabajo debería pasar primero a ejecución y por tanto, no podría decidirse un orden de ejecución para todos los trabajos. Dicho de otro modo, se da una situación similar a haber ordenado al azar los trabajos en la cola.

Para esta prueba se ejecuta una única planificación que ordena los trabajos aleatoriamente con lo que, los resultados expuestos a continuación no son, con mucha probabilidad, ni los mejores ni los peores que cabría esperar. Los resultados se recogen en la tabla 6.11, el esquema de la satisfacción de la figura 6.10 y la tabla 6.12, que presenta las medias aritméticas de satisfacción para cada uno de los usuarios, tanto para el caso actual como para el caso de maximización de la satisfacción global y el de maximización de la satisfacción primando al usuario A. En esta última tabla se aprecia como la satisfacción media de los usuarios es notablemente inferior para el caso planificado con una política aleatoria, permitiendo concluir que, asumiendo que los modelos de satisfacción generados se aproximen a la realidad, las políticas de planificación que tengan en cuenta la satisfacción guiarán a mejores resultados de planificación desde el punto de vista de la percepción de los usuarios.

Trabajo j	$t_c(j)$	**Usuario**	**Inicio (h)**	**Espera (h)**	$s(j)$
1	20	A	350	350	6,566
2	40	A	140	140	3,735
3	70	A	70	70	2,409
4	120	A	0	0	0,920
5	20	B	220	220	3,562
6	40	B	310	310	5,160
7	70	B	240	240	3,733
8	120	B	260	260	3,893
9	20	C	240	240	6,735
10	40	C	180	180	5,252
11	70	C	0	0	1,564
12	120	C	120	120	3,211

Tabla 6.11: Resultados de planificación por trabajo con política aleatoria

Usuario j	**Satisfacción global**	**Priorizando a A**	**Aleatoria**
A	2,557	1,815	3,407
B	2,333	2,516	4,087
C	2,741	3,413	4,190
Todos	2,544	2,581	3,895

Tabla 6.12: Satisfacción media por usuario y global para diferentes políticas de planificación

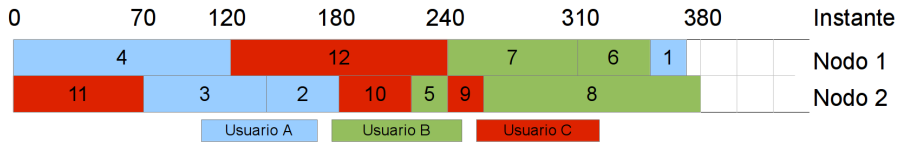

Figura 6.10: Planificación aleatoria (no tiene en cuenta la satisfacción)

En aras de complementar los resultados y conclusiones de estos tres últimos casos de planificación, se presenta en la figura 6.11 un gráfico comparativo de los niveles de satisfacción estimados por trabajo en las tres planificaciones utilizadas: maximización de la satisfacción global, maximización de la satisfacción priorizando al usuario A y planificación aleatoria. En dicho gráfico se representan los trabajos, numerados del 1 al 12 según su identificador en el eje de abscisas mientras que el eje de ordenadas representa el nivel de satisfacción. De esta manera, para cada trabajo se representan tres valores de satisfacción estimados, uno para cada política utilizada. Además, para mejorar la visualización, se han unido los puntos correspondientes a los valores de satisfacción de cada una de las políticas, resultando en una curva roja la política de maximización global, en una curva verde la de maximización priorizando al usuario A y en una curva azul la política aleatoria. Algunas observaciones pueden ser realizadas claramente en el gráfico:

- De manera general, la planificación aleatoria es la que resulta la peor de las tres. Este hecho puede observarse en que el área contenida bajo la curva azul se aprecia significativamente mayor que cualquiera de los otros dos casos.

- Como era de esperar, en la política aleatoria, algunos trabajos reciben mayor satisfacción en comparación con las otras políticas (trabajos 4, 11 y 12) y otros, en cambio, tienen estimaciones peores (trabajos 1, 6 y 7, por ejemplo). Además, tan solo se mejora en tres de los trabajos, empeorando el resultado en los nueve restantes.

- Las curvas de satisfacción de las dos primeras políticas (curvas roja y verde) tienen formas muy similares, apreciándose valores de satisfacción muy similares para cada uno de los trabajos.

- Se observa claramente cómo los trabajos correspondientes al usuario A (trabajos del 1 al 4) tienen valores ligeramente inferiores en la curva verde (política de maximización priorizando al usuario A) que en la roja (maximización global), mientras que para los trabajos del resto de usuarios (del 5 al 12) sucede exactamente lo contrario. Visualmente se aprecia como se han mejorado los valores del usuario A a costa de empeorar los del resto de usuarios, eso sí, sin que el daño sea demasiado grave.

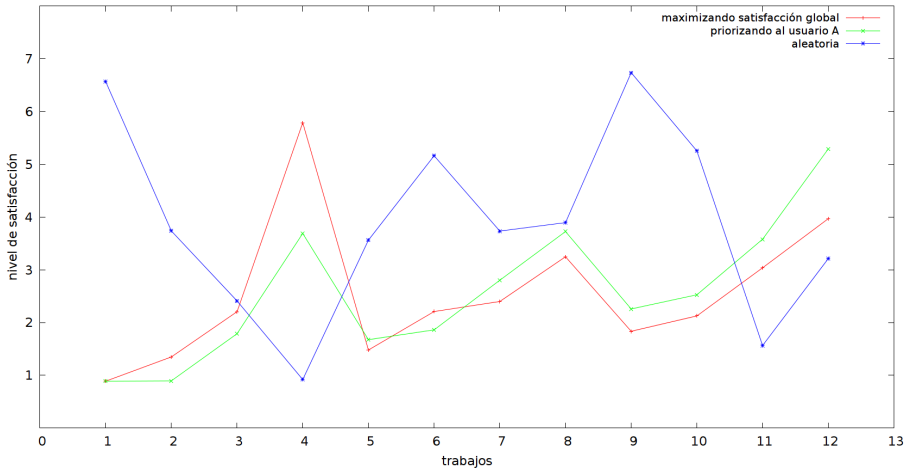

Figura 6.11: Comparativa de satisfacción por trabajo con diferentes políticas de planificación

6.6.2.4. Variando el perfil de satisfacción

Aunque se ha comprobado de manera secundaria en las pruebas anteriores el efecto de poseer distinto perfil de satisfacción en las planificaciones, en este nuevo experimento se comprueba explícitamente cómo afecta a las planificaciones un cambio en el perfil de satisfacción de un usuario. Para ello, y con el mismo *cluster* utilizado en los casos anteriores, dos usuarios diferentes lanzarán trabajos iguales a los de dichas pruebas. No obstante, la planificación se realizará bajo dos supuestos diferentes. En el primero de ellos, un usuario tendrá el perfil del usuario A ya presentado (figura 6.5) y el otro usuario tendrá el perfil del usuario B (figura 6.6). En el segundo supuesto, el primer usuario mantiene el perfil A mientras que el del segundo usuario será el del usuario C (figura 6.7). Planificando en ambos supuestos se podrán contrastar las diferencias entre los diferentes resultados arrojados por el sistema. La política utilizada en esta ocasión es la de maximización de la satisfacción global. En ambas pruebas, para asegurar la consecución del óptimo, se ha utilizado para la planificación el algoritmo de fuerza bruta.

Concretando, para las planificaciones se lanzan ocho trabajos, cuatro por usuario, que entran en el sistema en el instante 0. El conjunto de trabajos del primer usuario es de idénticas características al del segundo usuario. De esta manera, y para el primer supuesto, los trabajos lanzados al sistema por los usuarios se reflejan en la tabla 6.13. En la figura 6.12 se puede observar la planificacion obtenida, cuyos resultados por trabajos quedan recogidos en la tabla 6.14. La conclusión principal es que el planificador, de manera general, decide anteponer la entrada de los trabajos del usuario 1, con el perfil de satisfacción A, a los del usuario 2, cuyas medias de nivel de satisfacción son 1,528 y 1,723 respectivamente.

Para el segundo supuesto se repite la prueba pero sustituyendo el perfil de sa-

Trabajo j	$t_c(j)$	**Usuario**	**Perfil**	**Llegada (h)**
1	20	1	A	0
2	40	1	A	0
3	70	1	A	0
4	120	1	A	0
5	20	2	B	0
6	40	2	B	0
7	70	2	B	0
8	120	2	B	0

Tabla 6.13: Características de los trabajos planificados durante la prueba de variación en los perfiles de satisfacción (supuesto 1)

Trabajo j	$t_c(j)$	**Usuario**	**Inicio (h)**	**Espera (h)**	$s(j)$
1	20	A	0	0	0,888
2	40	A	0	0	0,895
3	70	A	20	20	1,354
4	120	A	100	100	2,977
5	20	B	40	40	1,301
6	40	B	60	60	1,462
7	70	B	90	90	1,716
8	120	B	160	160	2,414

Tabla 6.14: Resultados de planificación por trabajo variando el perfil de satisfacción (supuesto 1)

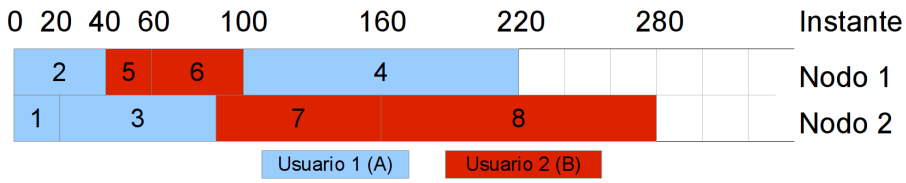

Figura 6.12: Planificación variando el perfil de satisfacción (supuesto 1)

Trabajo j	$t_c(j)$	**Usuario**	**Perfil**	**Llegada (h)**
1	20	1	A	0
2	40	1	A	0
3	70	1	A	0
4	120	1	A	0
5	20	2	C	0
6	40	2	C	0
7	70	2	C	0
8	120	2	C	0

Tabla 6.15: Características de los trabajos planificados durante la prueba de variación en los perfiles de satisfacción (supuesto 2)

tisfacción del usuario 2 que ahora pasa a ser el perfil C y no el B, con los que los trabajos que intervienen en la prueba son los de la tabla 6.15. Al planificar se obtiene el resultado de la figura 6.13. Los resultados de planificación por trabajo se pueden observar en la tabla 6.16. En esta ocasión, y aunque el usuario con el perfil C posea, de media, una menor satisfacción (2,499 frente al 1,990 del usuario 1) el planificador determina una situación de igualdad como mejor resultado. Independientemente de este efecto, lo más importante a destacar aquí es que queda de manifiesto como los perfiles de satisfacción determinan el comportamiento del planificador, que arroja resultados diferentes en cada caso para intentar alcanzar el objetivo de satisfacer a todos los usuarios por igual.

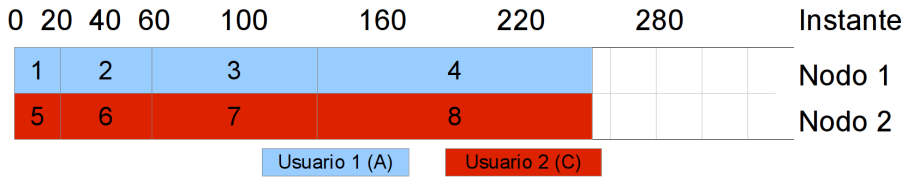

Figura 6.13: Planificación variando el perfil de satisfacción (supuesto 2)

Trabajo j	$t_c(j)$	**Usuario**	**Inicio (h)**	**Espera (h)**	$s(j)$
1	20	A	0	0	0,888
2	40	A	20	20	1,348
3	70	A	60	60	2,206
4	120	A	130	130	3,517
5	20	C	0	0	1,834
6	40	C	20	20	2,125
7	70	C	60	60	2,673
8	120	C	130	130	3,364

Tabla 6.16: Resultados de planificación por trabajo variando el perfil de satisfacción (supuesto 2)

6.7. Resumen y conclusiones

La incorporación de criterios subjetivos en las políticas de un planificador hace necesario un mecanismo de medida para este tipo de criterios. En este trabajo se ha considerado el concepto fundamental de satisfacción como una medida subjetiva de percepción de calidad de servicio por parte de los usuarios. La principal dificultad de manejar conceptos como el de satisfacción radica precisamente en la manera de medirla. Dejando de lado técnicas de obtención de datos biométricos, monitorización de constantes vitales o detección de alteraciones en esa línea, por no ser aplicables en este ámbito, los principales trabajos [88, 89, 92, 95, 96] realizados pasan por el diseño y construcción de encuestas que, una vez contestadas por los usuarios, permiten conocer el nivel de satisfacción en el uso de un sistema así como detectar problemas concretos con el fin de corregirlos. A pesar de que con el tiempo se han ido detectando los factores principales para el diseño de encuestas consiguiendo reducir el número de elementos, estas no dejan de ser un mecanismo invasivo y pesado para muchos usuarios.

Previamente se detallan algunas propuestas para la caracterización y medida de la satisfacción como parte de un sistema de optimización de la satisfacción, el SES. Estas propuestas parten de la idea de que el tiempo de espera, aunque puedan existir otros, es el principal factor que determina la satisfacción del usuario: a mayor espera, menor satisfacción. De manera resumida, una de las propuestas pasa por asignar perfiles de satisfacción a los usuarios que vienen determinados por una función que relaciona el tiempo de espera con la satisfacción, de forma que puede usarse esta función para predecir la satisfacción de cada usuario para una planificación dada. Esta aproximación adolece de dos problemas, por un lado, la forma de crear dichos perfiles y por otro, la de asignarlos a usuarios concretos. Se propone, por tanto, otra aproximación basada en encuestas, pues la manera más directa y precisa de determinar la satisfacción de los usuarios es preguntarles directamente. Para hacer eso se atiende a determinadas características:

- La encuesta debe ser lo menos invasiva posible, por tanto ha de ser corta y concisa.

- No interesa determinar el porqué de la satisfacción o insatisfacción del usuario, solo su nivel, pues el sistema se adaptará para optimizar su satisfacción sin necesidad de entrar en ningún tipo de análisis de causas.

Dado que las encuestas son un instrumento probado de medida de satisfacción y que el ámbito concreto de este trabajo es más reducido que el de un sistema de información en general, esta propuesta resulta viable, como se demuestran en [131], donde se han medido con éxito determinadas emociones utilizando escalas de diferencial semántico [127]. Entendiendo la satisfacción como una emoción y extrapolando dichas investigación al ámbito de los sistemas informáticos, se ha propuesto la aplicación de una encuesta mínima compuesta de una única pregunta que los usuarios deben contestar cuando se conectan al sistema siempre y cuando haya terminado alguno de los trabajos enviados. Dicha pregunta captura el nivel de satisfacción del usuario ante las condiciones de ejecución de sus trabajos. Esta información pasa a ser parte del modelo de satisfacción, resuelto mediante una regresión polinomial ajustada con un algoritmo DE. El modelo es actualizado de manera *on–line* para ser utilizado durante la etapa de planificación.

Esta manera de operar arroja resultados en las pruebas en las que se puede ver cómo, en función del perfil de satisfacción que posean los usuarios (determinados por un conjunto de respuestas a encuestas ficticio), las planificaciones obtenidas tratan de satisfacer de manera global al conjunto de usuarios. Es importante destacar que la política utilizada en las planificaciones no especifica ningún criterio objetivo de manera explícita, sino que únicamente se utilizan valores extraídos de los modelos de satisfacción generados con supuestas respuestas de los usuarios sobre su satisfacción ante el comportamiento del sistema. Esto es un punto fundamental y diferenciador de este trabajo. Además, se ha comprobado cómo dotando de más prioridad a uno de los usuarios, el planificador proporciona una planificación diferente para incrementar la satisfacción de dicho usuario, sin necesidad de decirle explícitamente que adelante sus trabajos, por ejemplo, pues no hay por qué saber que eso es lo que va a satisfacer más al usuario.

Capítulo 7

Optimización de la asignación de recursos

Hay que contar con las trampas del camino, así como con las ventajas que te brinde.

En muchos escenarios, sobre todo en aquellos con tecnologías *grid*, la heterogeneidad de los recursos así como su disponibilidad introducen nuevos factores a tener en cuenta a la hora de planificar. Seleccionar el mejor recurso para la ejecución de un trabajo es una decisión fundamental que debe tomar el planificador, que ya no puede limitarse a elegir, simplemente, el mejor trabajo para poner en ejecución. Además, la asignación de recursos a los trabajos tiene relación directa con los intereses del centro, pues de la propia asignación derivan, entre otras cosas, gastos económicos y energéticos. El control y optimización de estas variables incidirá directamente en la satisfacción del centro.

El presente capítulo recoge las propuestas relacionadas con la optimización de la asignación de recursos durante el proceso de planificación. Asimismo, se introducirán algunas ideas acerca de cómo modelar los recursos en diferentes escenarios para proporcionar al planificador mayor información con el fin de lograr mejores resultados. También se presentará la forma de modificar las políticas de asignación de recursos de manera declarativa en el planificador explicado en este trabajo.

7.1. Preámbulo

Hasta ahora se ha presentado un sistema que, mediante modelado de comportamiento y satisfacción de los usuarios y técnicas de optimización, conforma un planificador para colas de procesos en un entorno de computación de altas prestaciones. Dicho planificador resuelve una serie de problemas presentes en los mencionados

ambientes guiando el proceso de planificación para lograr cumplir con las expectativas de los usuarios así como las de los centros de HPC. No obstante, todavía queda un aspecto que cubrir para poder considerarlo un sistema completo que ataque en todos los frentes persiguiendo así un resultado óptimo.

En trabajos como el de Hovestadt *et al.* [132] queda patente la necesidad tener en cuenta las asignaciones de trabajos a un conjunto de recursos heterogéneos que el planificador hará en el futuro para tomar las mejores decisiones, tal y como se ha hecho en esta tesis. Además y a menudo, los recursos disponibles para computación sufren de diversas incidencias que afectan a su disponibilidad o a su rendimiento general, sobre todo en entornos *grid* donde dichos recursos son heterogéneos, pueden estar ubicados en diferentes lugares del mundo y son gestionados por diferentes administradores y organizaciones bajo distintas políticas. Entre estas incidencias se pueden encontrar averías, no disponibilidad o cambios en la carga de trabajo, por ejemplo. Este hecho lleva a que los sistemas de planificación tradicionales solo puedan tener en cuenta la información y el estado actual de los recursos para llevarla a cabo, conduciendo a planificaciones que puede que se alejen bastante de la óptima.

También sucede que las políticas clásicas de asignación de recursos (consultar sección 4.1.4) resultan demasiado rígidas y poco adaptables a la evolución del estado de los recursos. No puede asumirse que utilizando una política clásica voraz que tome siempre el primer nodo disponible, por ejemplo, para la ejecución del primer trabajo pendiente en la cola de espera, la planificación de los trabajos resulte óptima, pues la elección depende tanto del propio trabajo como del resto que queden en espera y de los demás recursos disponibles. Este problema cobra mayor importancia en sistemas de HPC heterogéneos. Así, el adoptar una política voraz de asignación de recursos puede minar el proceso de planificación de los trabajos.

Otro problema radica en que, de manera general, los planificadores no cambian su política de asignación dinámicamente para adaptarse a las circunstancias. Existen algunos trabajos que analizan este hecho y proponen alternativas, como hace Achim Streit en [6, 7] y ya comentadas en la sección 3.3.

Además, la introducción de políticas que persigan nuevos objetivos puede resultar complicada. Entre estos nuevos objetivos podrían considerarse el coste económico, la energía consumida[1], o el uso balanceado y equitativo de los recursos, por ejemplo. En concreto, es posible destacar una política que atañe directamente a la temática de este estudio. Se trata, nuevamente, de la satisfacción de los usuarios. ¿Qué política de asignación de recursos deberían configurar los administradores de un centro de HPC para optimizar la satisfacción de sus usuarios?

Para ilustrar los diversos efectos causados por la aplicación tradicional de políticas de asignación de recursos se presenta a continuación un pequeño ejemplo. Se trata del caso de un *cluster* con dos nodos de proceso (1 y 2) de diferentes características. Simplificando, el nodo 1 es capaz de procesar trabajos alrededor de un 25 % más rápido que el nodo 2. En el momento de planificar y de servir procesos, en la cola de espera se encuentran tres trabajos *A*, *B* y *C*, en ese mismo orden. Suponiendo una política de asignación de recursos que asigna los recursos en un

[1] En la actualidad se busca el uso eficiente de los recursos para minimizar el impacto medioambiental. Es lo que se conoce como *Green Computing* o *Tecnologías Verdes*.

orden preestablecido (1 y 2) o una en la que se asigne primero el nodo libre más potente, se obtendría la planificación de la figura 7.1. En ella se puede observar

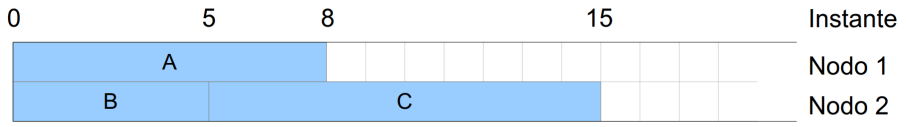

Figura 7.1: Ejemplo 1 de planificación para una política de asignación de recursos (1)

cómo el primer trabajo de la cola, el *A* se asigna al nodo 1 siguiendo la política. En dicho nodo, el trabajo tarda 8 unidades de tiempo (horas, por ejemplo) en finalizar. A su vez, el siguiente trabajo pasa a ejecución en el nodo 2, tardando 5 horas en finalizar. En cuanto este trabajo termina, el siguiente y último trabajo se pone en ejecución en el nodo libre. De esta manera, el trabajo *C* tarda 10 horas en finalizar. Si se toma la misma situación pero alterando la política de asignación de recursos para que sean asignados en el otro orden posible (2 y 1) la planificación resultante sería la reflejada en la figura 7.2 (que representa la ocupación de los recursos en el tiempo). En ella se observa como la asignación de recursos a los trabajos ha

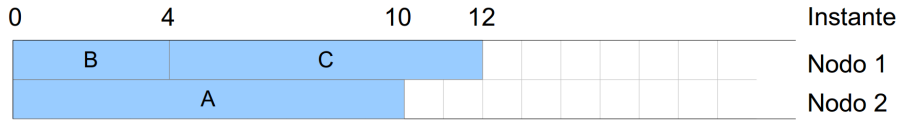

Figura 7.2: Ejemplo 1 de planificación para una política de asignación de recursos (2)

cambiado y con ello el tiempo en que empiezan y finalizan. Se puede observar cómo los trabajos utilizan más tiempo para su ejecución en el recurso 2 que el que utilizaban en el recurso 1. De la misma manera, los trabajos utilizan menos tiempo en el recurso 1 que el que necesitaban en el 2. Si se supone que la mejor política de asignación es la primera (se asigna antes el nodo más potente) y comparando ambas respuestas del planificador, se pueden hacer varias observaciones y lecturas interesantes:

- El *makespan* (tiempo que se tarda en finalizar desde que comienza) total de todos los trabajos con la primera política es 15 horas frente a las 12 horas de la segunda. Esto arroja *throughputs* (número de trabajos servidos por unidad de tiempo) de 0,20 y 0,25, respectivamente.
- Mientras que los tiempos de espera para el comienzo de la ejecución de los trabajos A y B son iguales para ambas políticas, el trabajo C empieza antes en el segundo caso. Este hecho podría repercutir positivamente en la satisfacción del usuario propietario del trabajo.

- El tiempo total que los recursos están ocupados es de 23 horas en el caso de la primera política, mientras que en el de la segunda, este tiempo es de 22 horas.

Se puede concluir, a la vista de estos resultados, que la segunda política resulta mejor que la primera (al menos para este ejemplo) aunque en un primer momento no resulte intuitivo. Esto demuestra que limitarse al uso de una única política de asignación de recursos, por conveniente que parezca, puede producir resultados no óptimos, como ya estudió Streit en los trabajos previamente comentados. Lo lógico sería dar un paso más y pensar en algún tipo de política dinámica que se ajuste en función de los objetivos perseguidos en un momento dado.

Si bien es cierto que la planificación A, B, C no es la óptima para la primera de las políticas y que sí lo resulta para la segunda, este ejemplo demuestra que el proceso de planificación tiene que tener en cuenta la política de asignación de recursos para lograr un mejor resultado.

Otra manera de verlo sería la siguiente. Asumiendo el mismo entorno que el utilizado en el ejemplo anterior se pueden suponer dos trabajos A y B. El trabajo A tardaría 8 horas en ejecutarse en el nodo 1 y 10 horas en el nodo 2. El trabajo B tardaría 12 y 15 horas respectivamente. Bajo una política que asigne primero el nodo más potente libre podría darse el siguiente caso:

1. El trabajo A llega al *cluster*, que se encuentra vacío, y pasa a ejecución en el nodo asignado por la política, el 1.
2. Una hora después, llega el trabajo B, que se pone en ejecución en el nodo 2.
3. En el instante 8 termina la ejecución del trabajo A dejando libre el nodo 1.
4. En el instante 16 termina la ejecución del trabajo B dejando libre el nodo 2.

Esta evolución puede verse en la figura 7.3. Si se utilizase otra política, la de asignar

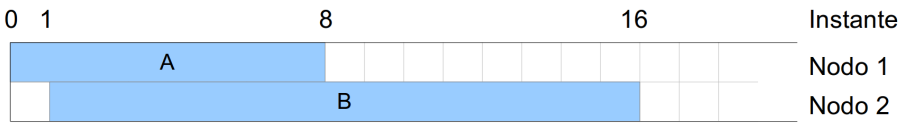

Figura 7.3: Ejemplo 2 de planificación para una política de asignación de recursos (1)

antes el nodo más lento, el escenario transcurriría como sigue:

1. El trabajo A llega al *cluster*, que se encuentra vacío, y pasa a ejecución en el nodo asignado por la política, el 2.
2. Una hora después, llega el trabajo B, que se pone en ejecución en el nodo 1.

3. En el instante 12 termina la ejecución del trabajo *A* dejando libre el nodo 2.

4. En el instante 13 termina la ejecución del trabajo *B* dejando libre el nodo 1.

Nuevamente, se puede comprobar de manera visual este comportamiento en la figura 7.4.

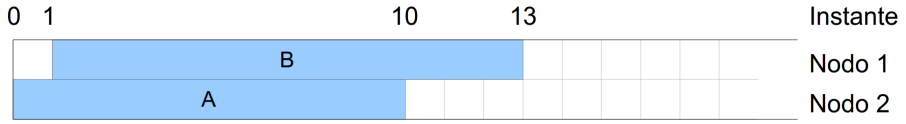

Figura 7.4: Ejemplo 2 de planificación para una política de asignación de recursos (2)

Lo que primero salta a la vista es el hecho de que, para una política de planificación dada, por ejemplo, la minimización del *makespan*, resulta peor la que a priori parece la mejor política de asignación de recursos, que es la de asignar primero el nodo más potente, haciendo la comparación en términos de *makespan*, *throughput* y ocupación de recursos, pero podría considerarse cualquier otro objetivo, como la satisfacción de los usuarios. Claramente arroja mejores resultados la política de asignar antes el nodo más lento. En sí puede parecer sorprendente y contra–intuitivo pero resulta más evidente al comprender que se está usando el nodo más potente para servir un trabajo que requiere menos cómputo. Este ejemplo también lleva a pensar que la mejor decisión para el momento actual puede resultar contraproducente en el futuro, por lo que resulta interesante el uso de políticas conservadoras que, por ejemplo, no asignen siempre el nodo más potente disponible. Es decir, se podría asignar dicho nodo en caso de que el trabajo necesitase mucho tiempo de cómputo. Generalizando, la asignación podría pasar por seleccionar recursos de una potencia acorde con los requisitos del trabajo. En el capítulo 10.1 se sugiere una línea de trabajo futuro relacionada con la predicción de la llegada de trabajos para lidiar con este problema desde otra óptica.

A pesar de lo sintético de estos pequeños ejemplos es fácil comprender que la casuística relativa a la planificación de trabajos es tal que se hace necesario un mecanismo diferente para la asignación de recursos durante la planificación, pues en un caso real con cientos de trabajos y recursos, estos y otros casos similares se repetirán en diversas ocasiones. La extrapolación de dichos casos puede servir de contraejemplo a cualquier política de asignación no dinámica y que no tenga en cuenta las asignaciones futuras.

7.2. Principales ideas

La idea aquí se basa en tener otro tipo de control sobre la asignación de recursos en el momento de la planificación buscando un triple objetivo: mejorar el uso

de los recursos en sí para conseguir un aprovechamiento concreto, mejorar las planificaciones para maximizar la satisfacción de los usuarios y, finalmente, facilitar la declaración de políticas de planificación considerando los intereses del centro de HPC para maximizar su satisfacción. De esta manera, se plantea en este trabajo una mejora más que complemente el sistema de planificación propuesto hasta el momento y que permita, además, la implantación de políticas de asignación de recursos diferentes de las clásicas y que atiendan a nuevas necesidades. Se busca además, que tras dicha implantación, la realización de estas políticas se haga de manera declarativa, como sucede con las políticas de planificación y como se detalla en el capítulo 4.

La idea principal es la de construir modelos de los recursos que permitan predecir diversos parámetros y valores para poder considerarlos durante el proceso de planificación evolutiva y comprobar si una planificación dada es adecuada. Las ideas y el trabajo expuestos a continuación se centra en las características de carga de CPU y disponibilidad de los recursos ya que son las que mayor complejidad aportan a la planificación. Los modelos serán utilizados por el planificador para realizar diferentes medidas que sirvan de ayuda al planificar para lograr unos objetivos concretos. A continuación se presenta un desarrollo para la construcción de estos modelos y del sistema que los genera y actualiza, sistema que se ha denominado SOAR, Sistema de Optimización de Asignación de Recursos, que es uno de los cuatro subsistemas que conforman el EvoProc.

7.3. Modelando los recursos

De manera similar al enfoque adoptado para el modelado de los usuarios, explicado en la sección 5.3.1, se propone en esta un sistema de modelado de recursos. Dicho sistema ha de contar con dos partes claramente diferenciadas. Por un lado, los modelos en sí, que serán utilizados para realizar las predicciones de disponibilidad y carga necesarias para la optimización de las planificaciones. Por otro lado, es necesario un sistema de obtención de los citados modelos que acometerá tal tarea de manera automática y sin intervención humana. No solo la obtención de los modelos será contemplada por este sistema, sino que también ha de tener la responsabilidad de actualizar los modelos de manera dinámica y adaptativa para garantizar que los modelos a utilizar por el planificador siempre se encuentren actualizados.

Para poner en orden algunas ideas y comprender las posteriores explicaciones se deben tener en cuenta otras dos asunciones importantes:

- Al igual que el caso de sistemas de HPC homogéneos, cada unidad de proceso del conjunto de recursos en un entorno *grid* o *cluster* heterogéneos tendrá capacidad para la ejecución de un único trabajo. La unidad de proceso podrá recibir trabajos siempre que esté activo dentro de un horario establecido y no esté ejecutando ningún otro enviado al sistema previamente. A la finalización del trabajo, la unidad podrá albergar la ejecución de uno nuevo.
- Determinados recursos del entorno pueden ser compartidos o donados por

organizaciones o usuarios particulares como parte de un programa de computación voluntaria. Debido a esto es posible que dichos recursos puedan estar ejecutando *software* ajeno al sistema de HPC. Esto puede hacer variar la capacidad de cálculo de dichos recursos en determinados momentos, lo que hace que la ejecución de un mismo trabajo pueda durar tiempos diferentes de ser ejecutado en distintos momentos en el mismo recurso.

De hacerse falsa la primera asunción sería necesario adaptar el trabajo expuesto a continuación pues necesitaría diversas modificaciones. No obstante el caso aquí tratado corresponde con la filosofía adoptada por la mayoría de las organizaciones y centros de HPC. Respecto de la segunda asunción, si se supusiera que no existe otro *software* en ejecución, se contaría con la capacidad de cálculo máxima el cien por cien del tiempo que el recurso estuviera activo. Este es solo un caso particular de la asunción, con lo que las soluciones propuestas resultarían igualmente válidas.

7.3.1. Los modelos de recursos

En un entorno *grid* es posible encontrarse con recursos para ejecutar trabajos, ya sean nodos dedicados u ordenadores personales cedidos por usuarios particulares, que difieren mucho entre sí, no solo en forma, sino en capacidad de cálculo. No obstante es fácil encontrarse con muchos recursos iguales repartidos por diferentes partes del globo y gestionados por distintas organizaciones. Esto hace que la capacidad de cómputo de cada uno de estos recursos pueda diferir del resto de sus semejantes. Todo esto unido puede llevar a una situación en la que la mayoría de los recursos son diferentes entre sí pudiéndose incurrir en un error grave de no hacer tal asunción. Es por esto que la solución planteada aquí pasa por generar un modelo para cada uno de los recursos disponibles con la excepción de aquellos *clusters* homogéneos existentes en el *grid* y que cumplan las condiciones manejadas hasta el momento en este trabajo, cuyos nodos de ejecución podrán, lógicamente, ser representados por un único modelo.

Pero, ¿qué es un modelo de recurso? Desde el enfoque actual de la planificación, donde se quieren utilizar políticas expresadas de manera declarativa, cosa que se consigue mediante el uso de funciones de *fitness* o calidad en un algoritmo evolutivo, un modelo de recurso es una caja negra que responde a preguntas de este estilo:

1. ¿Podría iniciarse en el instante t y ejecutarse con éxito el trabajo A (un trabajo con unos requisitos de recursos concretos) en el recurso al que representa?
2. De poder ejecutarse el trabajo A iniciándose en el instante t, ¿en que instante terminaría su ejecución?

Para responder a tales cuestiones, cuyas respuestas serían muy útiles a la hora de planificar (para que no quede lugar a dudas, se profundizará en este aspecto en la sección 7.4), un modelo necesita conocer información histórica acerca de la disponibilidad del recurso y de su carga de trabajo en cada momento para poder

efectuar las predicciones que se pretenden. Por supuesto, muchos otros factores podrían afectar al estado y por tanto a dichas predicciones, tales como los fallos que puedan suceder en los recursos y otras circunstancias excepcionales. Suponiendo esta información como conocida, el modelo lleva a cabo las estimaciones en el momento de las peticiones y utiliza para ello técnicas similares a las planteadas en el SOS u otras basadas en otras herramientas, inferencias o sistemas estadísticos.

Antes de profundizar en la manera en que estos modelos están construidos y su forma de funcionamiento, y teniendo en cuenta que la capacidad de los recursos es diferente así como la segunda asunción expuesta al comienzo de este capítulo, es necesario revisar el concepto de duración de los trabajos. Es evidente que en un entorno homogéneo el tiempo de duración de un trabajo se puede mantener estable entre recursos y con independencia del momento en que se ejecute (siempre y cuando la duración del trabajo sea determinista). Esto hace que se pueda hablar de la duración del trabajo como si de una constante se tratara (al menos dentro de un mismo entorno) y que, por tanto, puede ser solicitada al usuario una estimación del tiempo de ejecución de sus trabajos. Y con esto se ha jugado hasta ahora, momento en el que aparece un nuevo problema pues la duración de un mismo trabajo es diferente en función de los recursos asignados y del momento en que se ejecute. Esto hace que no se pueda solicitar al usuario una estimación de la duración, en especial teniendo en cuenta que no se puede, a priori, asignar unos recursos concretos y comunicárselos al usuario para que pueda hacer sus estimaciones. Se hace necesario un mecanismo de estimación de la duración de los trabajos en función de la capacidad y carga totales de un recurso. Dicho mecanismo debe ser capaz de extrapolar la duración de un trabajo en un recurso con un estado concreto a partir de valores de trabajos similares en otros recursos y sus estados. A su vez, este mecanismo sería el encargado de responder a la segunda de las preguntas planteadas para los modelos de recursos. Para materializar todo esto se propone la siguiente aproximación.

7.3.2. Creación y actualización de los modelos

La idea principal se basa en muestrear el estado de los recursos con una frecuencia determinada, por ejemplo cada quince minutos o media hora (se podría pensar en variar dinámicamente la frecuencia de muestreo para sistemas muy poco estables) obteniendo los parámetros que se deseen modelar. En este caso, el estado de actividad del nodo de proceso (activo o no) y la carga actual (en porcentaje). Recogiendo suficientes datos de muestras anteriores se puede elaborar un perfil del recurso útil para las estimaciones.

El problema del muestreo y recogida de datos del estado de equipos informáticos está resuelto mediante técnicas de gestión de redes de comunicaciones y a través del uso de *software* basado en los protocolos SNMP (*Simple Net Management Protocol*) o CMOT (*Common Management Information Protocol over TCP/IP*). De manera general, el *software* de gestión de colas implementa mecanismos para obtener estas medidas, no obstante, sería una tarea sencilla el desarrollo de un pequeño módulo para la obtención de las muestras en caso de hacerse necesario.

La carga de trabajo de una CPU se mide en porcentaje, asumiendo que el resto está disponible para la ejecución del trabajo (obviando la carga que aporte el sistema operativo). Así, tomando muestras periódicas de la carga de una CPU se puede promediar la carga en un intervalo para conocer la capacidad que el recurso puede brindar a un trabajo concreto en ese intervalo. Previamente es necesario, como se dijo, tener datos históricos suficientes que permitan al sistema perfilar la carga del recurso. Para comprender mejor la idea se presenta a continuación un caso práctico. Se puede suponer una CPU que forma parte de un programa de computación voluntaria como uno de los recursos de un *grid*. Dicha CPU se encuentra activa y disponible para la ejecución de trabajos las veinticuatro horas del día y los siete días de la semana. En principio, y como el propietario del recurso hace uso de él, interesa generar un perfil de su carga de trabajo y hacerlo teniendo en cuenta los fines de semana ya que es muy probable que varíe el patrón de carga. Para generar este perfil se monitoriza el recurso durante un tiempo mínimo que puede establecerse en, por ejemplo, tres semanas, por considerarse un intervalo de tiempo suficiente para la recogida de los datos que generen un buen modelo. Durante este periodo de tiempo se toman muestras del valor de la carga de CPU cada media hora, cosa que se puede hacer a las horas en punto y a las medias. Un lunes a las 00:00 horas comienza el muestreo, que se completa pasadas tres semanas. Se adquieren 48 muestras por día, lo que hace un total de 1008 muestras. Con esta cantidad de datos, que ya se supone suficiente, se puede elaborar un perfil de carga. La intención es que el perfil aporte indicadores de carga a las horas en punto y a las medias tanto de los días de diario indistintamente como para los sábados y los domingos. Para ello, el perfil contendrá valores de carga promediados de los datos históricos para cada uno de esos tres casos. Es decir, se tomarían todos los valores de carga contenidos en el histórico de todos los días de diario a las 00:00 horas y se promediarían para generar el valor del perfil. A continuación se haría lo mismo pero con los valores de carga de las 00:30 horas. Se continuaría hasta llegar a los valores de las 23:30 horas. Una vez generado el perfil para los días de diario, el proceso sería análogo para los días sábado y domingo. Tras su obtención, el perfil de carga forma parte del modelo del recurso tratado, y se utiliza para predecir o estimar la carga del recurso en un intervalo concreto. La figura 7.5 muestra un ejemplo de un perfil así generado.

Es evidente que una vez puesto en producción un perfil de un recurso seguirán tomándose muestras para mejorar la precisión del perfil y para tenerlo actualizado en todo momento. Es una idea análoga a la de la actualización de los modelos del SOS. De esta manera, según se adquiere una nueva muestra el perfil podrá ser actualizado en ese preciso momento.

El modo en que el perfil se procesa también puede variar y, si su calculo mediante el promedio de las muestras puede resultar un tanto burdo, es posible realizarlo de cualquier otra manera, por ejemplo ponderando en mayor medida las muestras más recientes. Otra posible forma sería crear modelos a partir de redes de neuronas encargadas de aprender y generalizar el estado de los recursos utilizando para ello memorias a corto plazo pobladas con las muestras obtenidas, de manera análoga al funcionamiento del sistema de optimización de solicitudes, tal y como se recoge en la sección 5.3.1.

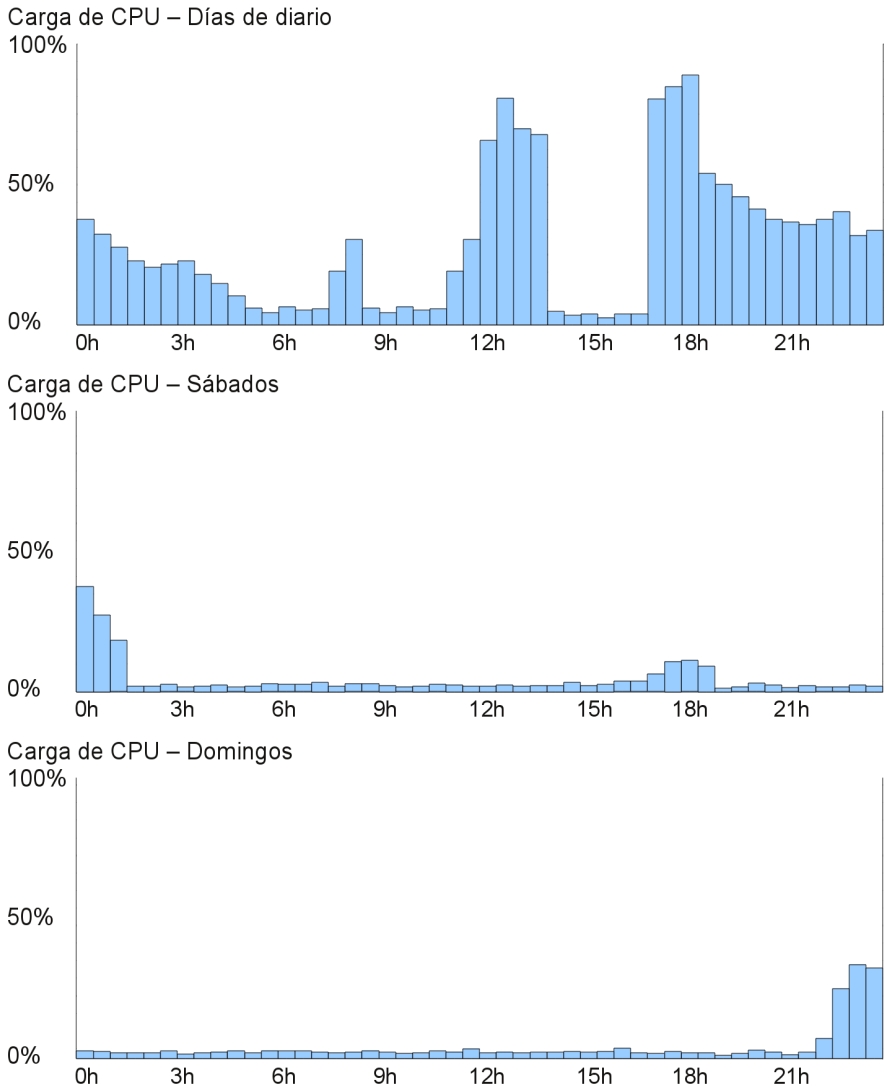

Figura 7.5: Ejemplo de perfil de carga de una CPU

El ejemplo presentado solo pretende aclarar de qué manera podría perfilarse un recurso para generar su modelo. Es fácil ver que no es necesario diferenciar entre días de diario y fines de semana si no hace falta o que podrían distinguirse todos los días de la semana de manera individual e, incluso, aumentar la frecuencia de muestreo para mayor precisión. Por otra parte, y de la misma manera, se puede generar un perfil del tiempo de actividad del recurso para conocer en qué intervalos se dispone de él o, en general, de cualquier otro parámetro necesario. De esta manera, la técnica de perfilado de recursos aquí expuesta permite modelar recursos de

manera muy flexible y abierta para la inclusión y prueba de diferentes tecnologías y métodos.

Asumido, entonces, este mecanismo de generación y actualización constante de perfiles, con un perfil de carga de CPU así construido, es trivial el realizar estimaciones del tiempo que tardaría la ejecución de un trabajo suponiendo que se conoce el tiempo que tardaría con la CPU a su completa disposición, tiempo que se conoce gracias a la propia estimación proporcionada por el usuario o por el SOS. Y es aquí donde reside la raíz del problema comentado y es que el SOS, tal y como está diseñado, hace la asunción de que los recursos de ejecución tienen las mismas características y potencia de cálculo, cosa que queda invalidada en un entorno *grid* o con recursos heterogéneos. Esto complica todavía más la solicitud de recursos a los usuarios, pues al no conocer *a priori* en qué máquina se van a ejecutar sus trabajos desconocen el tiempo de ejecución de los mismos, sumándose, además, a los problemas de las solicitudes explicados en la sección 5.1. Esto afecta directamente al funcionamiento del SOS, por lo que se plantean dos posibles alternativas:

1. Ya no se solicita al usuario una estimación del tiempo que durarán sus trabajos.
2. Se solicita al usuario una estimación de tiempo suponiendo que el trabajo se ejecutará en una máquina concreta de referencia.

Así pues, la aproximación propuesta a continuación, junto con la manera de generar perfiles de usuario, considera estas dos alternativas y dará solución al problema planteado, permitiendo la planificación de procesos en entornos heterogéneos manteniendo todos los beneficios obtenidos hasta ahora, sumándole, además, la optimización en la asignación de recursos.

Aunque para el segundo de los supuestos no es necesario rediseñar el SOS, sí lo es para el primero, pues ya no se cuenta con la solicitud de tiempo proporcionada por el usuario. Esto implica un cambio mínimo en el diseño actual pues basta con eliminar la entrada concreta relacionada con dicha solicitud. Las redes de neuronas utilizadas han de ser capaces de modelar el comportamiento del usuario y proporcionar estimaciones sin dicha entrada. No obstante, para ambas alternativas, el principal y común problema reside en las estimaciones de tiempo realizadas por los modelos. El tiempo de ejecución de un trabajo depende de la máquina concreta en la que se ejecuta por lo que no es de utilidad si se pretende estimar el tiempo de ejecución en otra máquina con diferentes características. Se requiere aquí la definición de una nueva medida de tiempo. Se asume $T_r(j)$ como el tiempo que tarda en ejecutarse el trabajo j en el recurso r. Esta es la medida utilizada hasta ahora en esta tesis. Se establece, por tanto, una nueva medida $T(j)$ que representa una medida de tiempo independiente del recurso en el que se ejecuta. Es posible que esta medida no tenga en sí demasiado sentido pero puede resultar intuitiva si se piensa en ella como una especie de tiempo genérico o el tiempo que tardaría la ejecución del trabajo j en una máquina de referencia, máquina que puede ser ficticia o real, tomándose en este último caso, por ejemplo un *cluster* homogéneo concreto perteneciente al *grid* con el que se trabaje. Es precisamente esa nueva medida la que puede proporcionar el usuario como solicitud de tiempo de ejecución en caso

de adoptar la alternativa de no rediseñar el SOS. Para terminar de darle entidad a esta nueva medida, denominada a partir de ahora *tiempo independiente* han de determinarse unos coeficientes de transformación o factores α que permitan traducir, por así decirlo, el tiempo independiente que tarda un trabajo j en el tiempo que tarda el trabajo en el recurso r y viceversa. De esta manera, teniendo un coeficiente de transformación para cada recurso r se puede obtener el tiempo que tarda un trabajo en dicho recurso a partir del tiempo independiente mediante la siguiente fórmula, donde α_r es el coeficiente de transformación para el recurso r:

$$T_r(j) = \alpha_r \cdot T(j) \tag{7.1}$$

Invirtiendo el coeficiente se puede obtener el tiempo independiente a partir del tiempo en un recurso dado:

$$T(j) = \alpha_r^{-1} \cdot T_r(j) \tag{7.2}$$

De esta manera, teniendo los coeficientes de transformación y al menos un tiempo de ejecución para el trabajo j, sea independiente o no, es posible calcular el tiempo que tardará el trabajo en cualquiera de los recursos del *grid*. No obstante, y como es evidente, el tiempo de ejecución de un trabajo no depende solo de las características y de la carga de la máquina y el estado de su sistema operativo, sino también de las características del propio trabajo como el uso que haga de disco o de memoria, hilos y procesos, tipo de comunicación entre procesos, cantidad de datos enviada entre procesos, el tipo de operaciones, el compilador utilizado, las opciones del compilador, etc. Es decir, no para todos los trabajos se cumple que el coeficiente de transformación se mantenga constante para una máquina dada. Por tanto, asumiendo que la proporción se mantenga en un porcentaje bastante alto, se busca una aproximación de los coeficientes que resulte una solución de compromiso para todos los trabajos y minimice el error de las transformaciones.

Los coeficientes α de transformación se pueden obtener empíricamente. La idea consiste en generar un conjunto A variado de aplicaciones y ejecutar cada uno de las aplicaciones del conjunto en cada uno de los recursos del *grid* con el que se trabaje (R). Cada una de las aplicaciones se ejecuta n veces en cada recurso y se promedia su tiempo de ejecución t con la máquina totalmente disponible para dichas ejecuciones. De esta manera se obtiene una matriz M_T de los tiempos medios de ejecución de cada trabajo en cada recurso con tantas filas como aplicaciones haya en el conjunto J y tantas columnas como recursos en R, donde cada uno de los elementos representa el tiempo promedio de lo que dura la ejecución de los trabajos en cada recurso. Se muestra dicha matriz a continuación donde $m = |J|$ y $n = |R|$ y cada elemento $t_{j,r}$ es el tiempo promedio que tarda la ejecución del trabajo j en el recurso r:

$$M_t = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,n} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{m,1} & t1_{m,2} & \cdots & t_{m,n} \end{pmatrix} \tag{7.3}$$

Nótese que podría suceder que algunos elementos de la matriz no tuvieran valor, pues no todas las aplicaciones son ejecutables en todas las máquinas pues pueden no contar con las características necesarias. Una vez obtenida la matriz M_T bastaría

con usar sus valores $t_{j,r}$ para calcular el valor del coeficiente de transformación α_r de cada recurso r. Pero antes de realizar el cálculo hay que determinar cuál será la escala de tiempo independiente que se va a utilizar. Como se dijo antes, lo más intuitivo es tomar los tiempos de uno de los recursos como tiempo independiente y trabajar con él. En un caso real podrían utilizarse los tiempos del recurso más importante del *grid*. Una vez determinado que el tiempo independiente coincide con los tiempos del recurso k se puede pasar a calcular los coeficientes mediante la siguiente fórmula, de donde resulta evidente que $\alpha_r = 1$ para $r = k$ y donde $m = |J|$:

$$\forall r : \alpha_r = \frac{\sum_{j=1}^{m} \frac{t_{j,r}}{t_{j,k}}}{m} \tag{7.4}$$

Como propuesta adicional, se pueden mejorar los coeficientes por medio de la utilización de trabajos reales, en lugar de un conjunto sintético de aplicaciones, aprovechando los trabajos enviados al sistema por los usuarios y por medio de su ejecución en varios recursos, y no solo en uno, para obtener las medidas deseadas, durante tiempos ociosos, por ejemplo. Aunque se puede hacer un sistema de estimación de tiempos en función del recurso mucho más complejo, no es el propósito de este trabajo. No obstante, las ideas aquí propuestas son suficientes para incluir en el sistema general de planificación un mecanismo de optimización de la asignación de recursos.

Tal y como está diseñado el SOS, cada vez que un trabajo termina se recogen sus datos de ejecución para actualizar los modelos de usuario asumiendo que los tiempos de ejecución son coherentes entre sí porque solo hay un tipo de recurso. Con el mecanismo presentado, basta con utilizar los coeficientes de transformación del recurso para calcular el tiempo de ejecución independiente y actualizar los modelos, que ahora trabajan con este tipo de tiempo. Así es posible actualizar los modelos de usuario con independencia del recurso en el que se ejecuten los trabajos evitando la necesidad de tener que generar un modelo de comportamiento por usuario y recurso con todos los problemas que ello conlleva, entre otros, el de no disponer suficientes datos de trabajos como para que todos los modelos de un usuario sean precisos y con capacidad de generalización.

Resumiendo, para poder optimizar la asignación de recursos es necesario, de cara a posibilitar el proceso de planificación concebido en este trabajo, poder estimar el tiempo de ejecución de los trabajos enviados por los usuarios en cualquier de los recursos aptos para su ejecución. Para ello han de generarse modelos de recursos que constan, por un lado, de tantos perfiles como parámetros se quieran modelar (carga de CPU, intervalos de tiempo en los que el recurso es utilizable, cantidad de memoria RAM, coste económico, etc) y, por otro lado, un coeficiente de transformación α que permita estimar el tiempo real de ejecución en el recurso a partir del tiempo independiente de ejecución del trabajo. Y para terminar de clarificar estos conceptos se expone el siguiente ejemplo:

1. Un usuario U ha enviado un trabajo j al sistema y el planificador necesita, para tomar sus decisiones, conocer el tiempo de ejecución de dicho trabajo

en un recurso r.

2. El SOS, mediante el modelo de usuario de U estima, basándose en la solicitud de ejecución de j que U ha hecho, que el tiempo independiente de ejecución será de 15 horas, es decir $T(j) = 15$.

3. Se calcula la estimación de tiempo $T_r(j)$ de ejecución de j en el recurso r por medio del uso del coeficiente de transformación α_r cuyo valor es de 1,2, con lo que resulta que $T_r(j) = \alpha_r \cdot T(j)$ es decir $T_r(j) = 1,2 \cdot 15$ cuyo resultado es 18 horas.

4. Como las estimaciones de tiempo de ejecución asumen que la CPU, y en general los recursos de la máquina, están totalmente disponibles para el trabajo y que no serán compartidos (excepto por los procesos prioritarios del sistema operativo), es necesario utilizar el perfil de carga de CPU que forma parte del modelo del recurso r para realizar la estimación coherente con el estado del recurso en el momento concreto que el planificador decida que debe ejecutarse.

5. Asumiendo que el perfil de carga de la máquina es similar al mostrado en la figura 7.5, que el planificador necesita conocer el momento en el que terminará el trabajo si comienza su ejecución el lunes a las 10:00 a.m. y que hacen falta 18 horas de CPU al 100 %, se realizará el cálculo que determine el intervalo de tiempo equivalente a esa capacidad de cómputo teniendo en cuenta el porcentaje de CPU libre en cada tramo del perfil. Es decir, se pretende transformar las horas de CPU del trabajo en horas reales (*walltime*). Integrando el perfil de carga entre las 10 a.m. y un momento desconocido X, igualando a 18 horas y resolviendo se obtiene que la diferencia de horas entre el momento X y las 10 a. m. del lunes será el tiempo real total de ejecución del trabajo, que para este ejemplo dado se estima en unas 24,5 horas, dando como resultado que el instante de finalización para el trabajo j será el martes a las 10:30 a. m.

7.4. Planificando para optimizar la asignación de recursos

Una vez resuelto el problema de la estimación del tiempo de ejecución de los trabajos en función de los recursos en los que serán ejecutados, el siguiente paso es proponer una aproximación para la asignación de los recursos. La propuesta aquí es mantener el esquema general de planificación que se ha venido usando y adaptarlo para permitir precisamente esto. Es deseable que el proceso de planificación tenga en cuenta también la asignación de los recursos para la consecución de los fines declarados en las políticas de planificación y no solo el orden de los trabajos supeditándose a un orden preestablecido de asignación de recursos. A menos que los administradores deseen establecer objetivos que tengan que ver con el uso de recursos, como maximización del tiempo de uso de algunos determinados o minimización de la energía utilizada, no existe necesidad de modificar las políticas ya

utilizadas, como la de maximización de satisfacción, por ejemplo, que ahora, presumiblemente y gracias a la consideración de los recursos a otro nivel, darán mejor resultado.

En la solución propuesta en la sección 4.3 se presentaba una codificación de los individuos para ser utilizada en diversos algoritmos de optimización, en especial, y los que más interesan en este trabajo, algoritmos evolutivos. Se pretende modificar esta codificación para incluir la representación de la asignación concreta de los recursos a cada uno de los trabajos en la cola de espera y no dejar que esta venga determinada por la posición del trabajo en la cola y el orden preestablecido de asignación. Para ello se mantiene la codificación utilizada para el orden de los trabajos, mediante un índice de posición en cada uno de los trabajos, y se añade para cada uno de ellos el recurso en el que será ejecutado. Por tanto, si para una cola con los trabajos j_1, j_2 y j_3 la representación consistía en $j_1(i_1), j_2(i_2), j_3(i_3)$, donde cada i representa el índice de posición para cada trabajo, ahora se añade un recurso r indicando el recurso concreto que le será asignado: $j_1(i_1, r_1), j_2(i_2, r_2), j_3(i_3, r_3)$. Así, por ejemplo, para los mismos trabajos A, B, C y D del apartado 4.3.2.1, con índices de posición 10, 40, 30 y 20, ahora se sumaría la asignación de los recursos x, y, z: $A(10, x), B(40, y), C(30, x)$ y $D(20, z)$, que resultaría en una planificación A en x, D en z, C en x, B en y. Esta manera de codificar permite mantener el tamaño del cromosoma constante y, además, tener en cuenta aquellas restricciones de recursos aplicables, pues solo serán asignados a los trabajos aquellos recursos capaces de satisfacerlos. Esta codificación está avalada por algunos trabajos e investigadores. Por poner algunos ejemplos se puede citar a Yeo Keun Kim *et al.* que en [133] desarrollan un modelo para planificación de trabajos en problemas de *job shop scheduling*, donde utilizan tal cual esta codificación para la asignación de recursos, o a Gkoutioudi *et al.*, que realizan en [57] un trabajo para la planificación de procesos en entornos *grid* mediante algoritmos genéticos.

Una consecuencia de esta codificación es que los índices de posición son aplicables ahora a los trabajos a los que ha sido asignado un recurso concreto, es decir, no importa el orden de trabajos con otros recursos asignados. Dicho de otra manera, el efecto ahora es el de manejar varias colas de trabajos, una por recurso. Por tanto, el objetivo es encontrar la mejor repartición de trabajos entre los recursos disponibles así como el orden de ejecución de cada una de las particiones.

Para comenzar con la evolución, el primer paso es generar la población inicial. Para generarla de manera aleatoria basta con obtener índices de posición aleatorios y, para cada trabajo, asignar al azar uno de los recursos en los que el trabajo en cuestión puede ser ejecutado. No obstante, podrían introducirse algunas reglas de generación para facilitar el trabajo del algoritmo evolutivo. A continuación, es necesario modificar el simulador que permitirá extraer los valores de *fitness* de los individuos para que tenga en cuenta la asignación de recursos. Respecto de los operadores de reproducción, el cruce puede continuar igual mientras que el de mutación varía la posición de un trabajo o cambia su recurso asignado.

7.4.1. Esquema general de planificación con optimización de la satisfacción de recursos

Al igual que se hizo con los sistemas de planificación evolutiva, de optimización de solicitudes y de estimación de la satisfacción, se presenta aquí el proceso general de planificación derivado del uso del sistema de optimización de la asignación de recursos, el SOAR.

La figura 7.6 refleja el ciclo de vida de la planificación en el que tienen lugar los eventos relacionados con la optimización de la asignación de recursos. En azul se presentan las acciones realizadas por el administrador, en verde las realizadas por los usuarios y en rojo las realizadas por los sistemas propuestos en esta tesis. En gris, nuevamente, se indica la acción de ejecución de trabajos que tiene lugar en los nodos de ejecución.

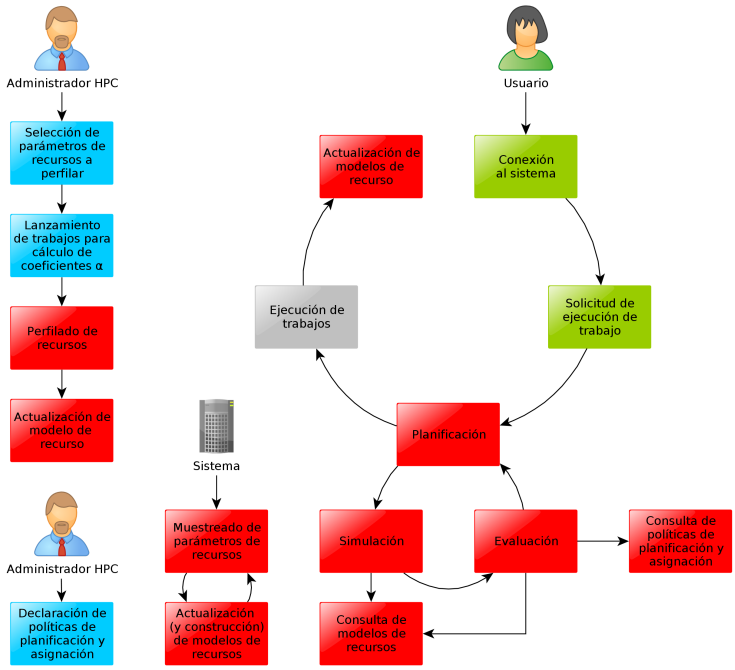

Figura 7.6: Eventos en el proceso de planificación para la optimización de la asignación de recursos

Como en los anteriores casos, una de las tareas del administrador es la declaración de políticas tal y como se vio en el capítulo 4. No obstante, ya no será su única tarea. Es responsabilidad del administrador decidir qué parámetros de los recursos serán modelados así como la estrategia a utilizar para ello. Para aquellos parámetros de los que dependa el tiempo de ejecución de los trabajos, como por ejemplo la potencia de cálculo, tal y como se vio en este capítulo, el administrador debe lanzar una batería de trabajos para obtener los coeficientes α que permiten

estimar las duraciones de los trabajos en los diversos recursos disponibles, como se indica en la sección 7.3.2, si no se había hecho previamente.

Con los parámetros a modelar seleccionados, el SOAR procede a muestrear los recursos para aquellos parámetros que varían con el tiempo. Según se obtienen la muestras se proporcionan a la tarea encargada de actualizar los modelos de recursos que también es la responsable de generar los nuevos modelos para aquellos recursos que todavía no tengan.

Con las políticas de planificación ya determinadas y el proceso de actualización continua de los modelos de recursos en marcha, los usuarios se conectan al sistema y realizan sus solicitudes de ejecución de trabajos. Durante la planificación evolutiva, en la etapa de simulación se utilizan los modelos de recursos para estimar la duración de los trabajos en función de aquellos parámetros de los que dependa, como la potencia de cálculo o la carga de CPU. Durante la etapa de evaluación se consultan las políticas de planificación así como, nuevamente, los modelos de recursos con el fin de comprobar si la planificación cumple con los requisitos de las políticas y satisface a los administradores. Una vez que la planificación termina, algunos trabajos son puestos en ejecución. Al finalizar los trabajos, se actualizan aquellos modelos de recursos que hayan ejecutado dichos trabajos.

Esta es la última de las propuestas de esta tesis, que contribuye a configurar un planificador global que considera muchos de los aspectos problemáticos de la planificación de trabajos. En lo que resta de capítulo se presentarán pruebas de funcionamiento del sistema SOAR y se terminará con un resumen final de resultados. En el capítulo siguiente se hace un resumen de todo el sistema construido a través de la integración de todas las propuestas presentadas que ayudará a clarificar su visión global, su utilidad y el nuevo proceso de planificación.

7.5. Pruebas y resultados

En esta sección se pretende mostrar la utilidad del modelado de recursos para lograr mejores planificaciones, para lo que se han realizado dos pruebas. En la primera de ellas se lleva a cabo la planificación de una serie de trabajos bajo dos supuestos. En el primer supuesto no se considera un modelado de recursos, realizando una planificación clásica que busca la minimización del *makespan* asumiendo que los recursos están siempre disponibles. Posteriormente se compara la planificación lograda con el resultado teórico de ejecutar dicha planificación, que difieren entre sí en gran medida. En el segundo supuesto se considera el modelado de dos parámetros distintos, que son la carga de CPU y la disponibilidad, y se realiza una planificación acorde a los modelos de recursos, pudiendo comprobar cómo la planificación realizada por el sistema resulta mejor, en términos de *makespan* que la ejecución teórica de la planificación sin modelado de recursos. En el segundo experimento se siguen estos mismos pasos pero alterando el perfil de carga de CPU en los recursos, donde, además de poder comprobar que la planificación con modelado resulta más ajustada, se puede ver como la planificación varía en función del perfil para, una vez más, lograr un resultado más fiel.

Como medida de comparación en las pruebas se utilizará el *makespan*. La primera prueba consiste en planificar una serie de trabajos de distintas características para un conjunto de recursos determinado. Las características de los trabajos se recogen en la tabla 7.1. Se utiliza un *grid* con dos recursos independientes. Cada uno de ellos tiene un perfil de funcionamiento diferente con distintas cargas de trabajo y horas en las que se encuentran fuera de servicio. Se utiliza un perfilado por día. El recurso etiquetado como "recurso 1" está fuera de servicio de 7 a 10 horas y de 14 a 18, mientras que de 0 a 7 y de 18 a 24 tiene una carga de CPU del 20 %. El resto de periodos en activo la carga de CPU es del 0 %. El segundo recurso, "recurso 2" se encuentra fuera de servicio de 10 a 12 y de 22 a 24, mientras que su carga de CPU es del 40 % de 12 a 16 horas. La imagen 7.7 muestra dos gráficas con estos perfiles, en las que se puede ver con un gráfico de barras la ocupación de la CPU mientras que una señal cuadrada muestra los periodos de servicio. Cuando dicha señal está arriba indica que el recurso está disponible, mientras que una señal baja muestra el estado de fuera de servicio.

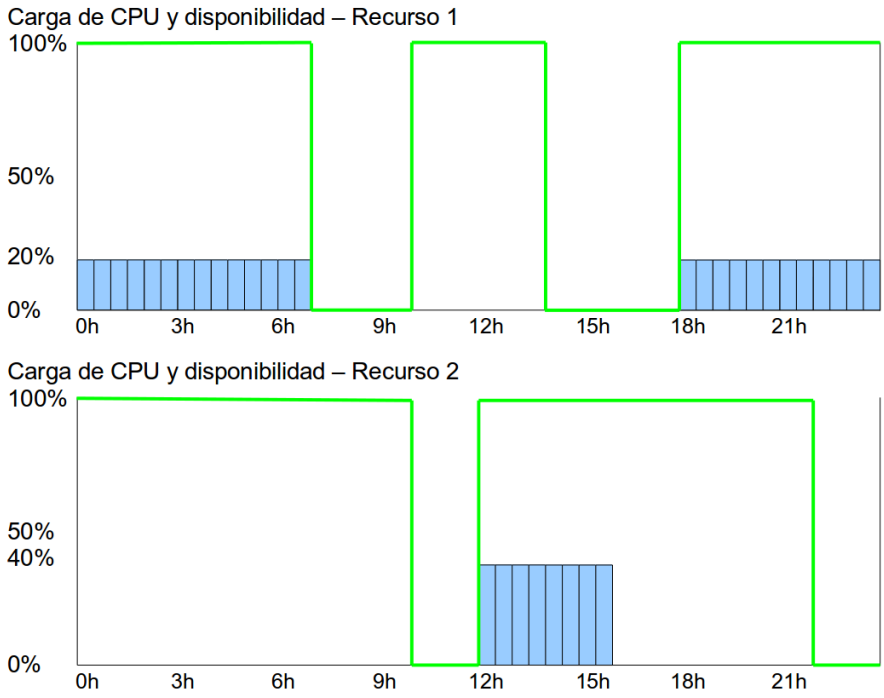

Figura 7.7: Perfil diario para un *grid* de dos recursos

La primera parte de la prueba consistió en planificar una cola con los trabajos indicados mediante el planificador evolutivo con el objetivo de minimizar el *makespan* sin considerar el modelado de recursos. De esta manera, el planificador asume que los recursos están totalmente disponibles. Las planificaciones para cada uno de los recursos están recogidas en la tablas 7.2 y 7.3. Dichas planificaciones quedan

Trabajo j	$T_s(j)$ **(h)**	**Llegada (h)**
1	1	0
2	1	0
3	1	0
4	1	0
5	1	0
6	1	0
7	1	0
8	1	0
9	2	0
10	2	0
11	2	0
12	2	0
13	4	0
14	4	0
15	6	0
16	6	0

Tabla 7.1: Características de los trabajos planificados en las pruebas de modelado de recursos

reflejada de manera visual en la figura 7.8. Se puede ver que el *makespan* logrado es de un total de 18 horas, coincidiendo en ambos recursos a la vez.

Trabajo j	**Usuario**	**Procesos**	$T_s(j)$	$T(j)$	**Inicio (h)**	**Fin (h)**
1	1	1	1	1,00	0,00	1,00
12	1	1	2	2,00	1,00	3,00
10	1	1	2	2,00	3,00	5,00
7	1	1	1	1,00	5,00	6,00
13	1	1	4	4,00	6,00	10,00
4	1	1	1	1,00	10,00	11,00
3	1	1	1	1,00	11,00	12,00
15	1	1	6	6,00	12,00	18,00

Tabla 7.2: Planificación sin modelado para el recurso 1

Aunque la planificación logra un *makespan* óptimo, el hecho de no considerar los recursos hace que la ejecución de los trabajos según dicha planificación resulte en un escenario bien diferente. Debido a la carga de CPU en determinadas horas los trabajos planificados tienen un tiempo de ejecución más largo del estimado, cosa que hace que los posteriores trabajos se retrasen. Por ejemplo, el trabajo 1, que se ejecuta el primero en el recurso 1 y cuya duración estaba estimada en 1 hora, debido a la carga de CPU dura 1,25 horas. Esto hace que el segundo trabajo planificado para ese recurso, el 12, entre más tarde lo planificado, encadenando dicho retraso

Trabajo j	Usuario	Procesos	$T_s(j)$	$T(j)$	Inicio (h)	Fin (h)
14	1	1	4	4,00	0,00	4,00
2	1	1	1	1,00	4,00	5,00
9	1	1	2	2,00	5,00	7,00
8	1	1	1	1,00	7,00	8,00
16	1	1	6	6,00	8,00	14,00
5	1	1	1	1,00	14,00	15,00
11	1	1	2	2,00	15,00	17,00
6	1	1	1	1,00	17,00	18,00

Tabla 7.3: Planificación sin modelado para el recurso 2

al resto de trabajos. Además, el trabajo 12 se encuentra en la misma situación que el primero, pues dura más de lo estimado generando un nuevo retraso. Por otro lado, los periodos en que el recurso está fuera de servicio causan un doble retraso. Primeramente, es evidente que durante el tiempo en que el recurso está inactivo no va a poder ejecutarse ningún trabajo. Adicionalmente, si un trabajo entra en ejecución antes de que el recurso pase a estar fuera de servicio y durante su ejecución se cambia a este estado, el trabajo es interrumpido. Sin ningún mecanismo de recuperación del trabajo, sería necesario volver a lanzar el trabajo en el momento en que el recurso volviera a estar disponible. Este es el caso de los trabajos 7 y 13, por ejemplo, que se ven interrumpidos y deben recomenzar, generando nuevos retrasos respecto de la planificación inicial. La figura 7.9 muestra cómo sería la ejecución de la planificación, que arroja un *makespan* de 49,5 horas en el recurso 1 y de 27 horas en el recurso 2, datos que distan mucho de los obtenidos en la planificación.

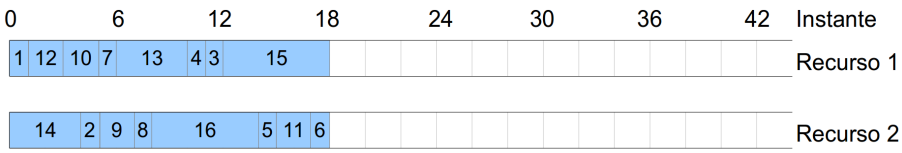

Figura 7.8: Planificación para un *grid* con dos recursos sin modelado

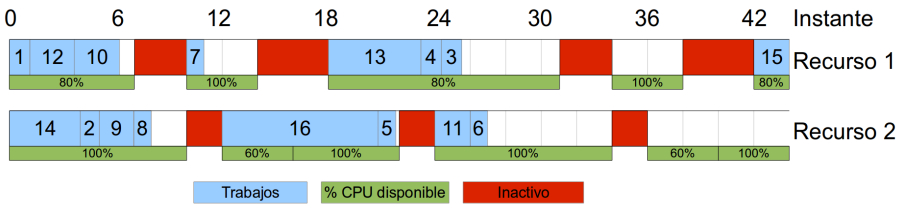

Figura 7.9: Ejecución según planificación sin modelado

La siguiente parte de la prueba consistió en repetir la planificación, pero esta vez teniendo en cuenta los perfiles de los recursos. La planificación obtenida está reflejada en las tablas 7.4 y 7.5 y en la figura 7.10. Se puede comprobar que, en este caso, el *makespan* arrojado es de 26 horas para el recurso 1 y de 26,75 horas para el recurso 2, que distan de ese ideal de 18, pero que proporcionan una planificación óptima que, evidentemente, mejora el resultado de 49,5 horas obtenido en la ejecución para la planificación sin modelado de recursos.

Trab. j	**Usuario**	**Procs.**	$T_s(j)$	$T(j)$	**Inicio (h)**	**Fin (h)**
5	1	1	1	1,00	0,00	1,00
15	1	1	6	6,00	1,00	7,00
6	1	1	1	1,00	7,00	8,00
9	1	1	2	2,00	8,00	10,00
12	1	1	2	3,33	12,00	15,33
2	1	1	1	1,67	15,33	17,00
4	1	1	1	1,00	17,00	18,00
14	1	1	4	4,00	18,00	22,00
10	1	1	2	2,00	24,00	26,00

Tabla 7.4: Planificación con modelado para el recurso 1

Trab. j	**Usuario**	**Procs.**	$T_s(j)$	$T(j)$	**Inicio (h)**	**Fin (h)**
3	1	1	1	1,25	0,00	1,25
13	1	1	4	5,00	1,25	6,25
11	1	1	2	2,00	10,00	12,00
7	1	1	1	1,00	12,00	13,00
1	1	1	1	1,00	13,00	14,00
16	1	1	6	7,50	18,00	25,50
8	1	1	1	1,25	25,50	26,75

Tabla 7.5: Planificación con modelado para el recurso 2

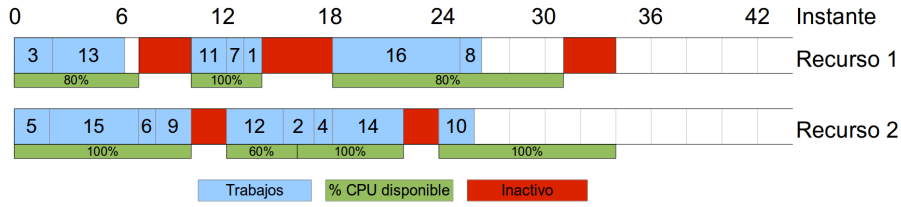

Figura 7.10: Planificación para un *grid* con dos recursos modelados

Con el fin de comprobar cómo afecta un cambio en el perfil de los recursos al planificar, se realizó una nueva planificación con los mismos trabajos de la tabla 7.1

y en los mismos recursos. En esta ocasión se varió el perfil de carga para que no resultase tan plano y sintético sustituyéndolo por uno que pueda asemejarse más a la realidad. La figura 7.11 muestra el nuevo perfil, donde se aprecia el cambio en las carga de CPU en aquellos periodos en que los recursos se encuentran disponibles. La disponibilidad permanece inalterada respecto de la prueba anterior. Así, en caso de ejecutar la planificación obtenida sin tener en cuenta el estado de los recursos, reflejada en las tablas 7.2 y 7.3, la nueva planificación resulta según la figura 7.12, produciendo un *makespan* de 51,03 horas para el recurso 1 (aunque no se vea en la figura, el trabajo 15 termina en el instante 51,03) y de 42,32 para el recurso 2.

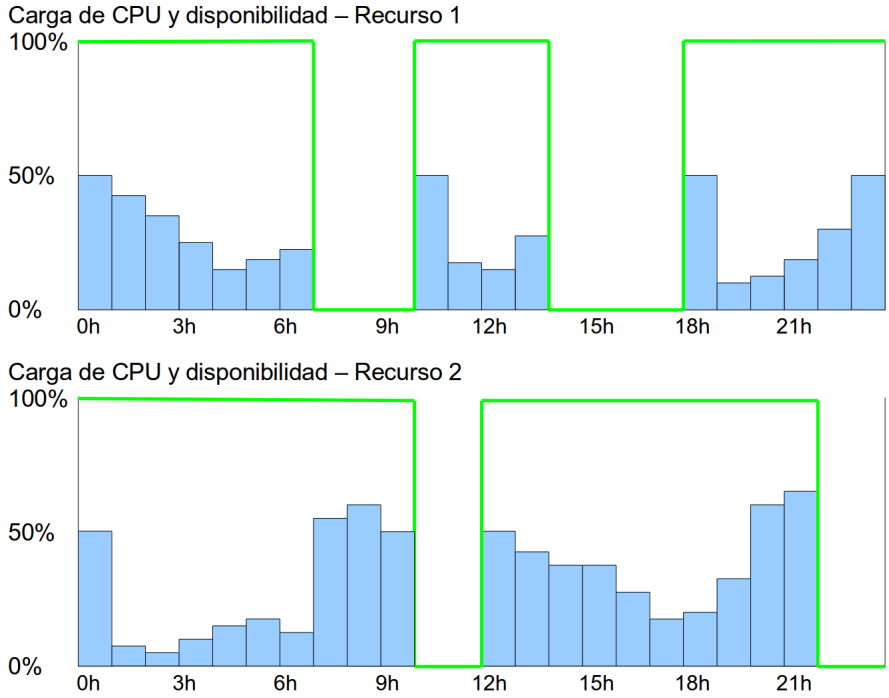

Figura 7.11: Perfil diario modificado para un *grid* de dos recursos

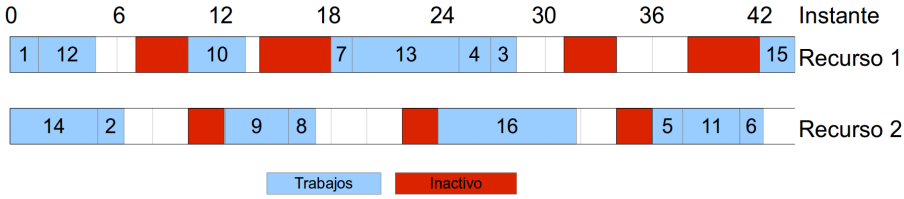

Figura 7.12: Ejecución según planificación sin modelado con perfiles modificados

Planificando con EvoProc, que sí tiene en cuenta el perfil mostrado, la planifica-

ción resultante es la reflejada en las tablas 7.6 y 7.7 y en la figura 7.13. El *makespan* logrado en este caso es de 39,58 y de 36,91 para los recursos 1 y 2 respectivamente. Se aprecia que, al utilizar máquinas con una carga mayor, los valores de *makespan* logrados son superiores a los de la prueba anterior, pues los trabajos ven incrementada su duración. Además, debido precisamente a la alta carga de CPU en algunas franjas horarias, se producen algunos huecos en la planificación en los que no se puede ejecutar un trabajo sin que se vea interrumpido. No obstante, se logra una buena planificación que resulta mejor que la ejecución teórica de la planificación que no consideraba el estado de los recursos, arrojando valores de *makespan* superiores y en la que los huecos existentes eran mayores.

En el siguiente capítulo se mostrarán pruebas en las que intervienen todos los sistemas propuestos en esta tesis donde, además de la disponibilidad y la carga de CPU, se modelará un parámetro de los recursos que no afecta a las duraciones de los trabajos pero que sí influye de manera directa en la satisfacción del centro de HPC: el coste económico de uso de los recursos.

Trabajo j	**Usuario**	**Procesos**	$T_s(j)$	$T(j)$	**Inicio (h)**	**Fin (h)**
16	1	1	6	7,89	0,00	7,89
11	1	1	2	3,58	12,00	15,58
4	1	1	1	1,48	15,58	17,06
12	1	1	2	2,66	17,06	19,72
13	1	1	4	5,00	24,00	29,00
2	1	1	1	1,29	29,00	30,29
3	1	1	1	1,59	30,29	31,89
10	1	1	2	3,58	36,00	39,58

Tabla 7.6: Planificación en recurso 1 con otro perfil de carga

Trabajo j	**Usuario**	**Procesos**	$T_s(j)$	$T(j)$	**Inicio (h)**	**Fin (h)**
14	1	1	4	5,88	0,00	5,88
6	1	1	1	1,69	10,00	11,69
7	1	1	1	1,23	11,69	12,91
15	1	1	6	9,03	18,00	27,03
8	1	1	1	1,33	27,03	28,35
9	1	1	2	2,51	28,35	30,87
1	1	1	1	1,69	34,00	35,69
5	1	1	1	1,23	35,69	36,91

Tabla 7.7: Planificación en recurso 2 con otro perfil de carga

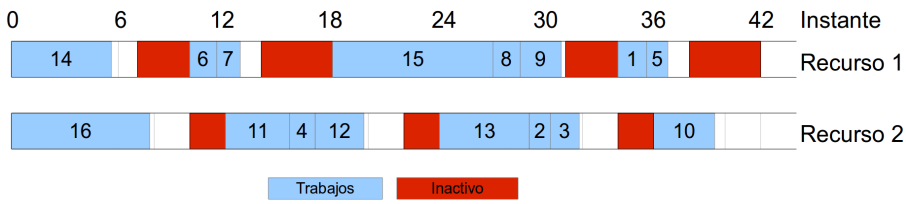

Figura 7.13: Planificación con modelado con perfiles modificados

7.6. Resumen y conclusiones

En entornos de recursos heterogéneos, como las tecnologías *grid*, se hacen mucho más presentes dos ideas principales. Primeramente, no se puede asumir que los trabajos utilicen los mismos recursos, en especial el tiempo de CPU, según la máquina o máquinas en las que se ejecuten debido a las diferencias entre los recursos. Por otro lado, la disponibilidad de los recursos deja de ser, idealmente, del 100 %, que al estar administrados por otros organismos o, incluso, pertenecer a particulares, pueden sufrir muchas variaciones, tanto de disponibilidad temporal como de carga de recursos disponibles (tiempo de CPU, RAM, etc.). Adicionalmente, gana mucha más relevancia la gestión de conceptos como el coste económico, la energía o la tasa de fallos. Algunas investigaciones, como por ejemplo [98, 99, 101, 102, 103, 104], han sido llevadas a cabo con estos problemas en mente. Los principales problemas detectados en estos trabajos pasan por la categorización y clasificación de los recursos para su modelado, predicciones a muy corto plazo o no considerar parámetros importantes para su modelado. Aquellos trabajos que integran de manera práctica el modelado de recursos en planificación lo hacen en meta–planificadores que no resuelven la fase final de planificación, que se delega al sistema final en el que se ejecutan los trabajos. Otros basan la planificación en sistemas de reservas de recursos o asumen la no existencia de trabajos paralelos. En general, ninguno de los trabajos sugiere un sistema de modelado abierto a perfilar múltiples parámetros entre los que se puedan incluir, además, aquellos relacionados con el coste económico o la energía, por ejemplo.

En este trabajo se planteó el desarrollo de un sistema de modelado de recursos para entornos heterogéneos y su integración en un planificador evolutivo como apoyo para lograr una mayor satisfacción entre los usuarios y en el centro de HPC. Este sistema, bautizado como SOAR, cumple las siguientes características:

- Permite el modelado individual de cada uno de las maquinas evitando el uso de clasificaciones o generalizaciones.
- Proporciona la flexibilidad de modelar diferentes parámetros en cada recurso: disponibilidad, carga de CPU o RAM, coste, energía, etc., de manera arbitraria según sea necesario.
- El diseño e implementación se realizaron para posibilitar el uso de cualquier técnica para la generación de los modelos y las predicciones.

- Proporciona predicciones del uso de recursos así como de diferentes parámetros para ser considerados por parte del planificador en sus simulaciones.
- Permite definir políticas de planificación que cumplan con los requisitos derivados de los intereses del propio centro, lo que implica un aumento en la satisfacción del mismo.

Para el desarrollo de este sistema, fue necesaria la modificación del planificador evolutivo, el SPE, presentado en el capítulo 4. Dicha modificación, junto con la integración del SOAR, permite resolver los problemas de configuración y definición y uso de políticas de planificación que ya existían en entornos homogéneos y que se ven magnificados en entornos heterogéneos.

Se realizaron diversos experimentos para comprobar la manera en que afecta el SOAR a la planificación y, sobre todo, para comparar el desvío de las planificaciones al tener en cuenta parámetros como la disponibilidad y la carga de CPU. La principal conclusión es que este sistema aporta una clara ventaja en entornos heterogéneos. Esto hace que las ventajas y la potencia de los diferentes sistemas presentados en esta tesis se puedan extender a dichos entornos logrando un sistema global más completo y utilizable en un mayor número de casos.

Capítulo 8

Integración y resumen del sistema de planificación

Aunque el final del camino parezca cercano
hay que caminar con la misma intensidad que
la aplicada en el primer paso.

El sistema de planificación propuesto y detallado en los capítulos anteriores incide, una vez integrado, en varios aspectos del proceso de planificación, que puede ser resumido en varios hilos, entendiendo hilo como un conjunto de acciones y eventos relacionados entre sí:

Establecimiento de políticas. Los administradores de centros HPC configuran políticas de planificación y asignación de recursos de manera declarativa.

Generación de modelos de recurso. Se toman muestras periódicas del estado de los recursos para generar sus modelos.

Operativa básica. Los usuarios envían trabajos al sistema.

Planificación. El sistema toma sus decisiones y selecciona qué trabajos ejecutar y en qué recursos.

Actualización de modelos de usuario y recursos. Se actualizan los modelos de usuario y de recursos del sistema con la información disponible tras la finalización de trabajos.

Generación y cumplimentación de encuestas de satisfacción Se adquiere información de satisfacción y se actualizan los modelos de satisfacción.

Estos hilos, que se han dispuesto en un orden más o menos lógico o natural, están compuestos de varios pasos o etapas que se exponen a continuación.

Hilo de establecimiento de políticas:

1. El administrador define un objetivo para las planificaciones, objetivo que será buscado por el planificador teniendo en cuenta tanto el orden de ejecución de los trabajos en espera como de la asignación de recursos. Entre las posibles políticas de planificación se encuentra la de la maximización de la satisfacción.
2. Adicionalmente, el administrador puede declarar una política de asignación de recursos que, al margen de lo buscado de manera general con la planificación, se cumpla un objetivo de segundo orden, como por ejemplo, la minimización de la energía utilizada o del coste económico.

Hilo de generación de modelos de recurso:

1. El administrador determina qué parámetros se quieren perfilar de cada recurso. Pueden ser la carga de CPU, la disponibilidad, la cantidad de memoria RAM disponible, el coste, etc.
2. De manera automática se muestrea el estado de cada recurso para generar los perfiles de recurso.
3. Se lanza el conjunto de trabajos destinado a medir la capacidad de los recursos para poder estimar los coeficientes de transformación α.
4. Se construyen los modelos de recurso que quedarán conformados por los perfiles del recurso más el coeficiente de transformación.

Hilo de operativa básica:

1. El usuario, conectado al sistema, envía un trabajo haciendo su pertinente solicitud.
2. El SOS, por medio del modelo de usuario correspondiente, si existe, optimiza la solicitud estimando la cantidad real de recursos de la que hará uso el trabajo enviado, entre ellos el tiempo independiente de ejecución.
3. El trabajo, junto con su solicitud optimizada, es puesto en la cola de espera.

Hilo de planificación:

1. Cuando sea necesario planificar (momento en el que quedan recursos libres) el sistema lee la cola de trabajos en espera junto con sus solicitudes optimizadas.
2. El sistema genera una población inicial de planificaciones (que puede partir de una población previa) y comienza el proceso de optimización.
3. Para evaluar la calidad de cada individuo de la población, el sistema ejecuta su simulador de planificación.

4. El simulador hace uso de los modelos de recursos (utilizando el coeficiente de transformación α y los perfiles del recurso) para poder calcular el tiempo real de duración de los trabajos durante la simulación del paso del tiempo y la ocupación y liberación de los recursos.

5. Una vez terminada la simulación del individuo se le asigna un valor de calidad midiéndolo en función de la política de planificación establecida por el administrador (considerando, si así lo dispuso el administrador, la satisfacción de los usuarios por medio de los modelos de satisfacción).

6. Tras el proceso de planificación se toma el mejor individuo hallado y el sistema pone en ejecución los trabajos seleccionados en los recursos asignados.

Hilo de actualización de modelos de usuario y recursos:

1. Cuando un trabajo termina se obtienen los valores reales de uso de recursos por parte del trabajo.

2. El sistema por medio del modelo del recurso correspondiente, calcula el tiempo de ejecución independiente.

3. El SOS actualiza, con los nuevos datos, el modelo del usuario propietario del trabajo recién terminado.

Hilo de generación y cumplimentación de encuestas de satisfacción:

1. Cuando un usuario se conecta al sistema, se le presenta una encuesta sencilla de satisfacción si ha terminado alguno de sus trabajos desde su última conexión.

2. El usuario responde la encuesta.

3. El sistema actualiza el modelo de satisfacción del usuario para ser utilizado durante el proceso de planificación.

Todos estos pasos conforman el proceso global de planificación, que de manera esquemática y visual se puede observar en la figura 8.1 desde el punto de vista de los eventos que transcurren en él y su secuencia lógica. Se presentan en azul las acciones realizadas por el administrador, en verde las realizadas por los usuarios y en rojo las realizadas por los sistemas propuestos. En gris se puede apreciar la acción de ejecución de trabajos, por parte de los nodos de ejecución.

Por otro lado, la figura 8.2 muestra el proceso desde la óptica del flujo e intercambio de datos e información entre las diversas partes del sistema. En ella se ven marcados en rojo los elementos de optimización propuestos en este trabajo, en verde los elementos pertenecientes al planificador en sí y el resto de elementos en azul.

Todo esto refleja la dinámica y operativa del sistema propuesto en este trabajo, en el que se ha desarrollado un planificador evolutivo que facilita la tarea de configuración permitiendo la definición de políticas dinámicas que tienen en cuenta

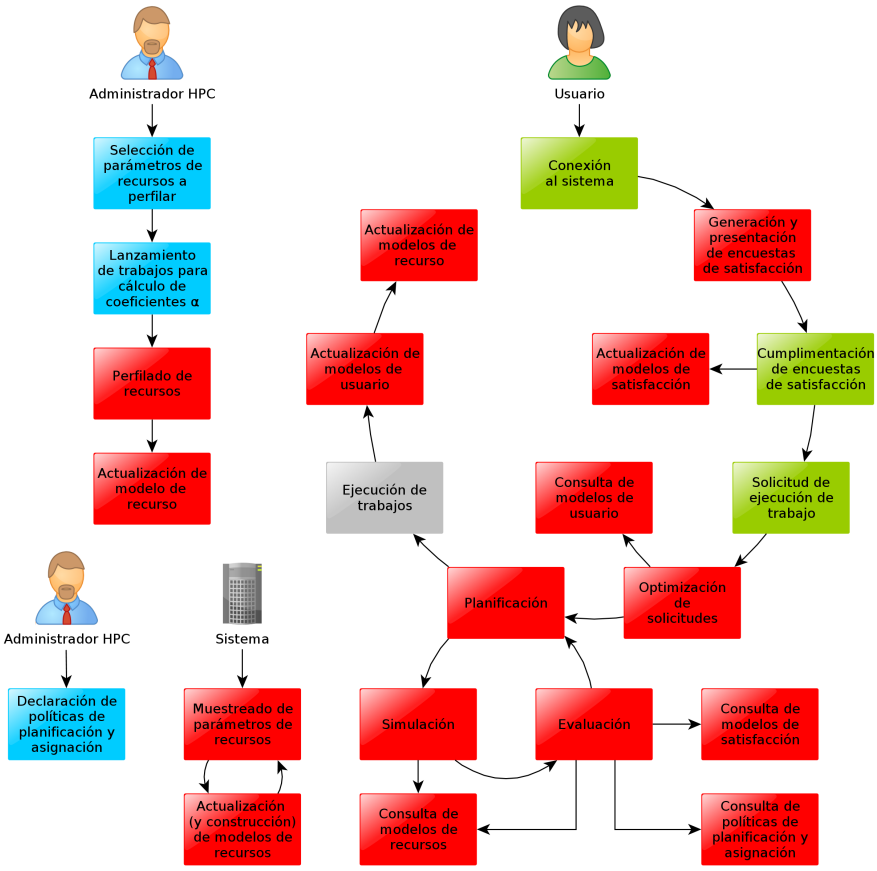

Figura 8.1: Eventos en el proceso global de planificación

estimaciones del uso de recursos, predicciones del estado de los propios recursos y de la satisfacción de los usuarios, a los que se adapta de manera particularizada.

8.1. Pruebas del sistema completo

El propósito de este apartado es mostrar el funcionamiento del sistema EvoProc completo, en el que se activan cada uno de los subsistemas presentados para lograr un resultado final y su comparación teórica frente a un planificador que no tiene en cuenta los diversos aspectos estudiados en este trabajo. Se realizan dos pruebas distintas. En la primera de ellas, el objetivo de la planificación es optimizar la satisfacción de los usuarios. En la segunda prueba se introduce un criterio de satisfacción del centro de HPC mediante el perfilado del coste económico de uso de recursos, que se optimizará junto con la satisfacción de los usuarios.

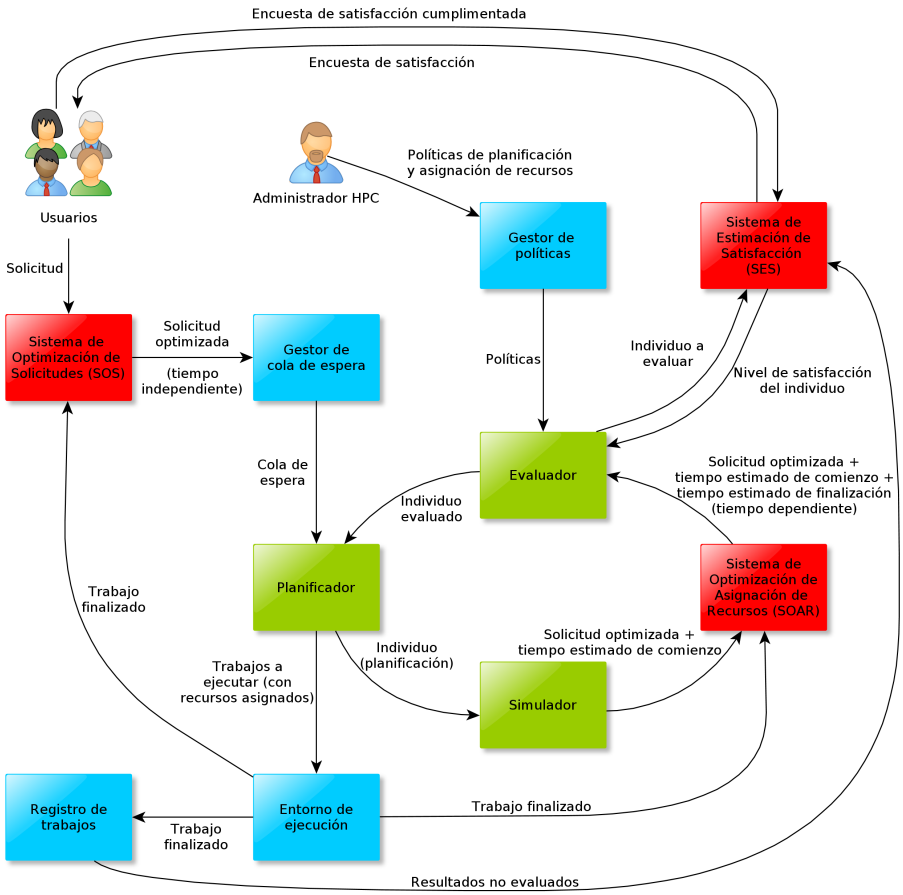

Figura 8.2: Proceso global de planificación, intercambio de información

8.1.1. Primera prueba: optimizando la satisfacción de los usuarios

El primer experimento comienza con el envío de trabajos a un *cluster* para su planificación por parte de tres usuarios diferentes, que serán "A", "B" y "C". Cada uno de estos usuarios ha sido modelado por el sistema de optimización de solicitudes, presentado en el capítulo 5. Con el fin de diseñar este experimento sintético se ha modelado a dichos usuarios para simular una duración de trabajos de un 90 % respecto del tiempo de CPU solicitado por los usuarios A y C, mientras que para el usuario B la duración de los trabajos es de un 85 % respecto de la solicitud. Los 27 trabajos enviados se recogen en la tabla 8.1, en la que se puede ver el número identificativo de cada trabajo, el usuario al que pertenece, el número de procesos del trabajo, el tiempo $T_s(j)$ de CPU solicitado y el instante del envío.

El sistema de HPC a utilizar es heterogéneo. Está compuesto de una máquina de

Trabajo j	**Usuario**	**Procesos**	$T_s(j)(h)$	**Envío**
1	A	1	1	0
2	B	1	1	0
3	C	1	1	0
4	A	2	1	0
5	B	2	1	0
6	C	2	1	0
7	A	1	1	0
8	B	1	1	0
9	C	1	1	0
10	A	2	1	0
11	B	2	1	0
12	C	2	1	0
13	A	1	1	0
14	B	1	1	0
15	C	2	1	0
16	A	1	2	0
17	B	1	2	0
18	C	1	2	0
19	A	1	2	0
20	B	1	2	0
21	C	1	2	0
22	A	1	4	0
23	B	1	4	0
24	C	1	4	0
25	A	1	6	0
26	B	1	6	0
27	C	1	6	0

Tabla 8.1: Características de los trabajos enviados para la prueba de integración final

dos procesadores y dos máquinas de un solo procesador. La máquina de dos procesadores, $R3$ en adelante, se supone dedicada y no se perfila. Las otras dos máquinas, $R1$ y $R2$ se perfilan según lo explicado en el capítulo 7 asumiendo los perfiles reflejados en la tabla 7.7. Además, los procesadores de estas máquinas cuentan con una potencia $1,2$ veces mayor que los procesadores del *cluster* homogéneo de dos procesadores, quedando determinado un coeficiente de transformación α de $0,8\dot{3}$ para el cálculo de la duración de los trabajos según lo presentado en la sección 7.3.1.

El planificador evolutivo se configura para maximizar la satisfacción de los usuarios (capítulo 6). Para el cálculo del nivel de satisfacción por trabajo se utilizan modelos de satisfacción muy similares a los generados a partir de los resultados sintéticos de encuestas recogidos en las tablas 6.2, 6.4 y 6.5.

Para contrastar la diferencia en los resultados del sistema frente a una planificación clásica en un sistema que no considera los aspectos tratados, se presentan a continuación tres resultados diferentes. Primeramente, se muestra la mencionada planificación clásica en la que se busca una minimización del *makespan* y solo se tienen en cuenta las solicitudes de los usuarios. Los resultados de la planificación se recogen en las tablas 8.2, 8.3 y 8.4 para cada recurso, $R1$, $R2$ y $R3$ respectivamente, y en las que se recoge el trabajo planificado, el usuario al que pertenece, el número de procesos del trabajo, el tiempo $T(j)$ de CPU solicitado, el tiempo $T_r(j)$ de CPU que el planificador tiene en cuenta para la realizar la planificación, los instantes en que el trabajo comenzaría a ejecutarse y terminaría y, como referencia, el nivel de satisfacción del usuario para ese trabajo.

Se puede comprobar de manera visual el resultado de la planificación en la figura 8.3. La satisfacción media para esta planificación sería de 2.431.

Trab. j	**Usuario**	**Procs.**	$T(j)$	$T_r(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
2	B	1	1	1,00	0,00	1,00	1,75
3	C	1	1	1,00	1,00	2,00	3,44
26	B	1	6	6,00	2,00	8,00	1,77
1	A	1	1	1,00	8,00	9,00	1,87
23	B	1	4	4,00	9,00	13,00	1,86
20	B	1	2	2,00	13,00	15,00	1,91
9	C	1	1	1,00	15,00	16,00	3,98

Tabla 8.2: Planificación clásica en el recurso $R1$

Trab. j	**Usuario**	**Procs.**	$T(j)$	$T_r(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
19	A	1	2	2,00	0,00	2,00	1,54
24	C	1	4	4,00	2,00	6,00	3,45
16	A	1	2	2,00	6,00	8,00	1,79
13	A	1	1	1,00	8,00	9,00	1,87
21	C	1	2	2,00	9,00	11,00	3,74
22	A	1	4	4,00	11,00	15,00	1,99
14	B	1	1	1,00	15,00	16,00	1,94

Tabla 8.3: Planificación clásica en el recurso $R2$

El problema con dicha planificación es que no se han tenido en cuenta diversos aspectos:

- El tiempo independiente de ejecución de cada trabajo, que difiere del tiempo de solicitud.
- Que algunos recursos tienen una potencia diferente, lo que hace que el tiempo de ejecución del trabajo varíe.

Trab. j	Usuario	Proc.	$T(j)$	$T_r(j)$	Inic. (h)	Fin (h)	$s(j)$
18	C	1	2	2,00	0,00	2,00	3,39
17	B	1	2	2,00	0,00	2,00	1,75
6	C	2	1	1,00	2,00	3,00	3,47
12	C	2	1	1,00	3,00	4,00	3,51
11	B	2	1	1,00	4,00	5,00	1,80
15	C	2	1	1,00	5,00	6,00	3,59
8	B	1	1	1,00	6,00	7,00	1,82
7	A	1	1	1,00	6,00	7,00	1,79
25	A	1	6	6,00	7,00	13,00	1,83
27	C	1	6	6,00	7,00	13,00	3,62
5	B	2	1	1,00	13,00	14,00	1,91
4	A	2	1	1,00	14,00	15,00	2,11
10	A	2	1	1,00	15,00	16,00	2,15

Tabla 8.4: Planificación clásica en el recurso $R3$

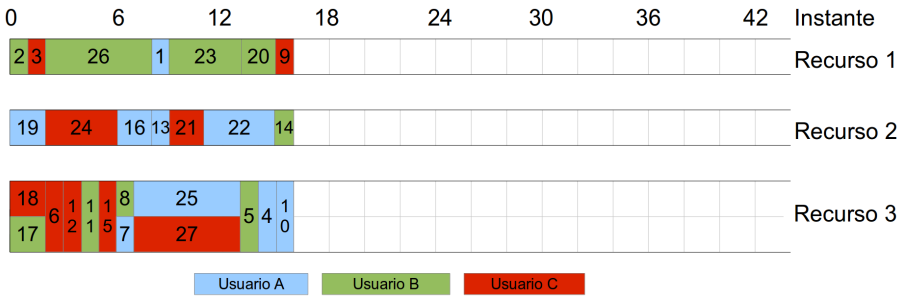

Figura 8.3: Planificación clásica sin modelado de recursos con política de optimización de *makespan*

- Que algunos recursos tienen momentos en los que no están disponibles y otros momentos en que tienen una carga de trabajo que, nuevamente, hace que los trabajos duren más de lo que deberían si tuvieran disponibilidad total del recurso.

- Que se está planificando con un criterio que puede no resultar el más adecuado.

Todo esto hace que, de ejecutarse esta planificación, el resultado en la realidad sería distinto de lo planificado. La ejecución teórica de la planificación queda reflejada en las tablas 8.5, 8.6 y 8.7 y en la figura 8.4. Dicha ejecución arrojaría un valor de satisfacción global de 2.480, superior al de la planificación original. Recuérdese que, según lo explicado en 6.3.4 la escala de satisfacción manejada en las encuestas es entre 1 y 7, siendo 1 el mejor valor de satisfacción y 7 el más bajo.

Trab. j	**Usuario**	**Procs.**	$T(j)$	$T_r(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
2	B	1	0,85	0,89	0,00	0,89	1,75
3	C	1	0,90	0,94	0,89	1,82	3,44
26	B	1	5,10	5,31	18,00	23,31	1,98
1	A	1	0,90	0,94	23,31	24,25	2,47
23	B	1	3,40	3,54	24,25	27,79	2,06
20	B	1	1,70	1,77	27,79	29,56	2,12
9	C	1	0,90	0,94	29,56	30,50	4,55

Tabla 8.5: Resultado teórico de ejecución en $R1$ de la planificación clásica

Trab. j	**Usuario**	**Procs.**	$T(j)$	$T_r(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
19	A	1	1,80	1,50	0,00	1,50	1,54
24	C	1	3,60	3,00	1,50	4,50	3,43
16	A	1	1,80	1,50	4,50	6,00	1,73
13	A	1	0,90	0,75	6,00	6,75	1,79
21	C	1	1,80	1,50	6,75	8,25	3,66
22	A	1	3,60	5,00	12,00	17,00	2,03
14	B	1	0,90	0,71	17,00	17,71	1,97

Tabla 8.6: Resultado teórico de ejecución en $R2$ de la planificación clásica

Trab. j	**Usuario**	**Procs.**	$T(j)$	$T_r(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
17	B	1	1,70	1,70	0,00	1,70	1,75
18	C	1	1,80	1,80	0,00	1,80	3,39
6	C	2	0,90	0,90	1,80	2,70	3,46
12	C	2	0,90	0,90	2,70	3,60	3,50
11	B	2	0,85	0,85	3,60	4,45	1,79
15	C	2	0,90	0,90	4,45	5,35	3,57
8	B	1	0,85	0,85	5,35	6,20	1,82
7	A	1	0,90	0,90	5,35	6,25	1,76
27	C	1	5,40	5,40	6,20	11,60	3,60
25	A	1	5,40	5,40	6,25	11,65	1,80
5	B	2	0,85	0,85	11,65	12,50	1,90
4	A	2	0,90	0,90	12,50	13,40	2,05
10	A	2	0,90	0,90	13,40	14,30	2,08

Tabla 8.7: Resultado teórico de ejecución en $R3$ de la planificación clásica

La planificación arrojada por EvoProc está recogida en las tablas 8.8, 8.9 y 8.10 y en la figura 8.5. Esta planificación proporciona un nivel de satisfacción global de 2,337, resultando el mejor valor de los tres aquí presentados.

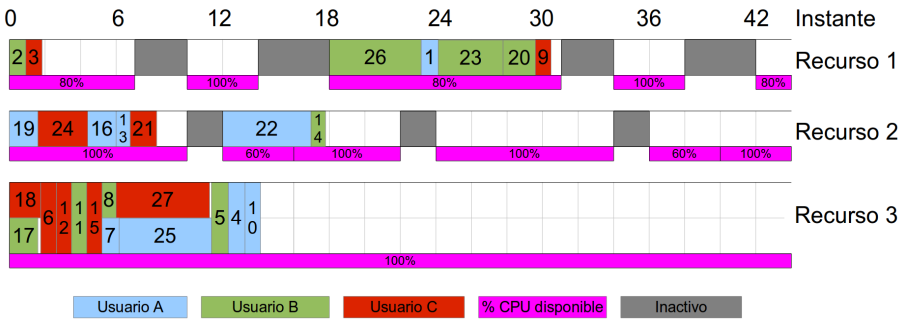

Figura 8.4: Resultado teórico de ejecución para la planificación clásica

Trab. j	**Usuario**	**Procs.**	$T_s(j)$	$T(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
7	A	1	0,90	0,94	0,00	0,94	1,54
3	C	1	0,90	0,94	0,94	1,88	3,44
16	A	1	1,80	1,88	1,88	3,75	1,62
18	C	1	1,80	1,88	3,75	5,63	3,54
2	B	1	0,85	0,89	5,63	6,51	1,82
20	B	1	1,70	1,42	10,00	11,42	1,87
17	B	1	1,70	1,42	11,42	12,83	1,89

Tabla 8.8: Planificación EvoProc en $R1$

Trab. j	**Usuario**	**Procs.**	$T_s(j)$	$T(j)$	**Inic. (h)**	**Fin (h)**	$s(j)$
13	A	1	0,90	0,75	0,00	0,75	1,54
1	A	1	0,90	0,75	0,75	1,50	1,57
9	C	1	0,90	0,75	1,50	2,25	3,46
19	A	1	1,80	1,50	2,25	3,75	1,63
21	C	1	1,80	1,50	3,75	5,25	3,54
14	B	1	0,85	0,71	5,25	5,96	1,81
8	B	1	0,85	0,71	5,96	6,67	1,82
24	C	1	3,60	3,00	6,67	9,67	3,63
26	B	1	5,10	7,08	12,00	19,08	1,90

Tabla 8.9: Planificación EvoProc en $R2$

8.1.2. Segunda prueba: optimizando la satisfacción de los usuarios y del centro

Tras comprobar el funcionamiento del sistema EvoProc completo en el caso del ejemplo anterior, en el que la política de planificación utilizada tenía como fin la optimización de la satisfacción de los usuarios, se configura un nuevo experimen-

Trab. j	Usuario	Procs.	$T_s(j)$	$T(j)$	Inic. (h)	Fin (h)	$s(j)$
10	A	2	0	0,90	0,00	0,90	1,54
4	A	2	0	0,90	0,90	1,80	1,58
12	C	2	0	0,90	1,80	2,70	3,46
15	C	2	0	0,90	2,70	3,60	3,50
6	C	2	0	0,90	3,60	4,50	3,53
11	B	2	0	0,85	4,50	5,35	1,80
5	B	2	0	0,85	5,35	6,20	1,82
22	A	1	3	3,60	6,20	9,80	1,79
25	A	1	5	5,40	6,20	11,60	1,80
27	C	1	5	5,40	9,80	15,20	3,73
23	B	1	3	3,40	11,60	15,00	1,89

Tabla 8.10: Planificación EvoProc en $R3$

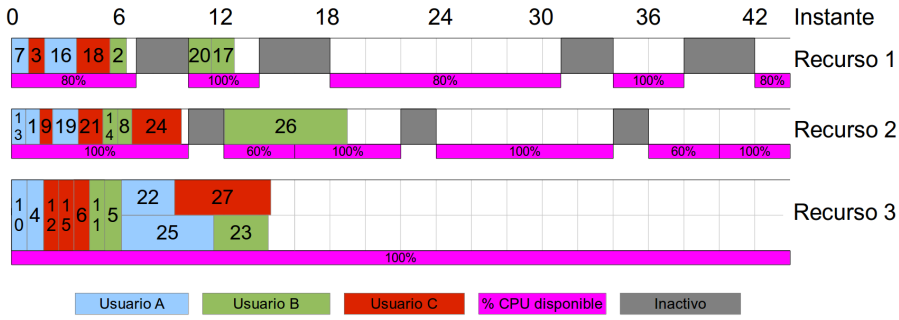

Figura 8.5: Planificación con el sistema EvoProc con política de maximización de la satisfacción

to que involucra en la política la satisfacción del centro de HPC. Para ello, se ha empezado por añadir un nuevo parámetro de perfilado de recursos directamente relacionado con el coste económico de uso de los recursos. Este coste puede venir dado por tasas de alquiler del uso de máquinas, costes energéticos y otros gastos indirectos, como por ejemplo, de mantenimiento.

El experimento parte del sistema configurado en el ejemplo anterior. Los recursos $R1$ y $R2$ están modelados, en lo que a carga de CPU y disponibilidad se refiere, según la figura 7.7. Además, se añade el nuevo parámetro de coste reflejado en la figura 8.6. En dicha figura, al igual que en el caso anterior, la línea verde indica la disponibilidad del recurso y las barras azules, la carga media de CPU por franja horaria. Las barras rojas representan el coste de uso de CPU, siendo el coste máximo de 0,10 € por franja horaria. Como ejemplo, si en una franja determinada el coste de uso es de 0,08 € y el trabajo hace uso de un 100 % de la CPU en esa franja, el coste para el centro es, justamente, de esa cantidad. Si el trabajo hace uso del 75 % de la CPU en esa franja, el coste para el centro sería de 0,06 €. Adicionalmente,

el recurso $R3$ tiene un coste constante de uso de 0,03 € por franja horaria. Los trabajos a planificar son los mismos que en la prueba anterior, recogidos en la tabla 8.1.

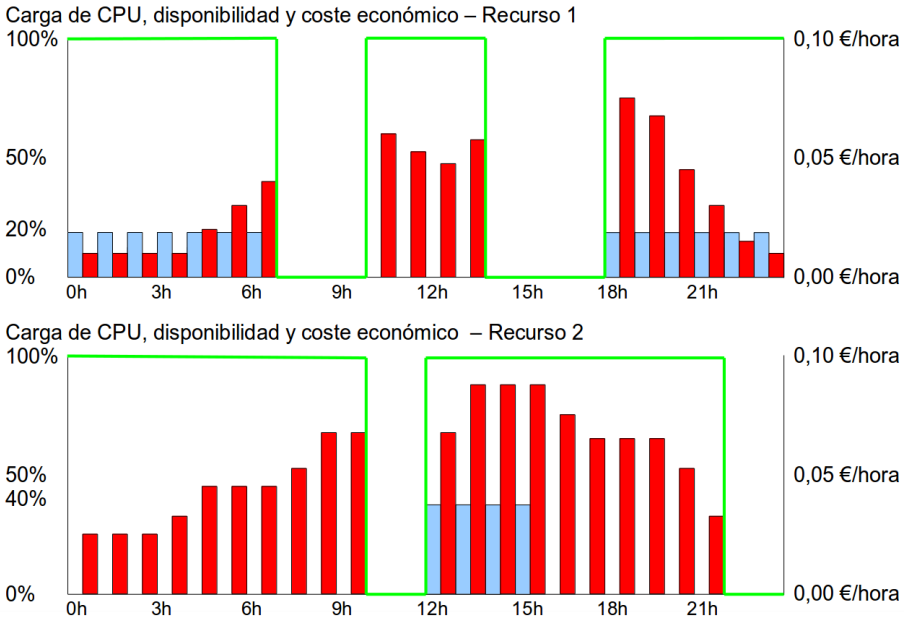

Figura 8.6: Perfil diario de tres parámetros para un *grid* de dos recursos

Con el fin de comparar el efecto en la planificación de incluir en la política los criterios de satisfacción del centro, que se traducen en la minimización del coste económico, y con la configuración indicada previamente, se han realizado tres planificaciones diferentes. La primera de ellas busca maximizar únicamente la satisfacción de los usuarios sin tener en cuenta el coste. La segunda planificación tiene en cuenta ambos criterios de satisfacción en la misma medida, es decir, busca un equilibrio entre la satisfacción de los usuarios y la del centro. Finalmente, la tercera planificación, deja de lado la satisfacción de los usuarios y busca únicamente la minimización del coste económico, es decir, la maximización de la satisfacción de centro.

La primera planificación, al obviar el coste económico de uso de los recursos, resulta exactamente igual que la planificación mostrada en la figura 8.5. Dicha planificación, como se indicó anteriormente, genera una satisfacción global de los usuarios de 2,337. El coste calculado para dicha planificación asciende a 1,739 € .

En la segunda planificación se busca un equilibrio entre la satisfacción de los usuarios y del centro. La planificación resultante es la mostrada en la figura 8.7, con la que se obtiene un satisfacción global de los usuarios de 2,393 y un coste de uso de 1,293 € . La diferencia con el primer caso es evidente. La política utilizada lleva a un menor uso del recurso $R2$, pues según el perfil presentado resulta el recurso

más caro. No obstante algunos trabajos permanecen en ese recurso, trabajos que se ejecutan en una franja en la que el coste no es demasiado elevado. Adicionalmente, se puede comprobar como la tendencia sigue siendo la del primer caso, en que se prima a los trabajos del usuario A frente a los del C y a los de este frente a los de B. Esto así debido a los perfiles de satisfacción, en los que el usuario A resulta el más impaciente y el B el más tolerante a las esperas. Es decir, se logra un cierto equilibrio, manteniendo parte de la satisfacción de los usuarios y logrando rebajar el coste.

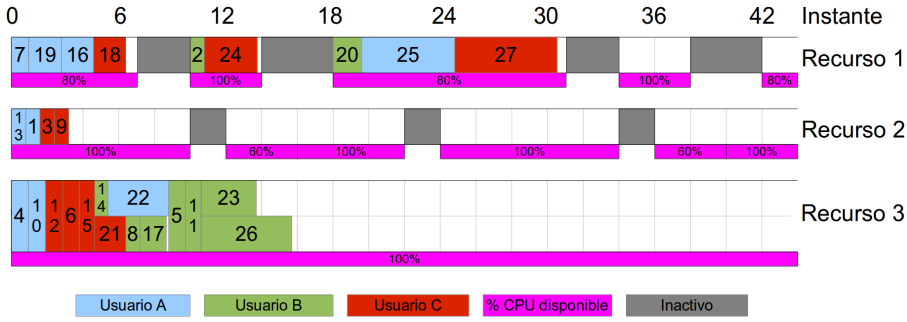

Figura 8.7: Planificación con el sistema EvoProc con política de maximización de la satisfacción de los usuarios y el centro

La política de la tercera planificación solo busca maximizar la satisfacción del centro, cosa que se logra minimizando el coste de uso de los recursos. La satisfacción de los usuarios no se tiene en cuenta. La figura 8.8 refleja la planificación lograda, que resulta en una satisfacción para los usuarios de 2,490 y un coste de uso de 1,285 €. En dicha figura se aprecia que el recurso $R2$ se utiliza todavía menos que en el caso anterior en favor de la utilización de franjas más económicas en otros recursos. Otro efecto interesante es que, al intentar minimizar el coste y no tener en cuenta la satisfacción de los usuarios, aparecen huecos en la planificación del recurso $R3$. Esto se debe al coste constante en dicho recurso, que hace que el orden en los trabajos no sea relevante, produciendo siempre el mismo coste sea cual sea la planificación.

La tabla 8.11 recoge los resultados de las tres planificaciones. En ella se aprecia que las satisfacciones logradas para los usuarios son más altas cuanto mayor importancia se le da a dicho criterio. Según se incorporan criterios de satisfacción del centro, como la minimización del coste económico, en este caso, la satisfacción de los usuarios se ve decrementada en favor de la del centro, logrando costes de uso menores.

Este ejemplo refleja uno de los casos más complejos que se pueden tratar con EvoProc, en el que el sistema realiza diversas operaciones para lograr un planificación final:

1. Se modela el comportamiento de los usuarios.

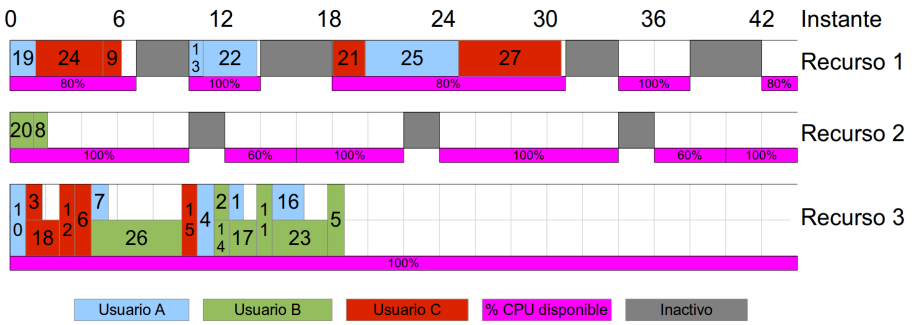

Figura 8.8: Planificación con el sistema EvoProc con política de maximización de la satisfacción del centro

Pol. usuarios	Pol. centro	Sat. usuarios	Coste
100 %	0 %	2,337	1,739
50 %	50 %	2,393	1,293
0 %	100 %	2,490	1,285

Tabla 8.11: Comparativa de políticas de planificación para la maximización de la satisfacción de los usuarios y del centro

2. Se estima el uso real de recursos gracias a los modelos de usuario, lo que permite disponer de información más precisa.

3. Se modela la satisfacción de los usuarios para poder tenerla en cuenta durante la fase de planificación.

4. Se dispone de recursos heterogéneos con diferentes potencias. Un *cluster* dedicado con un coste fijo de uso y dos recursos independientes.

5. Se modelan los recursos independientes según su disponibilidad, carga de CPU y coste de uso por franja horaria.

6. Se realiza una planificación con técnicas evolutivas. Esto permite la definición natural de la política de planificación.

7. Durante la etapa de simulación del proceso de planificación, se estima la duración de los trabajos en función de los perfiles de carga y disponibilidad de los recursos, así como de la potencia de las máquinas, mediante los factores α calculados.

8. Se declara una política que tienen en cuenta criterios subjetivos, tanto de los usuarios como del centro de HPC.

8.2. Resumen y conclusiones

En este capítulo se resume el sistema EvoProc completo formado a través de la integración de los cuatro subsistemas propuestos: el SPE, el SOS, el SES y el SOAR. Juntos resuelven los principales problemas indicados en esta tesis y proporcionan una solución completa para los mismos aplicable en el ámbito de un centro de HPC. En los diagramas presentados en las figuras 8.1 y 8.2 se puede ver el funcionamiento del sistema fraccionado en los distintos eventos relevantes dentro del ciclo de vida de la planificación, el intercambio de información y el funcionamiento y uso desde el punto de vista de los usuarios y los administradores del centro.

En las pruebas presentadas se pueden ver los resultados de ejemplos de planificación en el que cada uno de los sistemas lleva a cabo sus responsabilidades. Combinando los beneficios de cada uno de ellos se logra el objetivo final de obtener una planificación automática en la que se han conjugado varios aspectos:

1. Gracias al SPE, la tarea de definir una política resulta más sencilla que en un JMS clásico. En este caso, las políticas definidas tratan de maximizar la satisfacción de los usuarios y del centro. Para definir estas políticas basta con indicarle al sistema cuál es la función a maximizar. En las pruebas realizadas, se trabajó con tres políticas diferentes: la maximización de la satisfacción de los usuarios, la maximización de la satisfacción del centro y la combinación de ambas.

2. El sistema SES, que mantiene un modelo de satisfacción para cada uno de los usuarios (que se asumen ya creados), se encarga de proporcionar valores de satisfacción para una planificación dada. Esto es posible porque cada modelo de satisfacción estima la satisfacción que va a proporcionar al usuario los resultados concretos de cada uno de sus trabajos, en este caso, en lo que a tiempos de espera se refiere. Al mantener un modelo de satisfacción por cada usuario, el sistema los está considerando de manera individual. Este mecanismo de estimación es necesario para las políticas de planificación definidas por los administradores.

3. Antes de comenzar con el proceso de planificación evolutiva, se optimizan las solicitudes de recursos realizadas por los usuarios. Para ello, el SOS utiliza un modelo para cada uno de los usuarios con el que realiza las predicciones de recursos. Estos modelos son particulares de cada usuario con lo que el sistema, nuevamente, está teniendo en cuenta a los usuarios de manera individual.

4. El SOAR mantiene modelos de los recursos para poder calcular el tiempo de ejecución de cada uno de los trabajos. Además, estos modelos tienen en cuenta la disponibilidad de los recursos para evitar que el planificador envíe trabajos a nodos que van a dejar de estar disponibles impidiendo de esta manera su interrupción. Por otro lado, se modela un parámetro de coste económico asociado al uso de los recursos que permite la inclusión de políticas orientadas a la maximización de la satisfacción del centro, pues uno de los posibles intereses de este es la minimización de costes.

5. Durante el proceso de planificación, el SPE evalúa poblaciones de planificaciones teniendo en cuenta los modelos de recursos mantenidos por el SOAR y los modelos de satisfacción mantenidos por el SES para poder realizar tres estimaciones diferentes de recursos en cada una de las planificaciones de la población: el tiempo de ejecución de cada trabajo, la satisfacción de los usuarios y el coste económico. Una vez finalizado el proceso, el mejor individuo encontrado, según la política utilizada, pasa a ser la planificación real.

En los resultados de las pruebas se puede constatar la diferencia que hay entre la solución aportada por EvoProc y la planificación que produciría un planificador tradicional, en el que se utilizan los valores de solicitudes de recursos proporcionados por los usuarios y en el que los administradores asumen que optimizando el *makespan* lograrán la satisfacción de los mismos. Además, la propia planificación lleva a errores debido a no tener en cuenta la disponibilidad de los recursos así como su carga, haciendo que algunos de los trabajos se vean interrumpidos. Esto fuerza a relanzarlos posteriormente retrasando la ejecución de muchos de ellos. En definitiva, la diferencia es evidente y, a pesar de lo sintético del ejemplo, permite percibir claramente las ventajas de combinar los beneficios de cada uno de los sistemas propuestos en una única solución, el EvoProc.

Capítulo 9

Conclusiones

Siempre hay que acordarse del camino y hacer
balance al llegar a tu destino para poder
disfrutarlo en plenitud.

En este trabajo se ha estudiado un planificador de procesos para sistemas de computación de alto rendimiento que elimina, en gran parte, la necesidad de reconfiguración y actualización manual para adaptarse de manera individual a cada uno de los usuarios y evitando la generalización proveniente de un sistema de identificación de necesidades clásico. De esta manera, el sistema se ajusta dinámicamente a las necesidades cambiantes del conjunto de usuarios y del propio centro de computación. Para esto, y debido a la complejidad de estos sistemas, se ha atacado el problema en varios frentes.

Primeramente, se ha desarrollado un planificador evolutivo introduciendo la posibilidad de establecer políticas de planificación de manera más natural así como facilitar en gran medida la configuración del planificador hasta, incluso, evitarla por completo. En esta forma de establecer políticas ya no es necesario determinar la manera en la que la política debe actuar, sino que, simplemente, hay que proporcionar un valor de validez para las planificaciones. Al estar basado en técnicas evolutivas y gracias al uso de un simulador, dicho planificador es capaz de encontrar planificaciones válidas y mejoradas respecto de las técnicas y políticas tradicionales de planificación. El principal hallazgo tiene que ver con la comparativa de técnicas realizada en las pruebas, en la que el algoritmo micro–genético ha resultado el mejor, ofreciendo muy buenos resultados y mostrándose como un algoritmo prometedor para su utilización como base del planificador evolutivo propuesto.

Otro de los puntos principales tratado ha sido el de optimizar la información que los usuarios proporcionan al sistema al enviar un trabajo y hacer las solicitudes de recursos, que, como se ha explicado, o bien son inexistentes o no son muy precisas. Estos datos son los principales para que el planificador pueda hacer su trabajo. Para realizar esto, se ha desarrollado un sistema que, de manera autónoma y transpa-

rente al usuario, aprende y modela el comportamiento de los usuarios para poder llevar a cabo predicciones o estimaciones del uso real de recursos por parte de los trabajos. Para dicho desarrollo, y con el fin de dotarlo de las deseables características de autonomía y adaptabilidad, así como para lograr un aprendizaje gradual del comportamiento de los usuarios según nueva información va llegando al sistema, se desarrolló un algoritmo, el MCC, basado en técnicas bio–inspiradas como son los algoritmos evolutivos y las redes de neuronas artificiales. El algoritmo desarrollado toma como fuente de información un histórico reciente de las últimas operaciones de los usuarios en el sistema, que son modelados de manera individual, alejando a esta propuesta de otras soluciones basadas en clases o categorías de usuarios. Este modelado es ajustable mediante dos parámetros principales que son el tamaño del citado histórico y el número de generaciones entre actualizaciones del modelo. En las pruebas realizadas se ha encontrado que, para comportamientos cambiantes se prefieren tamaños de histórico mayores y bajar el número de generaciones, mientras que para comportamientos más estables lo ideal es la utilización de históricos de menor tamaño y un número de generaciones mayor.

En lugar de hacer asunciones sobre la manera idónea de mejorar un sistema, se ha propuesto que el propio sistema se adapte a sí mismo para mantener un alto nivel de satisfacción en general, tanto del centro como de los usuarios. Con esto, el sistema debe huir de generalizaciones y tener en cuenta de manera individual a cada usuario. Para ello, se ha desarrollado un sistema que aprende de las percepciones que los usuarios tienen del servicio para poder lograr, al ser integrado con el planificador desarrollado, planificaciones que alcancen niveles de satisfacción más altos. Este sistema se ha propuesto y desarrollado para no tener que llevar a cabo ningún análisis de causas o motivos de la satisfacción individual de cada usuario, evitando así tener que trabajar con una casuística, probablemente, muy compleja. De esta manera, el sistema aprende del nivel de satisfacción del usuario ante el servicio proporcionado y lo tiene en cuenta para realizar modelos que permiten obtener predicciones de satisfacción útiles durante el proceso de planificación.

Finalmente, se ha propuesto una manera de modelar o perfilar los recursos y sus distintos parámetros individualmente y de forma autónoma y dinámica para aquellos entornos heterogéneos en los que cada recurso tiene características y comportamientos diferentes. Esto extiende el ámbito de aplicación del sistema completo propuesto en esta tesis a este tipo de entornos, por ejemplo, entornos *grid*, haciéndolo utilizable en más casos. Con esto es posible realizar planificaciones más realistas teniendo en cuenta en todo momento información actualizada acerca de los recursos sobre los que planificar. Además, esta propuesta permite a los administradores definir, de manera natural, políticas de planificación relacionadas con el uso de recursos que inciden directamente en el nivel del satisfacción del centro de HPC. Una vez más, integrando este desarrollo en el planificador se logra, al componerse de estos cuatro pilares fundamentales, un sistema de planificación completo: el EvoProc.

Tras cada uno de los desarrollos individuales, así como de la integración de todos ellos, se han realizado diversos experimentos que muestran no solo su viabilidad, sino que demuestran que es posible la consecución de un sistema de planificación más fácil de mantener en el que poder instalar políticas de planificación que tienen

en consideración a cada uno de los usuarios por separado, los intereses generales del centro y de los administradores de los recursos, así como las diferentes características de estos.

Estos desarrollos muestran que la mejora y optimización de un sistema en los términos tratados en la introducción a este trabajo es posible. Es decir:

- Se puede huir de las generalizaciones en la forma de uso de un sistema por parte de los usuarios y hacer que este se adapte de manera personalizada a cada uno de ellos. Esto se ha logrado por medio del modelado de comportamientos y de la satisfacción.
- Es posible considerar otras medidas más allá de las clásicas o aquellas supuestamente más válidas universalmente o simplemente correctas para mejorar un resultado, al tener en cuenta en el proceso de planificación medidas o criterios subjetivos tales como la satisfacción de los usuarios o del centro de HPC.
- No es necesario descartar un sistema porque pueda parecer que se ha quedado obsoleto. En este caso, el propio sistema se adapta dinámicamente a los usuarios, tanto a su comportamiento, como a lo que esperan de él, cosa que se ha conseguido por medio de la introducción de técnicas que permiten dicha adaptación de manera autónoma. Además, esto permite utilizar el esfuerzo y el dinero en adquirir nuevos recursos teniendo la certeza de que el sistema principal se está adaptando a la naturaleza cambiante de su entorno para dar lo mejor de sí.

Capítulo 10

Trabajo futuro

Los sueños siempre están ahí. Tiene que haber
nuevos caminos esperando...

Se recogen en esta sección las actuaciones futuras más importantes que permitirían completar el sistema EvoProc bajo los preceptos extraídos en la introducción de este trabajo. Se comentan a continuación las posibles líneas futuras en cada uno de los cuatro pilares principales para terminar con aquellas que atañen al sistema completo.

10.1. Modelando el comportamiento de los usuarios

Uno de los principales planteamientos futuros es el de modelar el comportamiento de los usuarios en lo que a las conexiones al sistema se refiere. ¿Cuándo se conecta el usuario? ¿Con qué frecuencia? ¿Para qué fin? ¿Desde dónde lo hace? De todas estas preguntas se puede extraer información sumamente interesante con diversos fines. Sería deseable para mejorar todavía más las planificaciones el modelar este tipo de comportamientos de los usuarios, en concreto, el propio envío de trabajos. Si se pueden predecir que próximamente llegarán nuevos trabajos al sistema y estimar el uso de recursos de esos trabajos, se tendría una valiosa información para su consideración en la etapa de planificación. Este aspecto se comenta en la sección 7.1.

De cara a mejorar las predicciones del uso de recursos, se propone introducir en el sistema SOS aquí descrito la clasificación de trabajos tal y como se sugiere en desarrollos como el de Sonmez *et al.* [15], en el que se tienen en cuenta variables como el nombre del trabajo y el tamaño del ejecutable. Es presumible que esta clasificación permitirá realizar un modelado por usuario y por trabajo mejorando la precisión en las estimaciones.

10.2. Planificando

Muchas de las técnicas aquí utilizadas son iterativas y, en general, de cierto consumo de tiempo, lo que las hace buenas candidatas para ser en sí mismas trabajos susceptibles de ser ejecutados en sistemas de alto rendimiento. Gracias al propio planificador se pueden prever tiempos muertos en los recursos para que el sistema EvoProc realice los cálculos necesarios para mantener sus modelos actualizados o incluso para realizar planificaciones con antelación. Se plantea por tanto un mecanismo que lo permita, cosa que aliviaría la necesidad de tener una máquina especialmente dedicada a estas tareas.

En 2011, Alejandro Lucero del Centro Nacional de Supercomputación de Barcelona, presenta por primera vez en [134] el sistema SLURM (*Simple Linux Utility for Resource Manager*). Bajo una licencia *open source*, SLURM es un planificador de colas de trabajos para *clusters* de supercomputación que posee un mecanismo que permite la prueba de nuevas políticas de planificación ya que simula la ejecución de los trabajos, evitando la necesidad del uso de un *cluster* de producción. A pesar de que en este trabajo ya se ha desarrollado un simulador propio, es interesante evaluar la posibilidad de utilizar SLURM durante la fase de planificación en EvoProc, pues está siendo utilizando de manera exitosa en múltiples sistemas, entre ellos, algunos de la propia lista TOP 500.

Como se indicó en la sección 4.1.4, un centro de HPC puede perseguir diversos objetivos. En las políticas de planificación se puede requerir que se cumplan varios de estos objetivos simultáneamente, como por ejemplo, satisfacer globalmente a los usuarios a la vez que se minimiza el consumo energético y procurando no exceder un coste de uso de recursos concreto. Se plantea el uso de algoritmos evolutivos multi–objetivo [135] para poder atender a diversos criterios en las políticas evitando la necesidad de tener que combinar múltiples objetivos en uno.

10.3. Modelando la satisfacción

Hilando con la sección 10.1, y como se comentó en el capítulo 6, el tener conocimiento acerca de cuándo se conecta un usuario y para qué permitiría extraer, al menos en parte, de manera transparente al usuario y sin necesidad de encuestas, su nivel de satisfacción. Por poner un ejemplo, si un usuario se está conectado al sistema para consultar el estado de sus trabajos y lo hace un sábado a las once de la noche y desde su teléfono móvil, podría considerarse que el usuario tiene un cierto nivel de impaciencia con repercusión directa en su nivel de satisfacción. Es de relevancia este aspecto porque es el único punto en este trabajo en el que falta la transparencia lograda en el resto de subsistemas, donde el usuario accede y utiliza el sistema como caja negra sin ser consciente de su funcionamiento y sin modificar su comportamiento ni invertir un esfuerzo extra, como es el caso de la cumplimentación de las encuestas de satisfacción.

El concepto de satisfacción para el centro de HPC cambia respecto de los usuarios, pues el fin perseguido es distinto. De manera análoga a cómo se hace con estos,

podría plantearse un sistema en el que los administradores del centro evalúan su satisfacción respecto del comportamiento del planificador al final de una quincena o de un mes al estudiar el informe de resultados. La satisfacción en este caso puede venir determinada por varios aspectos:

- Los beneficios obtenidos, que pueden ser considerados un indicador bastante directo de la satisfacción.
- El consumo eléctrico, con incidencia directa en el coste económico o, incluso, en la mentalidad ecológica de la organización.
- Los gastos de mantenimiento.
- La cantidad de quejas o alabanzas recibidas por los usuarios de los sistemas.
- Las averías tenidas lugar en el periodo, algunas de las cuales pueden ser debidas al uso o desuso de determinados recursos.

Además de los usuarios y del centro, hay una tercera entidad directamente involucrada y digna de ser integrada en la ecuación de la satisfacción. Son los dueños de equipos que aportan parte de su tiempo o potencia a programas u organizaciones de computación voluntaria. Ya sea altruista o interesadamente, dichos propietarios también tienen una percepción del servicio al que están contribuyendo y que, como es lógico, afecta a la futura prestación.

La inclusión del concepto de las expectativas permitiría modular las predicciones de satisfacción. Se parte de la idea de que un pequeño incremento de satisfacción, en un usuario acostumbrado a un cierto nivel, puede derivar en una percepción mayor que la dictada por dicho incremento, y al revés. Por ejemplo, en una escala de 0 a 1, si a un usuario que suele estar un 0,5 satisfecho se le proporciona un servicio estimado como un 0.8 de satisfacción, se le causará una percepción de un 0.9 o un 1. Teniendo en cuenta que mejorar el nivel de satisfacción de un usuario particular implica decrementar, de manera general, la del resto, podría usarse este concepto para aplicar pequeños incrementos de satisfacción a los usuarios menos satisfechos y mantener un buen nivel de satisfacción general.

Finalmente, y como ya se adelantó en la sección 6.3.4, se planea la utilización de las mismas técnicas utilizadas en el SOS, es decir, el algoritmo MCC, para el modelado de la satisfacción, lo que implicaría un esquema de actuación muy similar para la creación de modelos de satisfacción y la predicción de la misma.

10.4. Modelando los recursos

En cuanto al modelado de recursos, cabe destacar que se pueden perfilar diversos parámetros para formar parte de las políticas de planificación. Muy en auge en estos días se encuentra la denominada *green computing* o *tecnologías verdes*. El término hace referencia al uso eficiente de los recursos para minimizar el impacto

medioambiental. No hay que olvidar que las máquinas con un alto número de procesadores tienen un consumo muy elevado, con lo que disminuirlo ha de ser, por varios motivos, una prioridad. Existen trabajos con esta idea en mente, de los que poder partir, como por ejemplo el presentado en [136] por Younge *et al.* o en [137] por Singh *et al*.

En [138], Foster *et al.* proporcionan una solución al problema que se encuentran los usuarios de un *grid* al intentar ejecutar aplicaciones no adecuadas a los recursos disponibles. En su trabajo proponen una forma de permitir a aquellos usuarios autorizados a crear *clusters* virtuales que se adapten a sus requisitos de *software*. Partiendo de esto, otra idea que resulta interesante es la posibilidad de poder utilizar el planificador para dimensionar un *cluster*, virtual o no, minimizando la inversión. Esto también resulta especialmente interesante, al margen de los entornos *grid*, en servidores de recursos virtuales como el EC2 (*Elastic Compute Cloud*) de Amazon [139] o el Azure de Microsoft [140].

10.5. Integración

A pesar de haberse hecho una integración temprana de EvoProc en el planificador SGE para poder probarlo en producción, sería muy interesante hacer una nueva integración, ya del sistema completo, en algún planificador usado ampliamente por muchos de los sistemas de colas utilizados en la actualidad, como es Maui [4] o TORQUE [141]. Es evidente que esto facilitaría en gran medida la integración con todos los sistemas de colas que soporten dichos planificadores.

Aunque los sistemas planteados en esta tesis colaboran entre sí para lograr un fin común, no se utilizan los resultados de unos para ajustar los otros. En concreto, los sistemas dedicados a modelar el comportamiento de los usuarios, el SOS, y a modelar la satisfacción, el SES, pueden intercambiarse información con el fin de mejorar ambos modelados. Goetlieb *et al*. estudian en [142] las relaciones existentes entre los conceptos de satisfacción, calidad percibida, expectativas y comportamiento, proponiendo un marco teórico para su modelado. Dicho trabajo resulta un buen punto de partida para mejorar la colaboración entre el SOS y el SES.

Apéndice A

Diseño *software* del EvoProc

No se ha querido dejar de lado durante las etapas de desarrollo de *software* del sistema de planificación propuesto en esta tesis el seguimiento de una metodología de desarrollo. Se ha utilizado una metodología iterativa e incremental, basada en el Proceso Unificado [143], que permitió alcanzar de manera gradual los objetivos planteados minimizando los riesgos y evitando, en la medida de lo posible, la modificación del código ya creado. El principal motivo para esto ha sido la complejidad del sistema EvoProc, en la que sus diversos subsistemas, SPE, SOS, SES y SOAR, colaboran entre sí para lograr planificaciones acordes a los objetivos propuestos en esta tesis. Esto requirió la aplicación de buenas prácticas que favorecieran un diseño directo y limpio pero flexible. Y es precisamente la flexibilidad uno de los grandes objetivos perseguidos en el diseño de los citados subsistemas, pues la necesidad de probar distintas tecnologías en la etapa de pruebas así lo requería. Además, el diseño se deja abierto a poder incorporar otras nuevas así como para facilitar el cambio de diversas estrategias en los principales algoritmos desarrollados. Como ejemplo de dichos cambios se pueden destacar las técnicas de optimización utilizadas en el algoritmo de planificación, en el algoritmo MCC para el modelado del comportamiento de los usuarios, en el algoritmo de modelado de la satisfacción, en el de modelado de los recursos, las estrategias de muestreo de los recursos o la aplicación de restricciones de ejecución de trabajos durante la etapa de planificación.

Sin entrar en detalles excesivamente técnicos o muy propios del ámbito de la Ingeniería del *Software*, en este apéndice se recogen los diagramas de clases, en el lenguaje *Unified Modeling Language* (UML)[144], uno por subsistema, que conforman el núcleo del EvoProc. Dichos diagramas muestran los elementos y componentes fundamentales de cada subsistema y los patrones de diseño [145] utilizados para dotar al desarrollo de la citada flexibilidad. Los componentes secundarios se obvian por claridad y legibilidad. Además de los diagramas de clases mencionados, se presentan dos diagramas de secuencia de la funcionalidad básica del EvoProc, es decir, la planificación, donde tienen cabida todos los subsistemas desarrollados y donde se puede comprobar la limpieza del diseño desarrollado a pesar de la complejidad del sistema. En las siguientes secciones se muestran dichos diagramas y

unos breves comentarios de cada uno de ellos.

A.1. Diseño del SPE

Para el desarrollo del Sistema de Planificación Evolutiva propuesto se han considerado una serie de clases que se pueden ver en el diagrama A.1, que muestra las clases fundamentales del subsistema:

EvoScheduler Es un componente fundamental y el que alberga la funcionalidad principal del sistema, la planificación.

OptimizationAlgorithm Esta clase está diseñada de acuerdo al patrón *estrategia* y permite cambiar de manera dinámica el algoritmo de optimización utilizado para la propia planificación. Ha sido utilizado en las pruebas para intercambiar el algoritmo y permite la inclusión futura de manera sencilla de cualquier otra técnica o algoritmo.

EvoSchedObjectiveFunction Es la clase que recoge las funciones que determinan la validez o calidad de las planificaciones, es decir, la materialización de las políticas de planificación.

Simulador Contiene las funciones principales de simulación, que permiten el cálculo de los tiempos de comienzo y fin de cada trabajo. Tras cada simulación, se invoca a la clase *EvoSchedObjectiveFunction* para calcular su calidad.

Job Representa un trabajo enviado al sistema por un usuario y la propia solicitud en sí (esto será modificado más adelante, como se puede ver en la siguiente sección).

PendingQueue Representa una cola de trabajos con todos los trabajos enviados por los usuarios que aun están en espera. El planificador se encarga de modificar esta cola que, finalmente, representará la planificación en sí.

ExecJob Contiene datos acerca de un trabajo ya simulado. Útil para el cálculo de la calidad de las planificaciones generadas por el algoritmo de optimización.

Resource Representa un recurso cualquiera de HPC. Para evitar soluciones *ad–hoc* se ha configurado esta clase como abstracta, pudiendo especificar cualquier tipo de recurso en el futuro, con vistas a la incorporación de recursos heterogéneos, como se hizo durante el diseño del sistema SOAR (sección A.4).

Cluster Esta clase se utiliza para representar tanto un *cluster* como un supercomputador o una máquina monoprocesador.

Así pues, las características claves del diseño del SPE se encuentran en la posibilidad de incluir cualquier técnica de optimización para la planificación, en la posibilidad de añadir recursos de diversos tipos y características y en la forma de configurar las políticas de planificación como simples funciones que evalúen los resultados de diversas simulaciones.

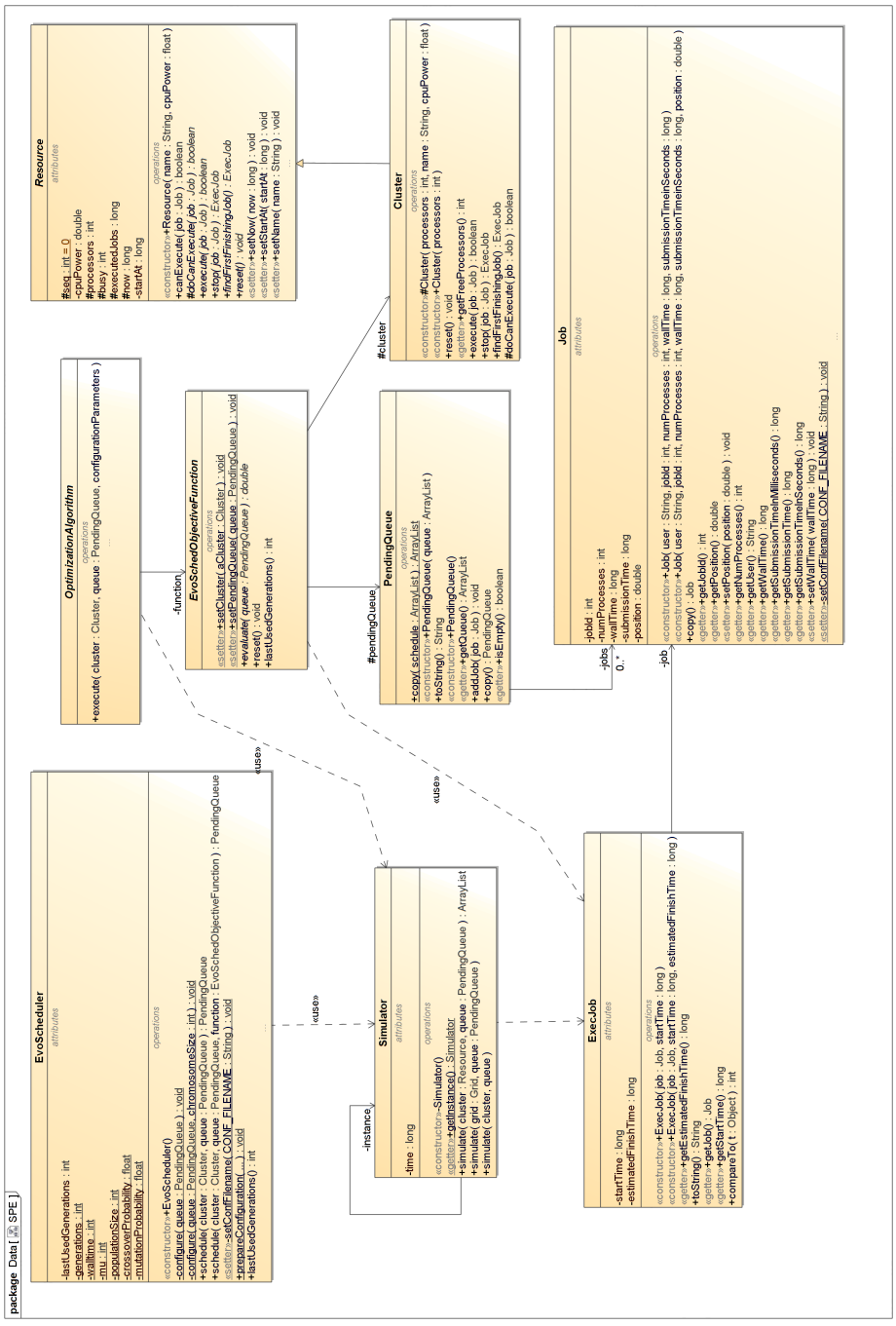

Figura A.1: Diagrama de clases para el SPE

A.2. Diseño del SOS

El Sistema de Optimización de las Solicitudes es el encargado de materializar el algoritmo de Modelado Continuo de Comportamiento propuesto en esta tesis. Para desarrollar el SOS, los cambios en el SPE han sido mínimos, pues bastó con incorporar un punto de anclaje del SOS en el algoritmo de planificación para la optimización de las solicitudes (como se puede ver en el diagrama de la figura A.6) e incorporar la posibilidad de trabajar tanto con las propias solicitudes del usuario o las solicitudes optimizadas generadas por el propio sistema.

Se destacan las siguientes clases, cuyas relaciones pueden verse en el diagrama de clases de la figura A.2:

MCCManager Es el componente encargado de manejar los modelos de usuario, es decir, de crearlos, de almacenarlos en disco y de recuperarlos. Se configura mediante el patrón estrategia para poder seleccionar de manera dinámica el algoritmo evolutivo a utilizar.

EvolutionaryAlgorithm Es la clase que representa la estrategia que encapsula al algoritmo evolutivo con el que se actualizan los modelos.

UserModel La pieza central del SOS son los modelos de usuario. Esta clase contiene la funcionalidad principal de optimizar las solicitudes. Existe un modelo por usuario y está compuesto de una población de redes de neuronas, además de una red que es la encargada de las estimaciones, tal y como se sugiere en el MCC.

NeuralNetwork Es una clase abstracta configurada como estrategia para encapsular cualquier tipo de red de neuronas para representar a los modelos de usuario.

Request Es un interfaz utilizado para independizar al SPE del tipo de solicitud con el que trabajar, ya sea la del usuario o la solicitud optimizada proporcionada por el sistema. De esta forma, se ven dos implementaciones distintas de dicho interfaz, *UserRequest* y *OptimizedRequest*.

Parameter Las solicitudes pueden variar mucho de un sistema a otro, pues existen diversos tipos de recursos susceptibles de ser solicitados. Para hacer una solución general se ha optado por diseñar las solicitudes como un conjunto de parámetros, de manera que se alcance una cierta flexibilidad. Ejemplo de dichos parámetros pueden ser tiempo de CPU, RAM o espacio de almacenamiento.

FinishedJob Agrupa la información proporcionada por el sistema operativo relativa a la ejecución de un trabajo, en concreto, los recursos utilizados por el trabajo. Es información indispensable para la actualización de los modelos.

De esta manera, el SOS permite la gestión de solicitudes con diversos parámetros y proporciona una solución sencilla (principio de diseño *programa sobre interfaces*) para que el SPE pueda trabajar con solicitudes de usuario u optimizadas.

Además se permite la selección dinámica del algoritmo evolutivo y del tipo de red de neuronas a utilizar.

A.3. Diseño del SES

El Sistema de Estimación de la Satisfacción tiene como principal objetivo la generación y actualización de modelos de satisfacción a partir de la información proporcionada por los usuarios en pequeñas encuestas realizadas por el sistema. Para añadir estas funcionalidades al sistema, el SPE y el SOS permanecen inalterados. No obstante, la manera de introducir criterios de satisfacción de los usuarios en las políticas guarda relación directa con el SPE. Como se indicó en la sección A.1, la manera de definir políticas de planificación pasa por proporcionar una sencilla función al sistema. Como parte del desarrollo del SES se han incorporado funciones estándar de maximización de la satisfacción, permitiendo además, priorizar a usuarios concretos si hiciera falta. Las principales clases de este sistema, que se pueden ver en la figura A.3 son:

SatisfactionProfile Es el componente principal y representa un perfil de satisfacción para un usuario concreto. Permite la estimación de la satisfacción para los trabajos de una planificación. La forma de calcular la satisfacción se configura mediante el patrón estrategia para poder utilizar cualquier método. Además, tiene una colección de trabajos evaluados para poder generar los perfiles y otra de trabajos finalizados para generar las encuestas de satisfacción.

SatisfactionProfileManager Es el sistema de gestión de los perfiles de satisfacción, que permite generar los modelos de satisfacción, guardarlos en disco y recuperarlos. Hace uso de una estrategia (*OptimizationAlgorithm*) que sirve para generar los perfiles. La estrategia permite cambiar de manera dinámica la tecnología utilizada.

Poll Es un sencilla encuesta generada a partir de un trabajo finalizado que es enviada al usuario cuando se conecta al sistema.

EvaluatedJob Representa la puntuación de satisfacción otorgada por el usuario tras la finalización de un trabajo concreto.

EvolPNComputeSatisfactionStrategy Representa una estrategia concreta para la generación de los modelos de satisfacción, que en este caso utiliza técnicas evolutivas para el ajuste de un polinomio.

PNEvolSatisfactionObjectiveFunction Función objetivo concreta para evaluar el ajuste del polinomio que representa un perfil de satisfacción.

Nuevamente, el uso de las estrategias facilita el intercambio de tecnologías, independizando al SES de estas y favoreciendo la inclusión de otras alternativas.

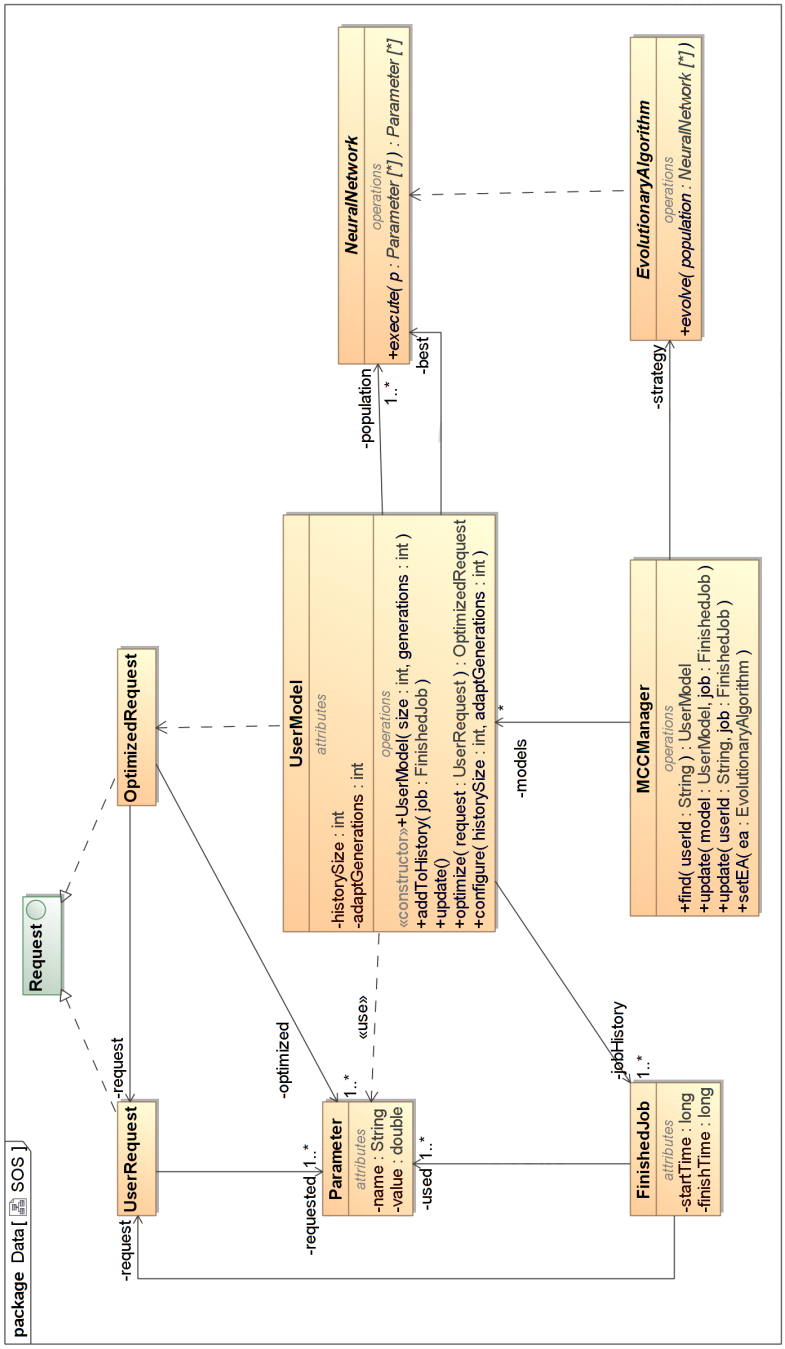

Figura A.2: Diagrama de clases para el SOS

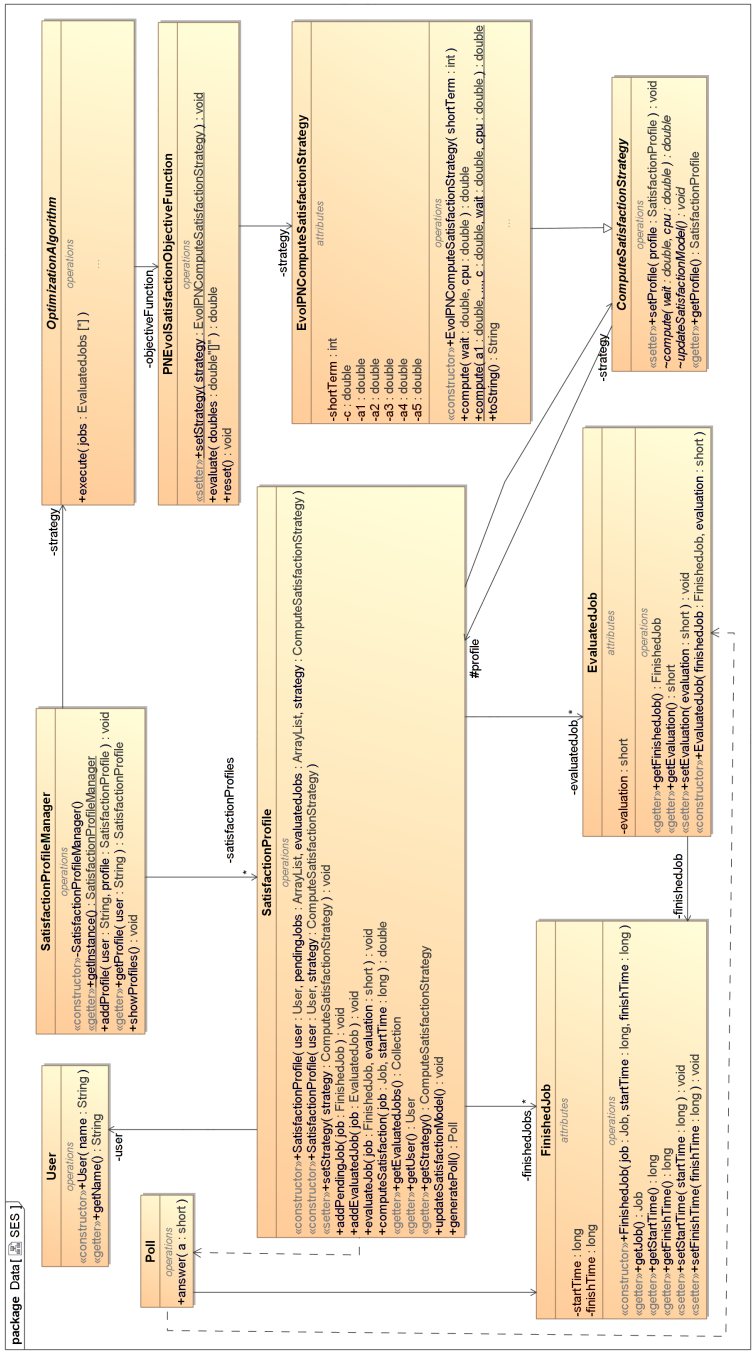

Figura A.3: Diagrama de clases para el SES

A.4. Diseño del SOAR

El Sistema de Optimización de la Asignación de Recursos es el más complejo en cuanto a número de componentes, estructuras y algoritmos. La consideración del estado y otros parámetros de los recursos implica determinados cambios en el SPE, en el que hay que incluir nuevos tipos de recursos, además de modificar los algoritmos de planificación para adaptar la simulación a dichos recursos. No obstante, el correcto diseño realizado para el SES facilitó dicha tarea. Además de las clases que se presentan a continuación, y que se pueden ver en el diagrama A.4, con el SOAR se proporcionan funciones para las políticas de planificación relacionadas con el cálculo de parámetros de los recursos, lo que hace posible la inclusión de criterios de satisfacción del centro de HPC en las políticas. Las principales clases del SOAR son:

ProfiledCluster Es una especialización de la clase *Cluster* ya presentada en el diseño del sistema SPE. Permite la creación de modelos de recursos por medio de la definición de perfiles de parámetros.

Grid Esta clase sirve para la representación de un entorno heterogéneo de recursos. Se utiliza una variante del patrón composición para poder configurar sistemas bastante completos que sean combinación de recursos sin perfilar y recursos perfilados, ya sean multiprocesador o máquinas sencillas. Permite la planificación de trabajos en dichos entornos a través del planificador *EvoScheduler*.

ResourceProfile Representa el perfil de los distintos parámetros de un recurso así como su modelo. Se puede utilizar para calcular las horas de comienzo y de fin de los trabajos en una planificación dada así como para tomar medidas de cualesquiera otros parámetros modelados, como el coste o la energía. Mediante el patrón estrategia se puede configurar la manera de realizar el perfilado. En este caso se presenta una estrategia concreta de perfilado diario (*DailyProfilingStrategy*).

ResourceProfileManager Al igual que en el SOS y el SES, esta clase se encarga de gestionar los diversos perfiles de recursos.

Sample Representa una muestra concreta de un parámetro concreto para un recurso dado. El tipo de parámetro se determina mediante el tipo enumerado *ResourceParameterType*.

ProfileParameterProcessChain Representa una cadena de comprobación de restricciones para la planificación en los distintos recursos. Aunque puede existir más, se presentan en el citado diagrama restricciones relacionadas con la carga de CPU y la disponibilidad de las máquinas.

CannotExecuteCause Debido al uso de restricciones puede suceder que alguna planificación sea inválida. Esta clase sirve para el control de este tipo de excepciones y permite asociar las causas, representadas mediante el tipo enumerado *tCannotExecuteCause*.

Lo que se buscó en el SOAR fue flexibilizar al máximo el perfilado de los recursos, tanto como la técnica utilizada para la representación de los modelos como para el muestreado de los recursos y su uso e interpretación, de ahí las estrategias utilizadas. Se ha obviado la estrategia de optimización para la generación de los modelos por analogía con la de los sistemas SPE y SES. Además, de cara a mejorar el simulador, se diseñó un sistema de comprobación de restricciones por medio del uso del patrón cadena de responsabilidad. Esto permite añadir y quitar restricciones dinámicamente a diferentes recursos para ser tenidas en cuenta durante la planificación.

A.5. Dinámica general del EvoProc

Se presenta en esta sección el diagrama de secuencia general y de alto nivel de la operación de planificación del sistema EvoProc concreto. Debido a su longitud se ha dividido en dos partes. La primera figura, A.5, muestra la secuencia de optimización de solicitudes tras la que se puede comprobar la referencia al resto de la planificación, que se puede encontrar en el diagrama de la figura A.6. En esta ocasión, el fragmento al que se hace referencia es al de la optimización, centrando la atención en el resto del proceso de planificación. Cabe destacar que a pesar de la complejidad y de los componentes involucrados, la interacción resulta directa y sencilla, prácticamente lineal, cosa que ayuda a la compresión y constata el buen diseño realizado. Esto también favorece la modificación del algoritmo en el futuro.

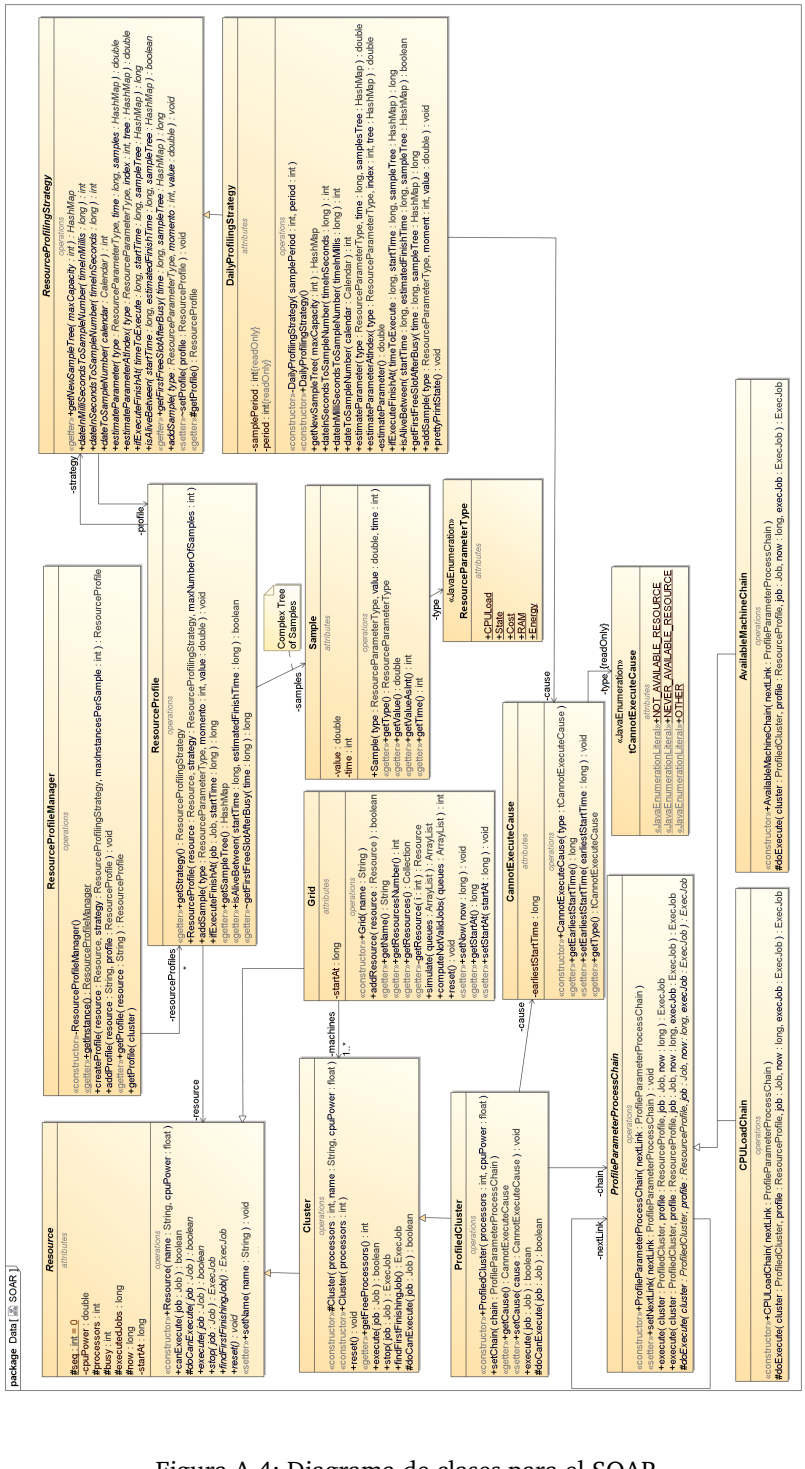

Figura A.4: Diagrama de clases para el SOAR

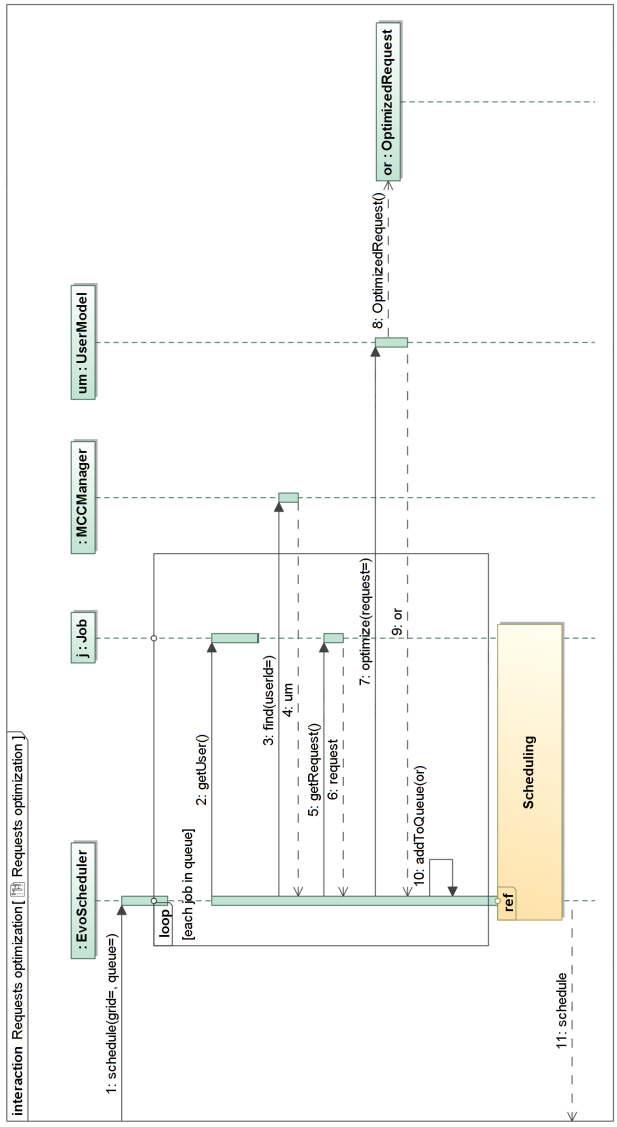

Figura A.5: Diagrama de secuencia para la optimización de solicitudes

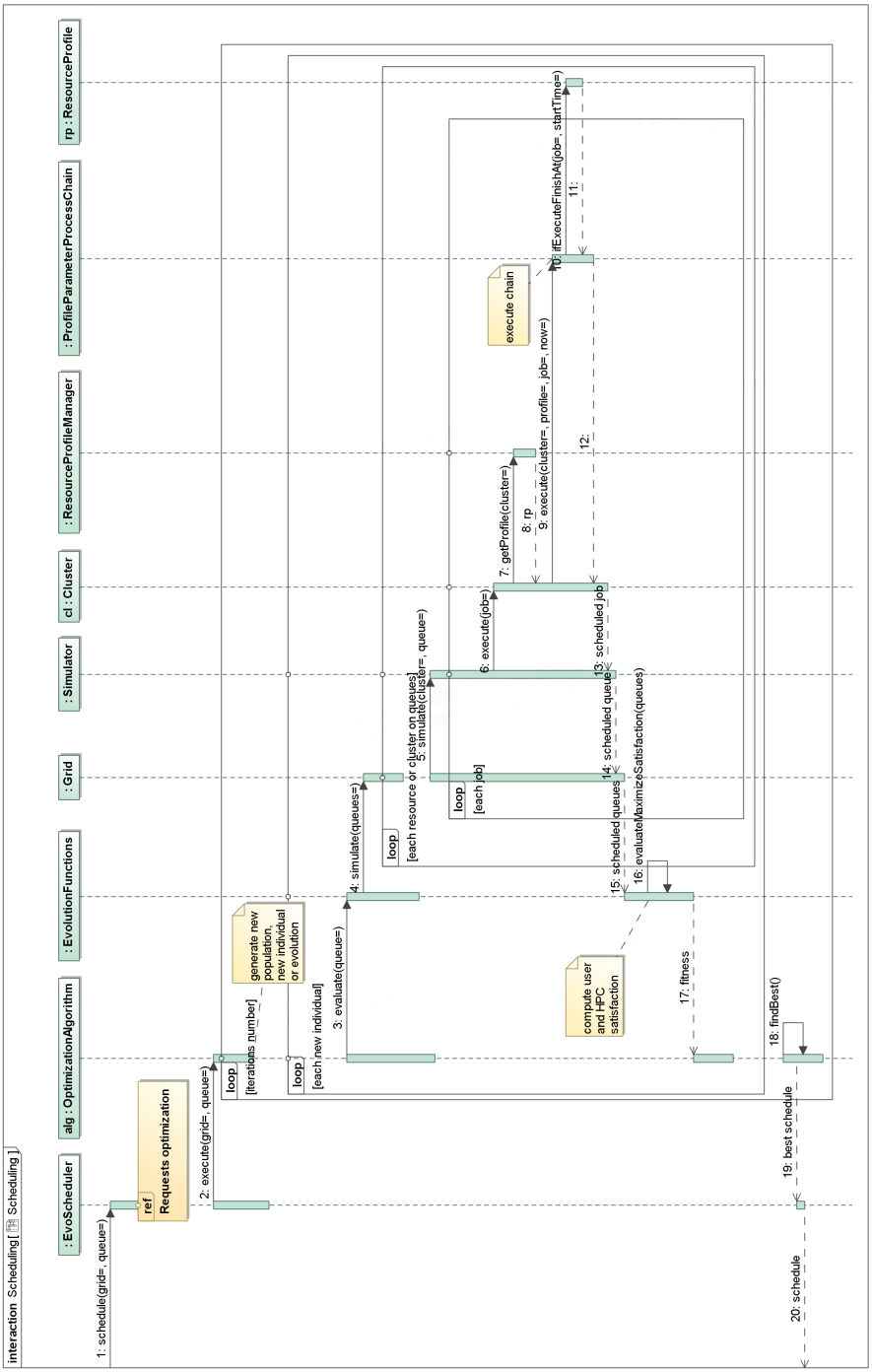

Figura A.6: Diagrama de secuencia para la planificación en un *grid*

Apéndice B

Integración con el *Sun Grid Engine*

Como se adelantó en la sección 4.3.3, este apéndice recoge brevemente los detalles más importantes de la integración del EvoProc en el sistema de gestión de colas de procesos *Sun Grid Engine* [36]. Esta integración surge en el marco del proyecto de investigación, subvencionado por la Xunta de Galicia, "Arquitectura de agentes autónomos para el modelado de usuarios y la gestión óptima de un centro de supercomputación", que fue una colaboración del Grupo Integrado de Ingeniería, perteneciente a la Universidade da Coruña, con el Centro de Supercomputación de Galicia (CESGA) [18]. A pesar de haber sido realizada en una etapa temprana del desarrollo de esta tesis, se ha querido dejar aquí reflejada la siguiente documentación con el fin de poder retomar en el futuro dicha integración, añadiendo, además, el resto de sistemas, SES y SOAR, tal y como están propuestos en este trabajo. De ahí que se proporcionen los detalles técnicos suficientes para poder abordar nuevas integraciones o mejoras, aliviando la necesidad de un estudio exhaustivo del código del SGE.

Para facilitar la integración, en el primer desarrollo de EvoProc se ha utilizando el lenguaje de programación *C* que es el lenguaje en el que el SGE está implementado. El desarrollo se comenzó para la versión SGE 6.0u3 y posteriormente se adaptó para la versión SGE 6.0u6, con lo que los detalles expuestos a continuación hacen referencia a dicha versión.

En líneas generales, la implementación del diseño presentado en las secciones A.1 y A.2 se centra en tres puntos básicos en cuanto a la capa del modelo:

1. La implementación de manera genérica e independiente de los componentes relacionados con los trabajos, las solicitudes y sus estimaciones y los recursos.

2. La implementación de una capa de adaptación del modelo de datos utilizado en el SGE al utilizado en el EvoProc. Esto pasa por un conjunto de componentes para traducir los datos manejados por el SGE para que puedan ser usados

por el SPE y viceversa.

3. La implementación del planificador evolutivo que va a sustituir al planificador SGE.

Respecto de la capa de servicios técnicos lo principalmente destacable es la implementación de los algoritmos evolutivos a integrar. En el momento de realizar esta implementación el principal algoritmo utilizado era el algoritmo genético canónico.

El punto de partida en la integración del EvoProc con el SGE pasa por el intercambio de datos, por lo que ha sido necesario el estudio de los tipos y las estructuras de datos utilizadas. Además, se puede considerar que hay que hacer una integración en dos sentidos. En primer lugar, SGE debe incluir llamadas a SPE, para realizar la planificación. Y, en segundo lugar, y en el otro sentido, el planificador del SPE hace uso del simulador y este, a su vez, invoca funciones del SGE. En las siguiente sección se abordan los tres puntos principales: la adaptación de datos, la implementación del planificador y el simulador.

B.1. Adaptación de datos

SGE utiliza una robusta y compleja librería de datos llamada *CULL*, que permite crear y mantener listas CULL, que son la estructura de datos principal donde casi todos los datos de SGE, como los trabajos, las colas o la configuración de los recursos, son almacenados. La librería tiene las siguientes características:

- Las listas son genéricas, es decir, soportan cualquier tipo de datos.
- Reusabilidad del código de gestión de listas.
- Doble interfaz y modo de uso: orientado a listas clásicas o inspirado en SQL.
- No necesita recompilación en el caso de ligeras modificaciones en las estructuras de datos.
- Búsquedas rápidas gracias al uso de tablas *hash*.

CULL posee varias estructuras internas de datos:

lList Es la cabecera de una lista CULL. Tiene campos que almacenan el número de elementos en la lista, el nombre, el descriptor, que indica el tipo de dato que almacena la lista, y punteros al primer y último elemento. La definición es como sigue:

```
typedef struct {
  int nelem;
  char *listname;
  lDescr *descr;
```

```
  lListElem *first;
  lListElem *last;
} lList;
```

lListElem Define la estructura que actúa como elemento genérico de una lista. Está compuesto de punteros a los elementos anterior y siguiente, un indicador de estado, el descriptor y un puntero de tipo *lMultiType*. El almacenamiento de datos se da en *arrays* del tipo *lMultiType*. El acceso a los datos se hace tomando el índice del *array* y el tipo del campo a través del *array* de descriptores. Queda definido por:

```
typedef struct {
  lListElem *next;
  lListElem *prev;
  lUlong status;
  lDescr *descr;
  lMultiType *cont;
} lListElem;
```

lDescr La estructura de descriptores contiene dos campos de tipo entero. Uno representa el nombre del campo mientras que el otro indica el tipo asociado a ese nombre. Estos nombres son representados por valores únicos y se pueden definir mediante enumeraciones de tipo *enum* o mediante el uso de elementos de tipo `#define`.

lMultiType Es una unión de estructuras que contiene varios tipos básicos de C. El campo *mt* de *lDescr* determina que miembro tiene que ser accedido.

```
typedef union {
  lFloat fl;   /* tipo float */
  lDouble db;  /* tipo double */
  lUlong ul;   /* tipo unsigned long */
  lLong l;     /* tipo long */
  lChar c;     /* tipo char */
  lInt i;      /* tipo int */
  lString str; /* tipo char* */
  lList *glp;  /* tipo sublist */
  lRef ref;    /* tipo puntero */
  lCondition *cp; /* tipo condición, para búsquedas */
} lMultiType;
```

La figura B.1 muestra la estructura de una lista de tipo CULL.

Para su uso es necesario la definición de listas por medio de la creación de tipos complejos. La definición se hace en tres pasos:

1. Definición de constantes que identifican los atributos de un elemento.
2. Creación de una sección que define el tipo de los atributos del elemento.
3. Definición de un conjunto de nombres para ser usados cuando un atributo requiere ser escrito de una manera legible por el ser humano.

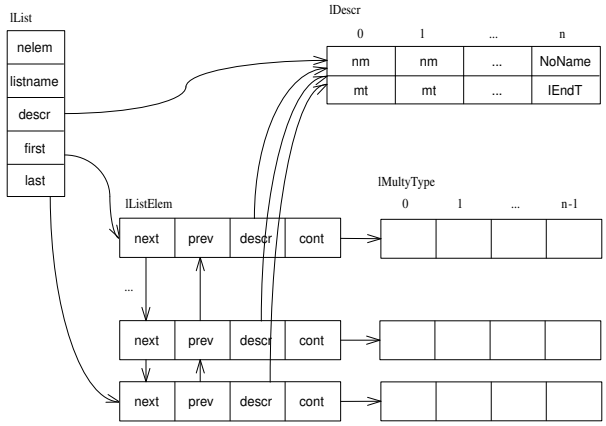

Figura B.1: Estructura de las listas de la librería CULL de SGE

Un ejemplo de la definición de un tipo puede ser el siguiente, que representa una máquina en SGE y que utiliza las *macros* `LISTDEF` y `NAMEDEF`, que facilitan la tarea:

```
enum {
  H_hostname,
  H_arch,
  H_os,
  H_memsize,
  H_queuelist
};

LISTDEF(HostT)
  SGE_HOST    (H_hostname)
  SGE_XSTRING (H_arch)
  SGE_XSTRING (H_os)
  SGE_XULONG  (H_memsize)
  SGE_XLIST   (H_queuelist, QueueT)
LISTEND

NAMEDEF(HostN)
  NAME ("H_hostname")
  NAME ("H_arch")
  NAME ("H_os")
  NAME ("H_memsize")
  NAME ("H_queuelist")
NAMEEND
```

SGE define muchos tipos más, de entre los cuales, el que más interesa de cara a la integración con EvoProc es *JB_Type*, que es el tipo que representa un trabajo.

Para mantener la mayor parte posible del SPE independiente del SGE, se ha creado una capa de adaptación que traduce listas y elementos de tipo *JB_Type* a las estructuras análogas propias de EvoProc, que son *Queue* (o *PendingQueue* en la nueva implementación) y *Job*. La figura B.2 muestra las clases del paquete de adaptación de datos. Existe un conjunto de funciones suficientemente amplio para el manejo de listas CULL que han facilitado la tarea de adaptación. De entre estas funciones se pueden destacar las siguientes:

- ***lGetNumberOfElem***. Devuelve el número de elementos de una lista.
- ***lFirst***. Sirve para obtener el primer elemento de una lista.
- ***lNext***. Devuelve el elemento siguiente de uno dado.
- ***lFindFirst***. Devuelve el primer elemento que cumple una determinada condición.
- ***lWhere***. Permite crear condiciones estilo *where* de SQL para la localización de sublistas o elementos.
- ***lGet*X**. Permite acceder a un atributo concreto, donde *X* es el tipo de dato de ese atributo.

Además, existen nuevas estructuras de control definidas, como *for_each*, que siguiendo la sintaxis `for_each (elem, list)`, permite recorrer una lista obteniendo en cada iteración un nuevo elemento en `elem`, de la misma forma en que se haría con un *iterador* [145].

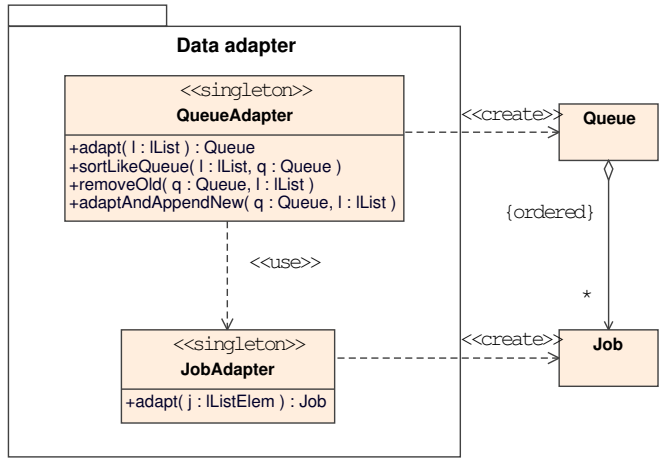

Figura B.2: Diagrama de clases de la adaptación de datos

Las clases de adaptación *JobAdapter* y *QueueAdapter* se pueden considerar *singleton* [145] y además no guardan estado. El método *adapt()* de *JobAdapter* toma

un elemento *lListElem* de tipo *JB_Type* y leyendo los atributos adecuados construye un objeto de tipo *Job*. Algunos de estos atributos son *JB_job_number*, *JB_owner*, *JB_submission_time* y *JB_ja_tasks*, que, respectivamente, guardan el identificador del trabajo, el nombre del usuario propietario, la hora de envío del trabajo y una lista con las diferentes tareas, en caso de ser un trabajo paralelo. Si el trabajo es para un único procesador, *JB_ja_tasks* solo contiene una tarea.

QueueAdapter realiza varias funciones. El método *adapt()* toma una lista *lList* de tipo *JB_Type*, en concreto las listas de trabajos pendientes y en ejecución, y crea objetos de tipo *Queue* en los que almacena otros de tipo *Job* tras hacer la adaptación utilizando la clase anteriormente comentada. El método *sortLikeQueue()* permite ordenar la lista que representa la cola de trabajos pendientes de SGE según el orden de los elementos de un objeto de tipo *Queue*. Esto se utiliza tras terminar el proceso de planificación, como se puede ver en el diagrama de la figura B.3. Los dos métodos restantes *removeOld* y *adaptAndAppendNew* permiten eliminar de un *Queue* aquellos elementos que no estén en una lista *lList* y añadir trabajos de la lista que no estén en el *Queue*. El fin de estos dos métodos es preservar la información de trabajos entre llamadas al planificador y mantener las listas *Queue* de trabajos pendientes y en ejecución actualizadas.

La integración de EvoProc con otro gestor requeriría el estudio del modelo de datos de este para hacer posible la implementación del subsistema de adaptación de datos, que puede resultar más o menos costoso en función de la complejidad de dicho modelo.

B.2. El planificador

La implementación del planificador evolutivo se abordó mediante la creación de un nuevo algoritmo de planificación en SGE, que es una modificación del algoritmo por defecto de SGE para incluir las llamadas al sistema SOS con el fin de optimizar las solicitudes.

Además de la implementación del propio planificador, se requieren otros dos pequeños cambios para que SGE pueda utilizarlo. Uno implica el registro del nuevo planificador con *qmaster*, el demonio principal de SGE, que controla al resto de componentes. El otro pasa por la inicialización de SPE en la función *main* del demonio *schedd*, el demonio encargado de ejecutar el planificador.

El nuevo planificador guarda dos estructuras *Queue* donde mantiene datos acerca de los trabajos en estado de espera y ejecución. Estos datos se conservan entre distintas llamadas al planificador. El motivo principal es el de mantener las estimaciones ya realizadas con el fin de no tener que repetirlas sobre los mismos trabajos cada vez que hay que planificar. Este paso solo es necesario si han llegado nuevos datos que permitan la actualización de los modelos de comportamiento de algún usuario y, además, dicho usuario tiene trabajos en la cola de espera. Cuando *qmaster* solicita la planificación a *schedd*, este recibe la lista de los últimos trabajos finalizados, lista que es adaptada y se utiliza para la actualización de los modelos de aquellos usuarios que tienen trabajos en ella. Después de eso, y una vez que el

planificador de SGE ha filtrado todos los trabajos y colas, para descartar trabajos inválidos o no ejecutables, y está dispuesto a servir los trabajos, es cuando se invoca al SPE, que realiza los siguientes pasos:

1. Actualiza las listas de trabajos pendientes eliminando aquellos que ya no existen (porque han sido eliminados o han terminado su ejecución, por ejemplo) y adaptando y añadiendo los nuevos.
2. Se actualizan los modelos con los trabajos finalizados desde la última planificación.
3. Se optimizan las solicitudes de los nuevos trabajos pendientes y aquellos que no son nuevos pero cuya estimación de recursos está hecha con una versión obsoleta del modelo de usuario.
4. Se prepara el simulador para que se puedan hacer las copias de los datos necesarios, indicándole además el *timestamp* de comienzo de todas las simulaciones para la planificación actual.
5. Se invoca al planificador del SPE, es decir, el método *schedule* de la clase *QueueManager* (o de la clase *EvoScheduler* en la nueva implementación), que devuelve la mejor planificación encontrada, por medio del uso del algoritmo genético, el simulador y la política de planificación configurada, que puede incluir alguna medida de satisfacción.
6. Se ordenan los trabajos pendientes en función de la planificación obtenida de SPE, tras lo que prosigue la planificación normalmente.

Se puede comprobar el flujo de la operación en el diagrama de secuencia presentado en la figura B.3.

B.3. La simulación

Es destacable el hecho de que no solo SGE necesita hacer uso del SPE, sino que este requiere la invocación de funciones del primero. Esto es debido a que el simulador se implementa en términos del SGE en lo que al planificador y ejecución de trabajos se refiere. La idea principal es hacer el simulador tan preciso para el SGE como sea posible, lo que lleva a utilizar en el simulador las funciones de SGE que se encargan de comprobar la disponibilidad de recursos, la asignación de recursos a trabajos, la puesta en ejecución, la finalización de ejecución y la liberación de recursos. Además, el hacerlo así, evita reescribir nuevo código que realiza funciones ya resueltas por el propio SGE. La principal función del simulador, como se indicó en la sección 4.3.2.4, es poder calcular la hora de comienzo y fin de cada trabajo en una planificación para poder evaluarla y compararla con otras. Así, se simula todo el proceso de ejecución de todos los trabajos para una sola planificación, para lo cual, se replica el entorno de trabajo y el estado actual de los recursos para poder operar en un entorno protegido sin afectar a la operación real. Se desgrana el proceso de simulación a continuación:

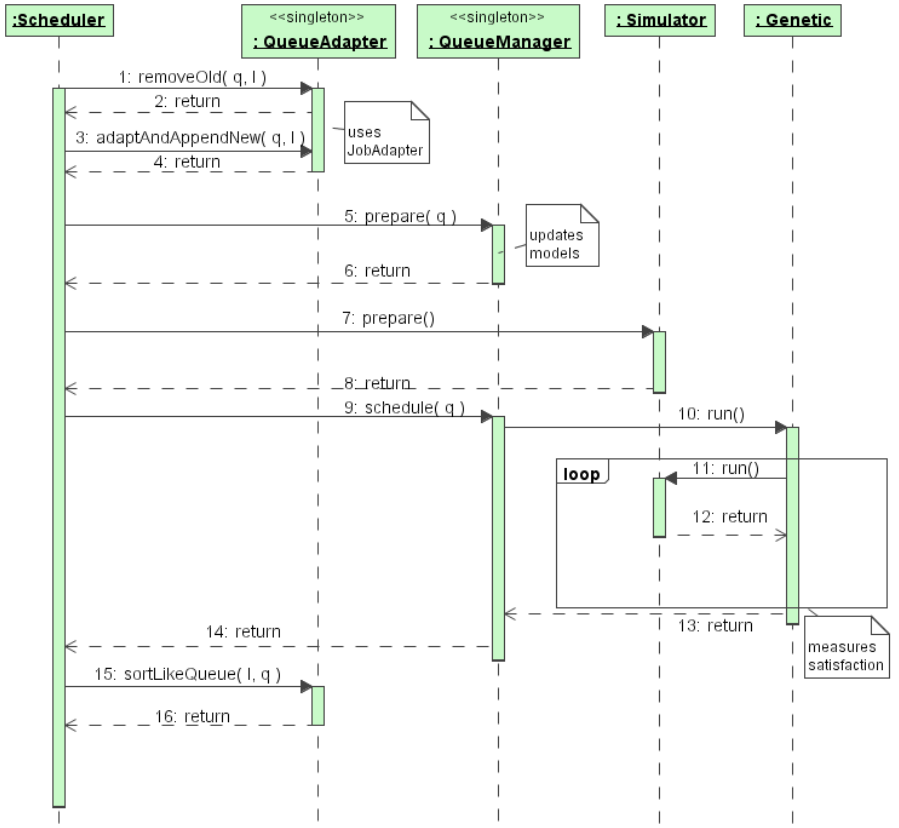

Figura B.3: Diagrama de secuencia de la planificación

1. El proceso comienza proporcionándole al simulador la planificación que se quiere simular y el estado de los recursos.

2. Se replica el entorno, para lo cual el simulador duplica diversas estructuras de datos entre las que se encuentran la siguientes:

   - La cola de trabajos pendientes, ya en el orden indicado por la planificación.
   - El conjunto de trabajos actualmente en ejecución.
   - Los recursos del sistema junto con su estado de ocupación y los trabajos a los que están sirviendo.

3. Se consultan los recursos que el primer trabajo de la cola de espera necesita para su ejecución. Si hay recursos suficientes se realizan los siguientes pasos:

   a) Se quita el trabajo de la cola de trabajos pendientes.

   b) Se pone el trabajo en el conjunto de trabajos en ejecución.

c) Se marcan los recursos a utilizar por el trabajo como ocupados.

d) Se asocian los recursos con el trabajo.

Si no hay trabajos en la cola de pendientes se continúa con el siguiente paso.

4. Se vuelve al paso anterior hasta que no queden recursos disponibles para servir al primer trabajo de la cola de pendientes.

5. Se simula el paso del tiempo hasta el evento de finalización de un trabajo más cercano. Para dicho trabajo se llevan a cabo los siguientes pasos:

   a) Se marcan como libres los recursos asociados al trabajo.

   b) Se elimina la asociación de los recursos con el trabajo.

   c) Se elimina el trabajo del conjunto de trabajos en ejecución.

6. Como han quedado nuevos recursos libres es posible que pueda pasar a ejecución otro trabajo. Si hay trabajos en la cola de pendientes se vuelve al punto 3. Si no hay trabajos en la cola de pendientes se vuelve al punto 5 hasta que no queden trabajos en el conjunto de trabajos en ejecución.

7. En este punto ya no quedan trabajos pendientes o en ejecución y se da por terminada la simulación.

8. Se destruye el entorno replicado.

Durante todo el proceso se van obteniendo y almacenando los datos de simulación necesarios, como instantes de comienzo y finalización de trabajos, ocupación de recursos, etc. Por supuesto, este proceso tal y como está aquí descrito ha sido optimizado para evitar repetir operaciones innecesarias en favor de la eficiencia.

Glosario de acrónimos

μGA	Micro–genetic Algorithm
BLX	Blend Crossover
BSC	Barcelona Supercomputing Center
CDC	Control Data Corporation
CESGA	Centro de Supercomputación de Galicia
CMA-ES	Covariance Matrix Adaptation – Evolutionary Strategy
CMIP	Common Management Information Protocol
CMOT	CMIP over TCP/IP
CODINE	Computing in Distributed Network Environment
CPU	Central Processing Unit
CULL	Common Usable List Library
CX	Cycle crossover
DE	Differential Evolution
EC2	Elastic Compute Cloud
FCFS	First Come First Served
FIFO	First In First Out
GIMPS	Great Internet Mersenne Prime Search
GUI	Graphical User Interface
HPC	High Performance Computing
IBM	International Business Machines
IP	Internet Protocol
ITER	Instituto Tecnológico de Energías Renovables
IVRS	Interactive Voice Response System
JMS	Job Management System
JSP	Job Shop Problem

KSP	Knapsack Problem
LSF	Load Sharing Facility
MCC	Modelado Continuo de Comportamiento
MDB	Multilevel Darwinist Brain
MGA	Micro–genetic Algorithm
NASA	National Aeronautics and Space Administration
NQE	Network Queueing Environment
NQS	Network Queueing System
NUDT	National University of Defense Technology
OGE	Oracle Grig Engine
OGS	Open Grig Scheduler
OGSA	Open Grid Services Arquitecture
P2P	Peer to Peer
PBS	Portable Batch System
PC	Personal Computer
QoS	Quality of Service
RAM	Random Access Memory
RBS	Recovering Beam Search
SES	Sistema de Estimación de la Satisfacción
SETI	Search for Extraterrestrial Intelligence
SGE	Sun Grig Engine
SNMP	Simple Network Management Protocol
SOAR	Sistema de Optimización de Asignación de Recursos
SOS	Sistema de Optimización de Solicitudes
SPE	Sistema de Planificación Evolutiva
SQL	Structured Query Language
TCP	Transmission Control Protocol
TSP	Traveling Salesman Problem
VoIP	Voice over IP
WAN	Wide Area Networks

Bibliografía

[1] Platform Computing. High Performance Computing. Website. http://www.scali.com/workload-management/high-performance-computing.

[2] PBS Works. PBS Works – Enabling On–Demand Computing. Website. http://www.pbsworks.com/.

[3] Oracle. Oracle Grid Engine. Website. http://www.oracle.com/technetwork/oem/grid-engine-166852.html.

[4] Maui Scheduler, Open Source Scheduler and Resource Manager. Website. http://mauischeduler.sourceforge.net/.

[5] Karl T Ulrich. *Product design and development*. Tata McGraw-Hill Education, 2003.

[6] Achim Streit. On job scheduling for hpc-clusters and the dynp scheduler. In Burkhard Monien, Viktor Prasanna, and Sriram Vajapeyam, editors, *High Performance Computing — HiPC 2001*, volume 2228 of *Lecture Notes in Computer Science*, pages 58–67. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-45307-5_6.

[7] Achim Streit. The self-tuning dynp job-scheduler. *Parallel and Distributed Processing Symposium, International*, 2:0087b, 2002.

[8] L Min. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13(4):399–403, October 1999.

[9] T Phan and K Ranganathan. Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm. *Job Scheduling Strategies for Parallel Processing*, 2005.

[10] T Loukopoulos, P Lampsas, and P Sigalas. Improved Genetic Algorithms and List Scheduling Techniques for Independent Task Scheduling in Distributed Systems. In *Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT '07. Eighth International Conference on*, pages 67–74, 2007.

[11] Eitan Frachtenberg and Dror Feitelson. Pitfalls in parallel job scheduling evaluation. In Dror Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 257–282. Springer Berlin / Heidelberg, 2005. 10.1007/11605300_13.

[12] Warren Smith, Valerie Taylor, and Ian Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pages 202–219. Springer Berlin / Heidelberg, 1999. 10.1007/3-540-47954-6_11.

[13] Hui Li, D. Groep, J. Templon, and L. Wolters. Predicting job start times on clusters. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 301–308, april 2004.

[14] Rosario M. Piro, Andrea Guarise, Giuseppe Patania, and Albert Werbrouck. Using historical accounting information to predict the resource usage of grid jobs. *Future Generation Computer Systems*, 25(5):499 – 510, 2009.

[15] Ozan Sonmez, Nezih Yigitbasi, Alexandru Iosup, and Dick Epema. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 111–120. ACM, 2009.

[16] Rueben Banalagay, KelsieJade Covington, D.M. Wilkes, and BennettA. Landman. Resource estimation in high performance medical image computing. *Neuroinformatics*, 12(4):563–573, 2014.

[17] Top 500 Supercomputer Sites. Website. `http://www.top500.org/`.

[18] CESGA. Cesga. Website. `http://www.cesga.es/`.

[19] CeSGa. Cesga - SVG. Website. `https://www.cesga.es/es/infraestructuras/computacion/svg`.

[20] CeSGa. Cesga - FinisTerrae. Website. `https://www.cesga.es/es/infraestructuras/computacion/finisterrae`.

[21] Barcelona Supercomputing Center – Centro Nacional de Supercomputación. Website. `http://www.bsc.es/`.

[22] TeideHPC. Website. `http://teidehpc.iter.es/`.

[23] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[24] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[25] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[26] Luis F. G. Sarmenta, Satoshi Hirano, and Stephen A. Ward. Towards bayanihan: building an extensible framework for volunteer computing using java. *Concurrency: Practice and Experience*, 10(11-13):1015–1019, 1998.

[27] Great Internet Mersenne Prime Search – PrimeNet. Website. `http://www.mersenne.org/`.

[28] SETI@home. Website. `http://setiathome.ssl.berkeley.edu/`.

[29] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: An experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, November 2002.

[30] Dan Werthimer, Jeff Cobb, Matt Lebofsky, David Anderson, and Eric Korpela. Seti@home—massively distributed computing for seti. *Computing in Science and Engg.*, 3(1):78–83, January 2001.

[31] IEEE standard for information technology - portable operating system interface (posix) - part 2: shell and utilities - amendment 1: batch environment. *IEEE Std 1003.2d-1994*, pages viii+140, dec 1994.

[32] NQE Introducing. Cray research publication, 1998.

[33] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor: a distributed job scheduler. In *Beowulf cluster computing with Linux*, pages 307–350. MIT press, 2001.

[34] IBM LoadLeveler. Website. `http://www-03.ibm.com/systems/power/software/loadleveler/`.

[35] F Ferstl. CODINE Technical Overview. *Genias, April*, 1993.

[36] Wolfgang Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001.

[37] Grid Engine. Open Grid Scheduler: The official Open Source Grid engine. Website. `http://gridscheduler.sourceforge.net/`.

[38] D.G. Feitelson and A.M. Weil. Utilization and predictability in scheduling the ibm sp2 with backfilling. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 542 –546, mar-3 apr 1998.

[39] A.W. Mu'alem and D.G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *Parallel and Distributed Systems, IEEE Transactions on*, 12(6):529 –543, jun 2001.

[40] D. Talby and D.G. Feitelson. Supporting priorities and improving utilization of the ibm sp scheduler using slack-based backfilling. In *Parallel and Distributed Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings*, pages 513 –517, apr 1999.

[41] Hubertus Franke, Joefon Jann, Jose Moreira, Pratap Pattnaik, and Morris Jette. An evaluation of parallel job scheduling for asci blue-pacific. *SC Conference*, 0:45, 1999.

[42] José Moreira, Hubertus Franke, Waiman Chan, Liana Fong, Morris Jette, and Andy Yoo. A gang-scheduling system for asci blue-pacific. In Peter Sloot, Marian Bubak, Alfons Hoekstra, and Bob Hertzberger, editors, *High-Performance Computing and Networking*, volume 1593 of *Lecture Notes in Computer Science*, pages 829–840. Springer Berlin / Heidelberg, 1999. 10.1007/BFb0100643.

[43] Dror Feitelson and Morris Jettee. Improved utilization and responsiveness with gang scheduling. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 238–261. Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63574-2_24.

[44] Yanyong Zhang, Hubertus Franke, Jose Moreira, and Anand Sivasubramaniam. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. *IEEE Transactions on Parallel and Distributed Systems*, 14:236–247, 2003.

[45] Joseph Skovira, Waiman Chan, Honbo Zhou, and David Lifka. The easy – loadleveler api project. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer Berlin / Heidelberg, 1996. 10.1007/BFb0022286.

[46] Y. Zhang, H. Franke, J.E. Moreira, and A. Sivasubramaniam. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 133 –142, 2000.

[47] D G Feitelson and M Naaman. Self-tuning systems. *IEEE Software*, 16(2):52–60, 1999.

[48] Chao-Chin Wu, Kuan-Chou Lai, and Ren-Yi Sun. GA-Based Job Scheduling Strategies for Fault Tolerant Grid Systems. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pages 27–32, 2008.

[49] J.M Correa and Melo. Using a classifier system to improve dynamic load balancing. In *Parallel Processing Workshops, 2001. International Conference on*, pages 411–416, 2001.

[50] M Nikravan. A genetic algorithm for process scheduling in distributed operating systems considering load balancing. In *Proceedings 21st European Conference on Modelling and Simulation*, 2007.

[51] E S H Hou, N Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, 1994.

[52] A S Wu, H Yu, S Jin, K C Lin, and G Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):824–834, September 2004.

[53] T Tsuchiya and T Osada. Genetics-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems*, 1998.

[54] M Aggarwal, R D Kent, and A Ngom. Genetic Algorithm Based Scheduler for Computational Grids. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pages 209–215. IEEE, 2005.

[55] Trilce Estrada, Olac Fuentes, and Michela Taufer. A distributed evolutionary method to design scheduling policies for volunteer computing. In *CF '08: Proceedings of the 5th conference on Computing frontiers*. ACM Request Permissions, may 2008.

[56] Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment. *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 89–96, 2010.

[57] KyriakiZ. Gkoutioudi and HelenD. Karatza. Multi-criteria job scheduling in grid using an accelerated genetic algorithm. *Journal of Grid Computing*, 10(2):311–323, 2012.

[58] Richard R. Burton and John Seely Brown. An investigation of computer coaching for informal learning activities. *International Journal of Man-Machine Studies*, 11(1):5 – 24, 1979.

[59] Gerhard Fischer and Curt Stevens. Information access in complex, poorly structured information spaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, pages 63–70, New York, NY, USA, 1991. ACM.

[60] G. Fischer. User modeling in human–computer interaction. *User Modeling and User-Adapted Interaction*, 11:65–86, 2001.

[61] Gerhard Fischer, Andreas Lemke, and Thomas Schwab. *Knowledge-based help systems*, volume 16. ACM, 1985.

[62] Robert Wilensky, David N Chin, Marc Luria, James Martin, James Mayfield, and Dekai Wu. The berkeley unix consultant project. *Computational Linguistics*, 14(4):35–84, 1988.

[63] Radboud Winkels, Joost Breuker, and Nienke denHaan. Principles and practice of knowledge representation in eurohelp. In *International Conference on the Learning Sciences (Evanston, IL), Association for the Advancement of Computing in Education, Charlottesville, Virginia*, pages 442–448, 1991.

[64] Tom Fawcett and Foster Provost. Combining data mining and machine learning for effective user profiling. *Proceedings on the Second International Conference on Knowledge Discovery and Data Mining*, pages 8–13, 1996.

[65] D.L. Pepyne, Jinghua Hu, and Weibo Gong. User profiling for computer security. In *American Control Conference, 2004. Proceedings of the 2004*, volume 2, pages 982 –987 vol.2, 30 2004-july 2 2004.

[66] Benny P. L. Lo, Jeffrey L. Wang, and Guang zhong Yang. From imaging networks to behavior profiling: Ubiquitous sensing for managed homecare of the elderly. In *Adjunct Proceedings of the 3rd International Conference on Pervasive Computing*, 2005.

[67] T. Kuflik and C. Rocchi. User modelling and adaptation for a museum visitors' guide. In Oliviero Stock and Massimo Zancanaro, editors, *PEACH - Intelligent Interfaces for Museum Visits*, Cognitive Technologies, pages 121–144. Springer Berlin Heidelberg, 2007. 10.1007/3-540-68755-6_6.

[68] Alex Pentland and Andrew Liu. Modeling and prediction of human behavior. *Neural computation*, 11(1):229–242, 1999.

[69] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[70] Alfred Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11:49–63, 2001. 10.1023/A:1011187500863.

[71] Giorgio Brajnik and Carlo Tasso. A shell for developing non-monotonic user modeling systems. *International Journal of Human-Computer Studies*, 40(1):31–62, 1994.

[72] Alfred Kobsa and Wolfgang Pohl. The user modeling shell system bgp-ms. *User Modeling and User-Adapted Interaction*, 4(2):59–106, 1994.

[73] Jon Orwant. *Doppelgänger–a user modeling system*. PhD thesis, Massachusetts Institute of Technology, 1991.

[74] Judy Kay. The um toolkit for cooperative user modelling. *User Modeling and User-Adapted Interaction*, 4(3):149–196, 1994.

[75] R. Yasdi. A literature survey on applications of neural networks for human-computer interaction. *Neural Computing & Applications*, 9:245–258, 2000. 10.1007/s005210070002.

[76] Su-Hui Chiang, Andrea Arpaci-Dusseau, and Mary K Vernon. The impact of more accurate requested runtimes on production job scheduling performance. In *Job Scheduling Strategies for Parallel Processing*, pages 103–127. Springer, 2002.

[77] William A Ward Jr, Carrie L Mahood, and John E West. Scheduling jobs on parallel systems using a relaxed backfill strategy. In *Job Scheduling Strategies for Parallel Processing*, pages 88–102. Springer, 2002.

[78] Walfredo Cirne and Francine Berman. A comprehensive model of the supercomputer workload. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 140–148. IEEE, 2001.

[79] Daniel Nurmi, John Brevik, and Rich Wolski. Qbets: Queue bounds estimation from time series. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 4942 of *Lecture Notes in Computer Science*, pages 76–101. Springer Berlin Heidelberg, 2008.

[80] Rajath Kumar and Sathish Vadhiyar. Identifying quick starters: Towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs. In Walfredo Cirne, Narayan Desai, Eitan Frachtenberg, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 7698 of *Lecture Notes in Computer Science*, pages 196–215. Springer Berlin Heidelberg, 2013.

[81] Rajath Kumar and Sathish Vadhiyar. Prediction of queue waiting times for metascheduling on parallel batch systems. In *Job Scheduling Strategies for Parallel Processing*, pages 108–128. Springer, 2014.

[82] Hui Li, Juan Chen, Ying Tao, David Gro, and Lex Wolters. Improving a local learning technique for queuewait time predictions. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 335–342. IEEE, 2006.

[83] NIKHEF. National Institute for Subatomic Physics. Website. `http://www.nikhef.nl/en/`.

[84] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times using historical information. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 122–142. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0053984.

[85] Wei Tang, N. Desai, D. Buettner, and Zhiling Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–11, April 2010.

[86] Wei Tang, Narayan Desai, Daniel Buettner, and Zhiling Lan. Job scheduling with adjusted runtime estimates on production supercomputers. *Journal of Parallel and Distributed Computing*, 73(7):926–938, 2013.

[87] Xin Chen, Charng-Da Lu, and K. Pattabiraman. Predicting job completion times using system logs in supercomputing clusters. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pages 1–8, June 2013.

[88] James E. Bailey and Sammy W. Pearson. Development of a tool for measuring and analyzing computer user satisfaction. *Management Science*, 29(5):530–545, 1983.

[89] Blake Ives, Margrethe H. Olson, and Jack J. Baroudi. The measurement of user information satisfaction. *Commun. ACM*, 26:785–793, October 1983.

[90] William J. Doll and Gholamreza Torkzadeh. The measurement of end-user computing satisfaction. *MIS Q.*, 12:259–274, June 1988.

[91] A W Gatian. Is user satisfaction a valid measure of system effectiveness? *Information & Management*, 26(3):119–131, 1994.

[92] Maarten Gelderman. The relation between user satisfaction, usage of information systems and performance. *Inf. Manage.*, 34:11–18, August 1998.

[93] Charles E. Downing. System usage behavior as a proxy for user satisfaction: an empirical investigation. *Information & Management*, 35(4):203 – 216, 1999.

[94] Adel M. Aladwani and Prashant C. Palvia. Developing and validating an instrument for measuring user-perceived web quality. *Information & Management*, 39(6):467 – 476, 2002.

[95] Joanne Sullivan, Rens Scheepers, and Catherine Middleton. Conceptualizing user satisfaction in the ubiquitous computing era. *ICIS 2009 Proceedings*, (103), 2009.

[96] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. Quantifying skype user satisfaction. *SIGCOMM Comput. Commun. Rev.*, 36:399–410, August 2006.

[97] PeterA. Dinda and DavidR. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, 2000.

[98] Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Perform. Eval. Rev.*, 30(4):41–49, March 2003.

[99] Lazar Adzigogov, John Soldatos, and Lazaros Polymenakos. Emperor: An ogsa grid meta-scheduler based on dynamic resource predictions. *Journal of Grid Computing*, 3(1-2):19–37, 2005.

[100] Ian Foster, Carl Kesselman, and Steven Tuecke. 17 chapter the open grid services architecture. *The Grid: Blueprint for a New Computing Infrastructure*, 1, 2004.

[101] F. Nadeem, R. Prodan, and T. Fahringer. Characterizing, modeling and predicting dynamic resource availability in a large scale multi-purpose grid. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 348–357, May 2008.

[102] Erik Elmroth and Johan Tordsson. Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. *Future Generation Computer Systems*, 24(6):585 – 593, 2008.

[103] G. Reig, J. Alonso, and J. Guitart. Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud. In *Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on*, pages 162–167, July 2010.

[104] Karthick Ramachandran, Hanan Lutfiyya, and Mark Perry. Decentralized approach to resource availability prediction using group availability in a P2P desktop grid. *Future Generation Computer Systems*, 28(6):854 – 860, 2012. Including Special sections SS: Volunteer Computing and Desktop Grids and SS: Mobile Ubiquitous Computing.

[105] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131, April 2008.

[106] Rajkumar Buyya and Manzur Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[107] Eugene L Lawler. The traveling salesman problem: a guided tour of combinatorial optimization. *WILEY-INTERSCIENCE SERIES IN DISCRETE MATHEMATICS*, 1985.

[108] Darrell Whitley, Timothy Starkweather, and Dan Shaner. *The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination*. Citeseer, 1991.

[109] Colin R Reeves. A genetic algorithm for flowshop sequencing. *Computers & operations research*, 22(1):5–13, 1995.

[110] Chiung Moon and Yoonho Seo. Evolutionary algorithm for advanced process planning and scheduling in a multi-plant. *Computers & Industrial Engineering*, 48(2):311 – 325, 2005.

[111] Kalmanje Krishnakumar. Micro-genetic algorithms for stationary and non-stationary function optimization. *Proc. SPIE*, 1196:289–296, 1990.

[112] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9:159–195, June 2001.

[113] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11:341–359, December 1997.

[114] M. Ghirardi and C.N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457 – 467, 2005. Project Management and Scheduling.

[115] F. Della Croce, M. Ghirardi, and R. Tadei. Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10:89–104, 2004.

[116] Purushothaman Damodaran and Mario Velez-Gallego. Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *The International Journal of Advanced Manufacturing Technology*, 49:1119–1128, 2010. 10.1007/s00170-009-2457-1.

[117] Ghazvini FJ Dupont L. Minimizing makespan on a single batch processing machine with non-identical job sizes. *Eur J Autom Syst*, 32:431–440, 1998.

[118] L ESHLEMAN. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms*, 2:187–202, 1993.

[119] IM Oliver, DJd Smith, and John RC Holland. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[120] Pilar Caamaño Sobrino. *Caracterización de espacios de calidad y algoritmos evolutivos en problemas de optimización con codificación real*. PhD thesis, Universidade da Coruña, 2011.

[121] Pilar Caamaño, Francisco Bellas, Jose A Becerra, and Richard J Duro. Evolutionary algorithm characterization in real parameter optimization problems. *Applied Soft Computing*, 13(4):1902–1921, 2013.

[122] Introducing Titan | The World's #1 Open Science Supercomputer. Website. `https://www.olcf.ornl.gov/titan/`.

[123] F. Bellas, R.J. Duro, A. Faina, and D. Souto. Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots. *Autonomous Mental Development, IEEE Transactions on*, 2(4):340–354, Dec 2010.

[124] Francisco J. Bellas Bouza. *MDB: Mecanismo congnitivo darwinista para agentes autónomos*. PhD thesis, Universidade da Coruña, 2003.

[125] D. E. Rumelhart, G. E. Hinton, and R. K. Williams. Learning representations by backpropagating errors. *Nature*, 323(9):533–536, 1986.

[126] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[127] Charles E Osgood. The nature and measurement of meaning. *Psychological bulletin*, 49(3):197, 1952.

[128] Michal Kosinski, David Stillwell, and Thore Graepel. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, 110(15):5802–5805, 2013.

[129] Alex Timian, Sonia Rupcic, Stan Kachnowski, and Paloma Luisi. Do patients "like" good care? measuring hospital quality via facebook. *American Journal of Medical Quality*, page 1062860612474839, 2013.

[130] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, Jan 2003.

[131] Margaret M Bradley and Peter J Lang. Measuring emotion: the self-assessment manikin and the semantic differential. *Journal of behavior therapy and experimental psychiatry*, 25(1):49–59, 1994.

[132] Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. Scheduling in hpc resource management systems: Queuing vs. planning. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2003.

[133] Yeo Keun Kim, Kitae Park, and Jesuk Ko. A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. *Computers & Operations Research*, 30(8):1151 – 1171, 2003.

[134] Alejandro Lucero. Simulation of batch scheduling using real production-ready software tools. *Proceedings of the 5th IBERGRID*, 2011.

[135] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *evolutionary computation, IEEE transactions on*, 3(4):257–271, 1999.

[136] A.J. Younge, G. von Laszewski, Lizhe Wang, S. Lopez-Alarcon, and W. Carithers. Efficient resource management for cloud computing environments. In *Green Computing Conference, 2010 International*, pages 357–364, Aug 2010.

[137] Sukhpal Singh and Inderveer Chana. Energy based efficient resource scheduling: a step towards green computing. *Int J Energy Inf Commun*, 5(2):35–52, 2014.

[138] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual clusters for grid communities. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 513–520, May 2006.

[139] AWS | Amazon Elastic Compute Cloud (EC2). Website. `http://aws.amazon.com/es/ec2/`.

[140] Microsoft Azure: plataforma y servicios de informática en la nube. Website. `http://azure.microsoft.com/`.

[141] Adaptive Computing. TORQUE Resource Manager. Website. `http://www.adaptivecomputing.com/products/open-source/torque/`.

[142] Jerry B Gotlieb, Dhruv Grewal, and Stephen W Brown. Consumer satisfaction and perceived quality: complementary or divergent constructs? *Journal of applied psychology*, 79(6):875, 1994.

[143] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[144] James Rumbaugh, Ivar Jacobson, and Grady Booch, editors. *The Unified Modeling Language reference manual*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[145] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

Índice de figuras

Índice de tablas