PLANIFICACIÓN DESCENTRALIZADA BASADA EN SISTEMAS MULTIAGENTE PARA ORQUESTADORES EN LA NIEBLA

O. Casquero, A. Armentia, I. Sarachaga, D. Orive, M. Marcos
Dpto. Ingeniería de Sistemas y Automática, UPV/EHU, España
{oskar.casquero, aintzane.armentia, isabel.sarachaga, dario.orive, marga.marcos}@ehu.eus

Resumen

La computación en la niebla (fog computing) es un paradigma que aprovecha, por una parte, la reducción que se produce en las latencias al acercar la infraestructura de computación a los dispositivos que recopilan datos, y, por otra parte, las funciones de alta capacidad de almacenamiento y procesamiento de la nube. La computación en la niebla puede usarse para mejorar la capacidad de control de los procesos de automatización mediante la introducción de un bucle de control de alto nivel: Fog-in-the-Loop (FIL). FIL permite capturar datos de la planta, procesarlos para extraer información de valor añadido y retroalimentar la planta. Por lo tanto, las aplicaciones FIL son aplicaciones sensibles al contexto que requieren el despliegue de componentes distribuidos y la reconfiguración dinámica. Este artículo describe un planificador para el orquestador Kubernetes (K8s) que distribuye la tarea de planificación entre los nodos de procesamiento mediante un sistema multiagente. Este nuevo enfoque de planificación demostró ser más rápido que el enfoque de planificación centralizado utilizado por el planificador por defecto de K8s.

Palabras clave: Planificador, Orquestador, Kubernetes, K8s, Sistemas Multiagente, MAS, JADE, Fog-in-the-Loop.

1 INTRODUCCIÓN

En la actualidad, en el entorno industrial se están adoptando tecnologías emergentes (tales como *big data*, internet de las cosas, redes de sensores inteligentes y conjuntos compartidos de recursos de procesamiento reconfigurables) para conseguir la transformación a la Industria 4.0. Una de las ventajas derivadas es la capacidad de captar una gran cantidad de datos y extraer información útil que contribuya a la mejora y optimización de los procesos productivos [1]-[3]. Esto, a su vez, abre la posibilidad a la introducción de niveles superiores de control, realimentando el estado de la planta productiva y pudiendo actuar con antelación ante situaciones de degradación. Sin embargo, las latencias no deseadas, así como los problemas de seguridad, impiden cerrar el lazo desde la nube en muchos casos. Ésta es la razón por la que se propone el procesamiento en la niebla (*fog computing*).

El procesamiento en la niebla es un paradigma situado a medio camino entre el *cloud computing* y el *edge computing* [4]-[6]. Las arquitecturas en la niebla están destinadas a ofrecer servicios de alto nivel, de características similares a los de la nube, pero más cercanos a la planta [7]-[10]. Sin embargo, a pesar del reconocimiento creciente de la computación en la niebla en el entorno industrial, esta línea de investigación es relativamente reciente [11]-[13]. Este artículo presenta un primer paso para habilitar sistemas *Fog-in-the-Loop* (FIL), dando soporte a las necesidades de estos sistemas y superando los problemas asociados a la nube. La Figura 1 muestra la arquitectura FIL, en la que se introduce una capa superior de control que permite a) capturar y almacenar datos de la planta, b) procesar dichos datos para extraer información y c) actuar en caso de detección de situaciones no deseadas.

Las aplicaciones FIL son aplicaciones sensibles al contexto que requieren el despliegue de componentes distribuidos y la reconfiguración dinámica. Esto conlleva una planificación eficiente que permita el despliegue de este tipo de aplicaciones en nodos con capacidades heterogéneas y flexibles para poder adaptarse a cambios en su entorno. En este sentido, el uso de un orquestador de aplicaciones resulta indispensable. Los orquestadores de aplicaciones de código abierto se pueden personalizar e integrar con otro tipo de sistemas para afrontar los requisitos de las aplicaciones FIL, e.g., la selección de los nodos de despliegue durante la reconfiguración teniendo en cuenta las latencias en la red [14]. En este contexto, este artículo describe un planificador para adaptar Kubernetes (K8s, el orquestador de aplicaciones en la nube de referencia en la actualidad) a las características de las aplicaciones FIL. Concretamente, se propone mejorar la selección de los nodos de procesamiento donde desplegar aplicaciones mediante la descentralización, basada en sistemas multiagente (MAS, Multi-Agent Systems), del proceso de planificación.

https://doi.org/10.17979/spudc.9788497497169.126

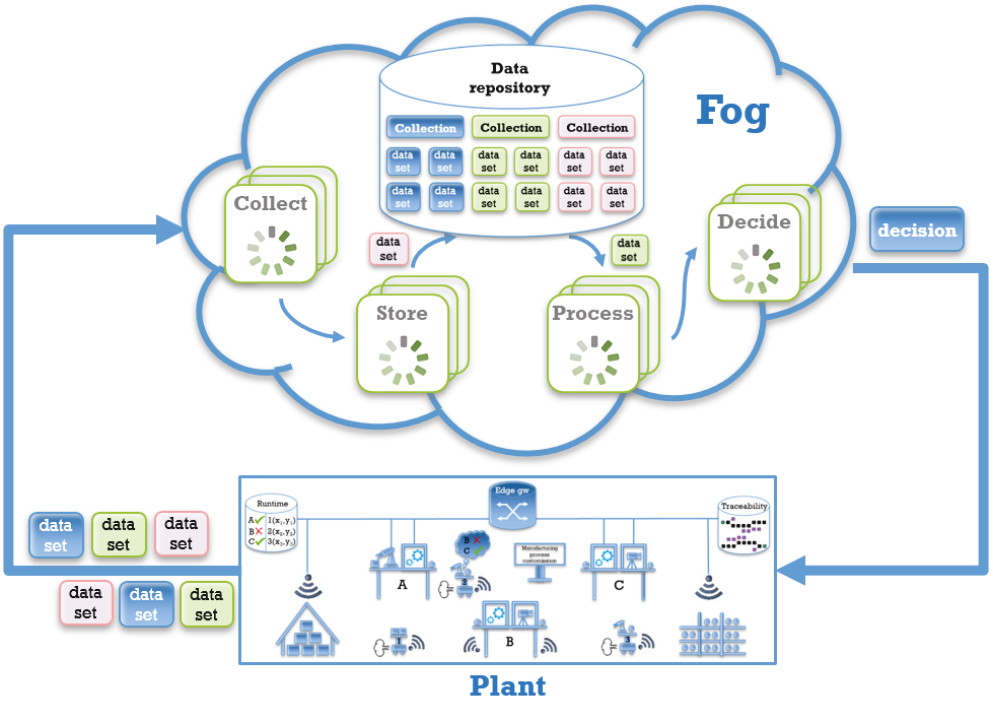

Figura 1: Arquitectura de un sistema FIL.

El resto de este artículo se estructura de la siguiente forma: En la sección 2 se presentan la arquitectura y el proceso de despliegue de aplicaciones de K8s. La sección 3 describe la arquitectura y el funcionamiento de un nuevo planificador que permite descentralizar el proceso de planificación mediante MAS. En la sección 4 se presentan los resultados de la evaluación del nuevo planificador. Finalmente, la sección 5 presenta las conclusiones y el trabajo futuro.

2 K8s: ARQUITECTURA Y DESPLIEGUE DE APLICACIONES

2.1 ARQUITECTURA DE K8s

La Figura 2 muestra la arquitectura de un clúster K8s. En ella se distinguen dos tipos de nodos: el nodo maestro y los nodos de procesamiento. En estos nodos se ejecutan dos tipos de procesos: los componentes ejecutables de la aplicación y los componentes del plano de control encargados de gestionar los anteriores.

K8s gestiona la ejecución de objetos denominados *pods*, que contienen componentes de aplicación. Los *pods* constituyen los bloques básicos para el despliegue de aplicaciones en K8s mediante contenedores. Un contenedor es una aplicación virtualizada que encapsula el código y las dependencias de una aplicación en un paquete ligero que puede transferirse y ejecutarse rápidamente en un nodo de procesamiento. Concretamente, en un *pod* se encapsulan uno o varios contenedores que se ejecutan conforme a unas determinadas especificaciones indicadas en el fichero de despliegue de la aplicación.

Los componentes del plano de control son un conjunto de procesos que trabajan de forma coordinada para asegurar que la aplicación alcanza y mantiene el estado deseado (en cuanto a redundancia, disponibilidad, escalabilidad, etc.) durante su ejecución. Todos los componentes del plano de control se ejecutan en el nodo maestro, a excepción del denominado *kubelet*, que se ejecuta en cada uno de los nodos de procesamiento. Los componentes del plano de control no interactúan directamente, sino que lo hacen a través de la información almacenada en una base de datos de tipo clave-valor denominada *etcd*. El componente *API server* actúa de intermediario en el acceso a la *etcd* por parte del resto de componentes, haciendo uso de un API REST (es decir, mediante protocolo HTTP) como interfaz. El componente *controller* monitoriza el estado del

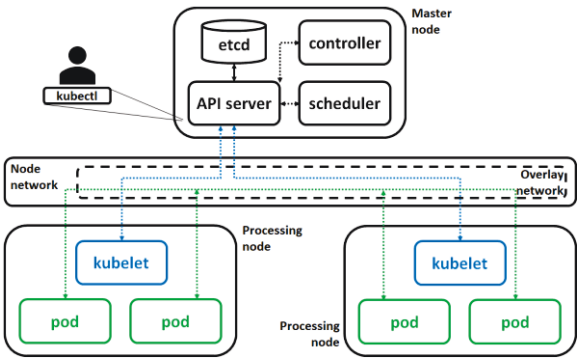

Figura 2: Arquitectura de K8s.

despliegue de la aplicación y realiza las operaciones necesarias para asegurar su correcta ejecución. El componente *scheduler* (de aquí en adelante, planificador) implementa la lógica de planificación. La planificación se refiere al método mediante el cual se asignan nodos a *pods* en función de los requisitos de recursos y las restricciones a nodo indicadas en el fichero de despliegue de la aplicación. Por último, el componente *kubelet* se encarga de arrancar los *pods* de acuerdo con las decisiones tomadas por el planificador. El componente *kubectl* es una herramienta de línea de comandos para la gestión del clúster por parte de un administrador.

La comunicación entre los *pods* y entre los componentes del plano de control se realiza a través de una red superpuesta (*overlay network*) a la red de los nodos. Esta red se divide en una serie subredes enlazadas lógicamente. A cada nodo del clúster K8s se le asigna una subred que constituye el medio lógico que interconecta los contenedores dentro de un nodo y los contenedores en diferentes nodos.

2.2 DESPLIEGUE DE UNA APLICACIÓN EN K8s

En la Figura 3 se muestra el diagrama de secuencia del despliegue de una aplicación sin réplicas en K8s, cuyas operaciones se detallan a continuación.

Pasos 1, 2: *Solicitud de despliegue de la aplicación por parte del usuario*. Como resultado de esta acción, se crea un objeto *v1Deployment* en el que se registra la petición: *desiredReplica=1* y *availableReplica=0*.

Pasos 3-6: *Solicitud de arranque del pod*. Cualquier cambio introducido en un tipo de objeto de la *etcd* puede ser monitorizado por otros componentes mediante un *Watcher*. Así, el *controller* detecta el cambio en la *etcd* y realiza las acciones necesarias para que la aplicación alcance el estado deseado: *availableReplica=1*. En este paso, el *controller* crea un objeto *v1Pod* con el que solicita el arranque de un *pod*: *status=Pending* y *nodeName=none*.

Pasos 7-10: *Planificación del pod*. El planificador detecta la solicitud de arranque del *controller* y asigna un nodo de procesamiento al *pod* mediante un algoritmo de planificación centralizado que tiene en cuenta los requisitos de recursos y las restricciones a nodo de la aplicación. Como resultado, se crea un objeto *v1Binding* en el que se registra el vínculo pod-nodo: *name=PodName* y *targetNodeName=node1*.

Pasos 11-14: *Arranque del pod*. El *kubelet* del nodo con identificador *node1* detecta el cambio en la *etcd* y realiza las acciones necesarias para arrancar el *pod* (e.g., descargar la imagen del contenedor). Al finalizar, el *kubelet* actualiza el objeto *v1Pod*: *status=Running* y *nodeName=node1*.

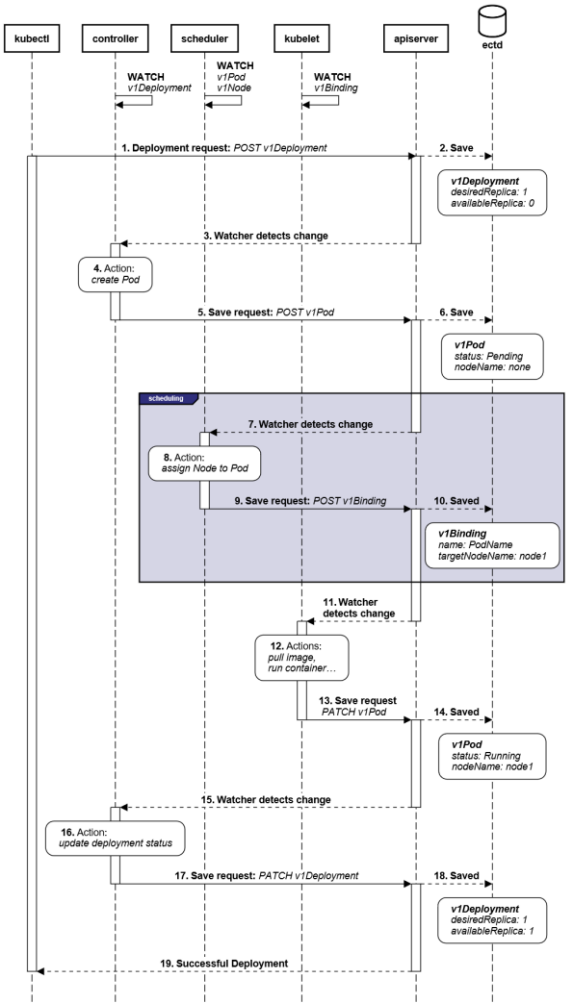

Figura 3: Diagrama de secuencia del proceso de despliegue de una aplicación en K8s

Pasos 15-18: *Actualización de estado del despliegue*. El *controller* detecta el cambio en la *etcd* y actualiza el atributo *availableReplica* del objeto *v1Deployment*. Dado que el estado deseado de la aplicación coincide con su estado actual, el *controller* da por finalizado el despliegue.

3 PLANIFICACION DISTRIBUIDA BASADA EN MAS EN K8s

3.1 PLANIFICADOR POR DEFECTO EN K8s (PLANIFICADOR CENTRALIZADO)

El planificador de K8s opera de forma centralizada en el nodo maestro para realizar las operaciones 7-10 indicadas en la Figura 3, a saber:

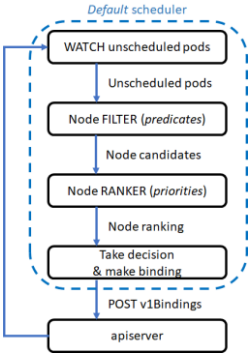

Figura 4: Ciclo de vida del planificador de K8s.

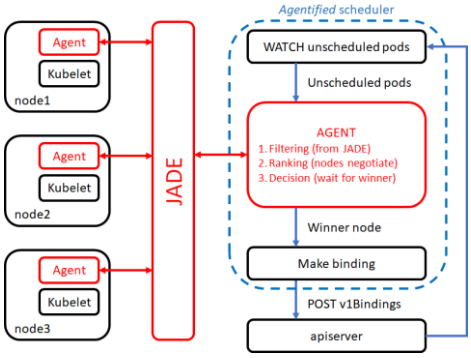

Figura 5: Ciclo de vida del planificador agentificado.

- Detectar, a través de la *etcd*, aquellos *pods* sin planificar registrados por el *controller*.
- Ejecutar un algoritmo de planificación que asigna los nodos más adecuados para ejecutar dichos *pods*.
- Registrar en la *etcd* el vínculo pod-nodo para que los *kubelets* en dichos nodos puedan arrancar los *pods*.

El algoritmo de planificación se compone de dos partes: filtrado y clasificación. El filtrado de nodos está basado en *predicados*, funciones que devuelven un valor booleano que indica si un determinado nodo puede ejecutar un *pod*. Por lo tanto, los predicados permiten obtener un listado de nodos candidatos para ejecutar un *pod*. La clasificación de nodos está basada en *prioridades*, funciones que devuelven un valor que se utiliza para ordenar el listado de nodos candidatos según su idoneidad para ejecutar un *pod*.

Tanto los predicados como las prioridades aceptan diferentes criterios de selección de nodos, tales como requisitos de recursos (e.g., CPU, memoria), restricciones a nodo (e.g., un *pod* persistente puede requerir un nodo con un volumen de almacenamiento) y afinidad entre *pods* (e.g., que dos *pods* se puedan ejecutar de forma conjunta en el mismo nodo para reducir la latencia entre ellos).

La Figura 4 muestra el ciclo de vida del planificador por defecto en K8s.

3.2 PLANIFICADOR AGENTIFICADO (PLANIFICADOR DISTRIBUIDO)

K8s ofrece distintas opciones de personalización, entre las cuales destaca la posibilidad introducir planificadores de terceros. La única condición es que el resultado de la planificación (el vínculo pod-nodo) se almacene en la *etcd*, de forma que el *kubelet* pueda seguir operando de forma transparente.

Esta característica de K8s ofrece la posibilidad de crear un planificador personalizado para reducir la carga de procesamiento en el nodo maestro y la carga de interacción contra el conjunto *etcd-API server*. Para ello, en este trabajo se plantea la creación de un nuevo planificador que incorpora un agente que delega la operación de filtrado en una plataforma MAS y distribuye las operaciones de clasificación y decision entre agentes ubicados en los nodos candidatos. La Figura 5 muestra el ciclo de vida de este nuevo planificador *agentificado*.

El planificador agentificado es un componente capaz de comunicarse tanto con el cluster K8s como con el MAS. Su secuencia de operación se describe en la Figura 6:

Paso 1: *Obtención del listado de pods pendientes de asignación de un nodo*. Esta operación se realiza en K8s mediante consulta a la *etcd*.

Pasos 2, 3: *Obtención del listado de nodos candidatos a ejecutar un pod.* Esta operación tiene lugar en el MAS mediante consulta al DF (Directory Facilitator) donde están registrados los servicios de cada nodo.

Pasos 4-6: *Negociación entre nodos*. El agente en el planificador pone a negociar a los agentes en los nodos candidatos para que uno de ellos se proponga para ejecutar el *pod*. La negociación tiene lugar según se indica en la Figura 7: el agente en el planificador envía la solicitud de negociación; cada nodo calcula un valor (con la misma función de *prioridad*) que determina su idoneidad para ejecutar el *pod*, se lo envía al resto de nodos y queda a la espera de recibir los valores del resto; el nodo vencedor informa al agente en el planificador de que ha ganado la negociación.

Paso 7: *Almacenamiento del vínculo pod-nodo en la etcd*. El planificador crea un objeto *v1Binding* en el que se registra el vínculo pod-nodo.

El planificador agentificado se programó en Java usando las librerías de cliente para K8s y JADE (Java Agent DEvelopment Framework), y se desplegó en el plano de control de K8s dentro de un *pod*. Para hacer

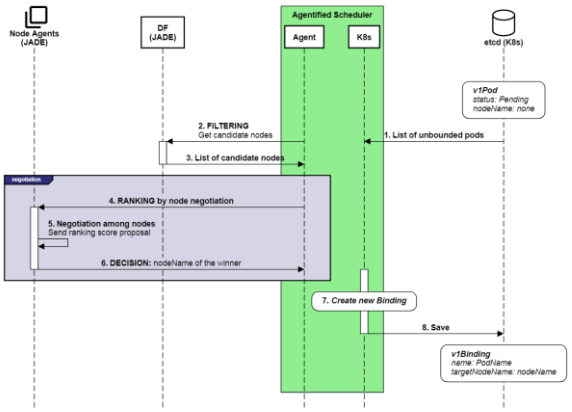

Figura 6: Diagrama de secuencia del proceso de planificación del planificador agentificado.

uso del nuevo planificador, se utilizó la etiqueta *schedulerName* en el fichero de despliegue. El código fuente del planificador agentificado está disponible en el siguiente repositorio: http://github.com/gcis-ehu/agentified-scheduler.

4 EVALUACIÓN

El planificador agentificado fue sometido a una serie de pruebas para comparar su rendimiento frente al del planificador centralizado. Esta sección se divide en dos partes: en la primera se describe el hardware y el software del demostrador utilizado para realizar las pruebas; en la segunda se muestran los resultados de un análisis diferencial entre ambos planificadores.

4.1 DEMOSTRADOR

La Figura 8 muestra el hardware sobre el que se despliega el clúster K8s. Este hardware está formado por cuatro nodos de procesamiento (Raspberry Pi 3 Model B con una tarjeta SD de 16GB) conectados a una red Ethernet en la que un hub de cinco puertos actúa de punto de interconexión (el quinto puerto proporciona salida fuera del clúster a los nodos). Las Raspberries están conectadas a dos conmutadores USB y HDMI que permiten al usuario operar el clúster en local a través de un teclado inalámbrico y una pantalla de siete pulgadas.

Todas las Raspberries utilizan la misma imagen de sistema operativo: Raspbian Stretch 2018-11-13. El procedimiento y el script de instalación utilizados para desplegar el cluster K8s en las Raspberries se encuentra disponible en el siguiente repositorio: http://github.com/gcis-ehu/agentified-scheduler. La red superpuesta utilizada fue Flannel. Por simplicidad, se optó por utilizar el servicio Google Container Registry para almacenar y distribuir las imágenes Docker, lo cual requirió la instalación de Google Cloud SDK en las Raspberries.

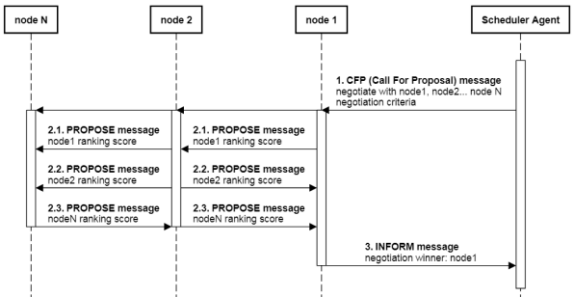

Figura 7: Diagrama de secuencia del proceso de negociación entre nodos de procesamiento.

Para compilar y ejecutar los programas en Java que implementan los planificadores que interactuan con K8s y JADE, se instaló la última version de OpenJDK, se crearon las librerías de cliente para K8s a partir de su código fuente y se descargó librería de JADE.

4.2 ANÁLISIS DIFERENCIAL DEL TIEMPO DE PLANIFICACIÓN: CENTRALIZADO VS DISTRIBUIDO

Se analizó el tiempo que tarda el planificador agentificado a la hora planificar un *pod*, esto es: el tiempo que transcurre desde que se detecta que existe un *pod* pendiente de planificar, hasta que se registra el vínculo pod-nodo en la etcd. Con el objetivo de tener un referente contra el que comparar estos valores, se analizó también el tiempo que tarda un planificador centralizado en realizar la misma operación. El criterio de selección de nodo utilizado fue la memoria asignable.

La prueba consistió en 20 repeticiones de despliegue de 1, 10, 50 y 100 *pods* en ambos tipos de planificadores. Los *pods* ejecutan una imagen de un programa "Hello World" escrito en Java y encapsulado en un contenedor Docker. La Tabla 1 muestra la tendencia (media, M) y la dispersión

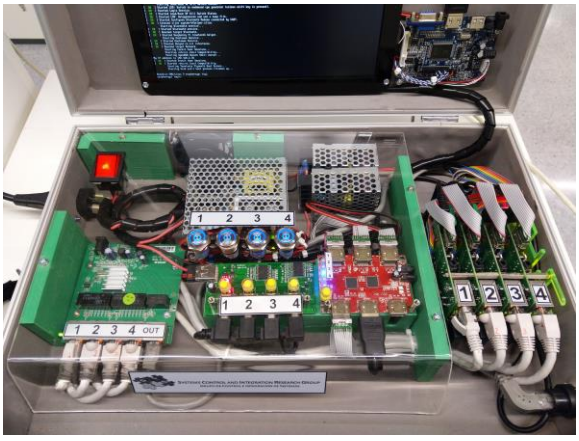

Figura 8: Demostrador utilizado para las pruebas.

Tabla 1: Estadísticos descriptivos y análisis diferencial del tiempo de planificación (en ms) de 20 despliegues consecutivos de 1, 10, 50 y 100 pods.

N pods		Centralized scheduler		Agentified scheduler		t
		M	*SD*	*M*	*SD*	
1		563	37.4	455	27.8	10.3***
10	1st pod	547	25.1	443	36.2	10.5***
	9 replicas	69.7	8.34	63.7	7.97	2.89**
50	1st pod	583	46.4	467	39.4	9.66***
	49 replicas	84.6	9.19	79.6	8.92	0.72
100	1st pod	647	256	415	42.1	3.98***
	99 replicas	80.8	8.99	77.3	8.79	1.17

**p < .01, ***p < .001.

(desviación estándar, SD) del tiempo de planificación (en milisegundos) de ambos planificadores. Los resultados muestran que el tiempo de planificación que toma el planificador distribuido es menor que el del planificador centralizado, especialmente para el primer *pod*. Sin embargo, la diferencia se reduce significativamente cuando se planifican las réplicas.

Resulta de interés analizar si las diferencias observadas entre los planificadores en las pruebas realizadas reflejan una diferencia real independiente de la muestra. La Tabla 1 muestra los resultados de una prueba de significancia estadística (t-test de una cola, t) que revela una diferencia significativa entre el tiempo de planificación de ambos planificadores cuando se planifica 1 *pod*. Sin embargo, dicho test no reveló una diferencia significativa en la planificación de las réplicas a medida que el número de replicas aumenta.

5 CONCLUSIONES

Este documento presenta el concepto de Fog-in-the-Loop (FIL) con el que se pretende contribuir a alcanzar los requisitos de la fábrica del futuro. El objetivo de FIL no es sustituir los sistemas de control a nivel de campo y planta, sino complementarlos con un circuito de control de nivel superior en la niebla que se alimenta de los datos de la planta de producción, los procesa y actúa sobre la planta en el caso de que se detecte una situción no deseada.

Como parte de la plataforma de aplicaciones necesaria para desplegar y administrar las aplicaciones de FIL, este documento presenta un planificador personalizado para K8s que transfiere la decisión de planificación a los nodos de procesamiento. La distribución de la planificación se logra delegando la operación de filtrado en una plataforma MAS y la operación de clasificación mediante negociación en los agentes situados en los nodos candidatos. Este nuevo planificador recibe el nombre de planificador agentificado.

Los tiempos de planificación del planificador agentificado (distribuido) se analizaron comparándolos con los del planificador por defecto de K8s (centralizado). Aunque los tiempos de planificación del planificador distribuido fueron inferiores a los del planificador centralizado, los resultados no pueden ser generalizables más allá de la muestra de prueba.

El trabajo futuro incluye pruebas para escalar a) el número de nodos y b) los criterios de selección de nodos para evaluar la validez del planificador agentificado como planificador para aplicaciones FIL. Además, también debe explorarse la integración de los agentes y los *kubelets* en los nodos de procesamiento.

Agradecimientos

Este trabajo has sido financiado por MCIU/AEI/FEDER, UE (proyecto RTI2018-096116-B-I00), UPV/EHU (proyecto PES17/48) y GV/EJ (proyectos IT1324-19 y KK-2018/00104).

Los autores desean agradecer a César Pérez su ayuda en el montaje del demostrador.

English summary

MULTI-AGENT SYSTEM BASED DECENTRALIZED SCHEDULING FOR FOG ORCHESTRATORS

Abstract

Fog computing is a paradigm that takes advantage of the fast response times of moving the computing infrastructure closer to the devices that collect the data, and the storage and processing features of the cloud. Fog computing can be used to improve the controllability of automation processes by introducing a higher-level control loop: Fog-in-the-loop (FIL). FIL allows capturing data from the plant, processing it to extract value-added information and feedback actions to the plant. Therefore, FIL applications are context-aware applications that require the deployment of distributed components and dynamic reconfiguration. This paper describes a custom scheduler for Kubernetes (K8s) orchestrator that distributes the scheduling task among the processing nodes by means of a multi-agent system. This new scheduling approach proved to be faster

than the centralized scheduling approach used by the default scheduler of K8s.

Keywords: Scheduler, Orchestrator, Kubernetes, K8s, Multi-Agent Systems, MAS, JADE, Fog-in-the-Loop.

Referencias

[1] S. Wang, J. Wan, D. Li and C. Zhang, "Implementing smart factory of industrie 4.0: an outlook," International Journal of Distributed Sensor Networks, vol. 12, no. 1, 3159805, 2016.

[2] F. Longo, L. Nicoletti and A. Padovano, "Smart operators in Industry 4.0: a human centered approach to enhance operators' capabilities and competencies within the new smart factory context," Computers & Industrial Engineering, vol. 113, pp. 144–159, 2017.

[3] K. Qayumi, "Multi-agent based intelligence generation from very large datasets," in 2015 IEEE International Conference on Cloud Engineering, IC2E 2015, Tempe, AZ, USA, March 9-13, 2015. Los Alamitos: IEEE Computer Society, 2015. pp 502-504.

[4] J.L. Fiadeiro and A. Lopes, "An interface theory for service-oriented design," Theoretical Computer Science, vol. 503, no. 9, pp.1–30, 2013.

[5] M. Wooldridge and N.R. Jennings, "Intelligent agents: theory and practice," The Knowledge Engineering Review, vol. 10, no. 2, pp.115–152, 2009.

[6] Weiss, G., Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA: MIT Press, 1999.

[7] W. Shi and S. Dustdar, "The promise of edge computing," Computer, vol. 49, no. 5, pp. 78-81, 2016.

[8] A. Rahmani, P. Liljeberg, J.S. Preden and A. Jantsch, Fog Computing in the Internet of Things: Intelligence at the Edge. Berlin: Springer, 2017.

[9] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan and K.K. Leung, "Dynamic service migration in mobile edge-clouds," in 2015 IFIP Networking Conference, IFIP 2015, Toulouse, France, May 20-22, 2015. Los Alamitos: IEEE Computer Society, 2015. pp 502-504.

[10] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren and C. Mahmoudi, "Fog Computing Conceptual Model," National Institute of Standards and Technology, Spec. Publ. 500-325, 2018.

[11] M. Engelsberger and T. Greiner, "Self-organizing service structures for cyberphysical control models with applications in dynamic fabric automation - A Fog/edge-based solution pattern towards service-oriented process automation," in Proceedings of the 7th International Conference on Cloud Computing and Services Science, CLOSER 2017. Sétubal: SciTePress, 2017. pp. 238-246.

[12] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski and P. Leitner, "Optimized IoT service placement in the Fog," Service Oriented Computing and Applications, vol. 11, no. 4, pp. 427-443, 2017.

[13] D. Wu, S. Liu, L. Zhang, J. Terpenny, R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," Journal of Manufacturing Systems, vol. 43, pp. 25–34, 2017.

[14] A. Orive, A. Agirre, J. Bilbao and M. Marcos, "Passive Network State Monitoring for Dynamic Resource Management in Industry 4.0 Fog Architectures," in 2018 IEEE 14th International Conference on Automation Science and Engineering, CASE 2018, Munich, Germany, August 20-24, 2018. Los Alamitos: IEEE Computer Society, 2018. pp 1414-1419.