Received: 00 Month 0000 | Revised: 00 Month 0000 | Accepted: 00 Month 0000

DOI: xxx/xxxx

ARTICLE TYPE

Multimethod optimization in the Cloud: a case-study in Systems Biology modelling

Patricia González*[1] | David R. Penas[2] | Xoan C. Pardo[1] | Julio R. Banga[3] | Ramón Doallo[1]

[1]Computer Architecture Group, Universidade da Coruña, A Coruña, Spain
[2]MODESTYA research group, Department of Statistics, Mathematical Analysis and Optimization, and Institute of Mathematics, IMAT, Universidade de Santiago de Compostela, Santiago de Compostela, Spain
[3]BioProcess Engineering Group, IIM-CSIC, Vigo, Spain

Correspondence
*Patricia González, Email: patricia.gonzalez@udc.es

Summary

Optimization problems appear in many different applications in science and engineering. A large number of different algorithms have been proposed for solving them, however, there is no unique general optimization method that performs efficiently across a diverse set of problems. Thus, multimethod optimization, in which different algorithms cooperate to outperform the results obtained by any of them in isolation, is a very appealing alternative. Besides, as real-life optimization problems are becoming more and more challenging, the use of HPC techniques to implement these algorithms represents an effective strategy to speed up the time-to-solution. In addition, a parallel multimethod approach can benefit from the effortless access to large number of distributed resources facilitated by cloud computing.

In this paper we propose a self-adaptive cooperative parallel multimethod for global optimization. This proposal aims to perform a thorough exploration of the solution space by means of multiple concurrent executions of a broad range of search strategies. For its evaluation we consider an extremely challenging case-study from the field of computational systems biology. We also assess the performance of the proposal on a public cloud, demonstrating both the potential of the multimethod approach and the opportunity that the cloud provides for these problems.

KEYWORDS:
Parallel Metaheuristics; Multimethod Optimization; Cloud Computing; Hybrid Programming; Microsoft Azure

1 | INTRODUCTION

Many optimization problems in diverse areas of science and engineering are highly constrained and based on models which exhibit complex nonlinear dynamics. These properties often result in complex multi-modal optimization problems that require for efficient and reliable global optimization (GO) solvers. Since the computational effort required by deterministic GO methods to ensure global optimality is extremely large for realistic applications, stochastic GO methods (and, in particular, metaheuristics), have become the *de-facto* choice for complex problems. Although metaheuristics can usually locate the vicinity of global solutions in reasonable computation times for small problems, in most realistic applications a large computational effort is still required in order to reach an acceptable result. Therefore, applying high performance computing (HPC) techniques to implement parallel metaheuristics represents an effective strategy to speed up the time-to-solution.

Metaheuristics as algorithms may have limited parallelism. Nevertheless, metaheuristics as problem solving methods offer opportunities for large-scale parallel computing. A metaheuristic algorithm started from different initial solutions will explore different regions of the solution space and return different solutions. Different configuration settings using the same metaheuristic algorithm will certainly give different results. Moreover, there will also be large variations in performance over different instances of the same problem. On top of that, it is not easy to know in advance which of the numerous existing metaheuristics will be the most suitable for solving a given problem. In this paper we explore the use of HPC

techniques in the context of multimethod global optimization, in which multiple different global search algorithms are executed concurrently and cooperate through information exchange. This work aims to be a *proof-of-concept* to show that, if we can devote a significant amount of resources to the search, an adaptive multimethod would achieve a more effective exploration of the search space. It could be compared to a Swiss Army knife, which contains different tools to solve different types of problems. Since we often do not know in advance the most suitable algorithm for a given optimization problem, the adaptive multimethod is able to try a number of methods and select the appropriate one. Moreover, it would be able to use different algorithms in different search spaces and dynamically adapt the search according to the results obtained during the execution.

The multimethod approach could be applied to any global optimization problem dealing with models of arbitrary complexity, including nonlinear dynamic models, which is a class of problems relevant to many areas of science and engineering. In this paper we focus on parameter estimation (i.e. model calibration) problems in systems biology as a case-study. The aim of systems biology is to generate new knowledge about complex biological systems by combining experimental data with mathematical modeling and advanced computational techniques. Dynamic modeling (typically described by sets of non-linear ordinary differential equations) is the most common formalism in this area. Model calibration consists in finding the parameters of a dynamic model that give the best fit to a set of time-series experimental data, which is formulated as minimizing a cost function that measures the goodness of this fit. This class of problems is a key research topic in current computational biology as they are extremely challenging due to its ill-conditioned and multimodal nature (1). The mathematical statement is a non linear programming (NLP) problem with differential algebraic constraints (DAEs). However, it should be noted that the multimethod described in this paper is applicable to other model classes, provided that the metaheuristics included are suitable for the problem at hand. To efficiently solve this calibration problem many research efforts have focused on developing metaheuristic global optimization methods, which combine mechanisms for exploring the search space and exploiting previous obtained knowledge to find good solutions in reasonable computation times (2).

Considering the demanding and dynamic nature of the HPC implementation of these methods, they are an important representative of the many applications that can benefit from the ability of the cloud to rapidly provision virtual clusters for particular workloads. Cloud computing represents a powerful approach to the provision and management of computing resources (3). Clouds are built on virtualized resources that public providers offer to external users by way of on-demand self-service interfaces charging them using a *pay-as-you-go* model (4, 5, 6). Besides, they have some additional features that are very appealing. The combination of scalability and elasticity provides a new capability that did not exist in traditional HPC data centers. Multi-tenancy and isolation allow each user having their own virtualized resources such as compute, networking and storage, while sharing the computing infrastructure. On-demand self-service interfaces provide users with zero queue time and system architectures configured to their applications' needs. And additionally, users have not only the ability to control software environments, but also to control access to collaborators and/or to share environments through virtual machine images, which is a critical challenge faced by the scientific community today (7). So, cloud computing is nowadays one of the most interesting means to obtain massive computing resources for scientific applications.

In conclusion, in this paper we explore the use of multimethod optimization to parameter estimation problems in systems biology, which are representative of a class of challenging global optimization problems that are common in many areas of science and engineering. We also assess its performance on the Azure public cloud and provide some guidance on how this type of applications could benefit from the use of cloud computing.

The structure of the paper is as follows. Section 2 discusses related work. Section 3 describes the proposed self-adaptive cooperative multimethod. Section 4 describes the experiments carried out and discusses on the obtained results. Finally, Section 5 summarizes the conclusions of the paper.

2 | RELATED WORK

The concept of multimethod algorithms has appeared in different fields, including parallel cooperative search methods (8), memetic algorithms (9), algorithm ensembles (10), algorithm portfolios (11), and hyperheuristics (12). One of the first examples can be found in (13), where a self-adaptive Differential Evolution (DE) algorithm makes use of DE learning strategies that are weighted based on the algorithm success. Also, the heterogeneous cooperative algorithm presented in (14) employs different evolutionary algorithms to update the subpopulations in a cooperative algorithm framework. Another popular example is AMALGAM-SO (15), a population-based genetic adaptive method for single objective optimization that updates the allocation of algorithms during the optimization run. A multimethod hyperheuristic algorithm that makes use of a number of common metaheuristics was presented in (16), obtaining promising results in terms of solution quality and algorithm robustness.

With the recent developments in technology the use of parallel computing is becoming increasingly popular. Additionally, the cost/performance ratio of HPC architectures is constantly decreasing, and, moreover, with the appearance of cloud computing attaining access to a large number of distributed resources is now more feasible. The parallelization of metaheuristics has received much attention to reduce the time for solving large-scale problems. Many parallel algorithms have been proposed in the literature. A nice review can be found in (17). Most of these proposals are parallel implementations based on traditional parallel programming interfaces, such as MPI or OpenMP, executed in traditional parallel infrastructures, such as local clusters or supercomputers.

Research on cloud-oriented parallel metaheuristics based mainly on the use of MapReduce (18) and Spark (19) has also received increasing attention in recent years (20, 21, 22, 23, 24, 25, 26). The experimental results reveal that the extra cost of the I/O operations and the system bookkeeping overhead significantly reduces the performance of the parallelization using MapReduce in iterative algorithms. This performance can be improved by an order of magnitude when using Spark (19, 27).

In the last decade, several researchers have also paid attention to the performance of MPI applications in the cloud. Most of these studies use classical MPI benchmarks to compare the performance of MPI on public cloud platforms (28, 29, 30). Besides, also real applications have been assessed in the cloud, such as bioinformatics applications (31), high-energy and nuclear physics experiments (32), and different e-Science applications (33, 34). Also, an extensive analysis to detect the more critical issues and bottlenecks of HPC applications in the cloud has been carried out in (35). These works conclude that clouds were not designed for running tightly-coupled HPC workloads, like MPI applications. Overall, the lack of high-bandwidth, low-latency networks, as well as the virtualization overhead, has a large effect on the performance of such applications in the cloud.

In a previous work (36) we have explored the use of MPI and Spark in the parallel implementation of the DE algorithm. This implementation was based on the island model, that drastically reduces the inter-process communications leading to a loosely-coupled parallel application, thus, more suitable for cloud environments. We discussed about the differences that arise from the inherent features of each programming model, and we assessed the performance of both implementations in different computing platforms, including the Microsoft Azure public cloud. We concluded that from a computational point of view the MPI implementation outperforms the Spark one.

In this paper we propose a new self-adaptive cooperative multimethod algorithm, implemented using MPI and OpenMP, and assess its performance in a public cloud provider. We believe that multimethods are a class of algorithms that can take benefit from the capabilities of cloud computing.

3 | SELF-ADAPTIVE COOPERATIVE MULTIMETHOD

In the majority of the most popular metaheuristics, the search of new solutions depends on previous iterations of the algorithm, which not only complicates the parallelization itself but also limits the speedup and scalability of the solution. The time-consuming operations are located in inner loops which can be easily performed in parallel. However, since the external loops of the algorithm usually present dependencies between different iterations, a fine-grained parallelization in the inner loops would limit the scalability of the solution in distributed systems. A more effective approach would be a coarse-grained parallelization that will imply finding a parallel variant of the sequential algorithm. Using an island-model implementation is a popular approach. The population is divided into subsets (*islands*) where the metaheuristic steps are run isolated and then sparse individual exchanges are performed among islands. This solution drastically reduces the communications between distributed processes. But, its scalability is again heavily restrained by the size of the population matrix. Reducing the population matrix by dividing it between different islands may have a negative impact on the convergence of each individual island. Thus, based on the ideas outlined in (37), here we propose an island-based method where each island performs a different metaheuristic algorithm with different initial populations and different configuration parameters, while they cooperate modifying the systemic properties of the individual searches.

Current HPC systems, that usually comprise clusters of multicore nodes, can benefit from the use of a hybrid programming model in which a message passing library, such as MPI, is used for the inter-node communications while a shared memory programming model, such as OpenMP, is used at intra-node level. The hybrid model provides several advantages such as reducing the communication needs and memory consumption, as well as improving load balancing and numerical convergence (38). These features are particularly appealing for HPC applications running in the cloud. However, due to the overhead of the network virtualization, loosely-coupled MPI processes, that is, where inter-node communications are reduced to a minimum, are desirable. The solution proposed in this work pursues the development of an efficient cooperative multimethod, called self-adaptive cooperative multimethod (saCMM), focused on both, the acceleration of the computation through a fine-grained parallelization of the cost function evaluations using OpenMP, and the convergence improvement through a loosely-coupled coarse-grained parallelization of the search diversification and the cooperation among islands using MPI.

3.1 | Fine-grained parallelization

The evaluation of the cost function values is the most time consuming task in the metaheuristics. Besides, this task uses to appear in several steps of the algorithms. Algorithm 1 shows a basic pseudocode for performing the cost function evaluations in parallel using a shared memory paradigm. Every time an evaluation step is needed, a parallel loop is defined based on the fork-join programming model. As it can be observed, the synchronization at the end of the parallel loop may lead to a load imbalance that can cause significant delays when different evaluations have different computational loads. A dynamic schedule clause is used to assign new evaluations to threads as they complete their previous assignments.

Algorithm 1 Parallel evaluation of solutions

```
$$ parallel do (dynamic schedule, private(eval,newsol,i)
for i=1 to numSolutions do
    newsol = solutions(:,i)
    eval = f_eval(newsol)
end for
$$ end parallel do
```

3.2 | Coarse-grained parallelization

The proposed coarse-grained parallelization follows a master-slave approach. However, the master does not play the role of a globally accessible central memory, which would harm the performance in a distributed memory environment with low bandwidth or high-latency networks, such as the cloud. The data is completely distributed among the slaves (islands) that perform a different sequential metaheuristic each. The saCMM master is in charge of:

- *cooperation among islands*: the exchange of information is driven by the quality of the solutions rather than by elapsed time, thus, achieving a more effective cooperation.
- *asynchronous communication protocol*: to avoid having idle islands waiting for information exchanged from others. The asynchronous communication protocol has also a great impact in the performance of the proposal in the cloud, because nodes will not be affected by communication delays in the virtual network.
- *self-adaptive procedure*: changing dynamically the settings of those islands that do not cooperate with those of the most promising searches.

Algorithm 2 shows a schematic, simple pseudocode of the proposed saCMM method. By now two metaheuristics are performed by the islands, the Differential Evolution (DE) (39), implemented with the enhancements described in (40), and the enhanced Scatter Search (eSS) (41), using the implementation outlined in (37). Nevertheless, as already pointed before, this paper aims to be a proof-of-concept, and the saCMM could be easily extended including more metaheuristics.

Cooperation among islands

The proposed cooperative strategy accelerates the exploration of the search space by launching simultaneous searches, using different metaheuristics with different configurations and independent initial points, and providing cooperation mechanisms to share information among them. When a solution is considered to be *promising* in one search it is sent to the master process that is in charge of spreading it to the rest of the islands. However, the information exchange among islands should not be too frequent to avoid premature convergence to local optima (42, 43). Thus, exchanging all current best solutions is avoided to prevent the cooperation entries from filling up the islands' populations leading to a rapid decrease of the diversity. Besides, reducing the number of inter-node communications would improve the performance in the cloud. Thus, a threshold is used in the master to determine when a new best solution significantly outperforms the current best solution, deserving to be spread to the rest of islands.

The strategy used to select those members of the islands' population to be replaced with the incoming *promising* solutions, should be also carefully considered. The most popular selection/replacement policy in parallel metaheuristics is to replace the *worst* solution in the population with the new solution when its value is better. However, after a few iterations receiving new *promising* solutions, the initial population in each island will be lost and most of the entries in different islands would be the same, thus, losing diversity and potentially converging to suboptimal solutions. To avoid that, we propose to label one member of the population as a cooperative solution, so that a foreign solution can only enter the population by replacing this member. The first time a shared solution is received, the *worst* solution in the population will be replaced and labeled as the *cooperative* solution for the next iterations. The cooperative solution will be combined and improved like the rest of the entries in the population. With this replacement policy every search will evolve over its initial population still taking advantage of those cooperative solutions from other islands that might improve its own search.

Asynchronous communication protocol

An important aspect when designing the communication protocol is the interconnection topology of the different components of the parallel algorithm. A widely used topology in master-slave models, the *star* topology, is used in this proposal. The master process is in the center of the star

Algorithm 2 Pseudocode of the self-adaptive Cooperative Multimethod (saCMM).

Pseudocode for the master:

```
! Best solution initialization
bestSol = DBL_MAX
! Threshold initialization
ϵ = 0.1
nRefuse = 0
! Scoreboard initialization
slaveComm(:) = 0
slaveScore(:) = 0
! Listen to requests from slaves
repeat
    ! Non-blocking asynchronous reception message
    MPI_IRecv(buffer,...,&req)
    MPI_Wait(&req,&status)
    if req is a promising solution then
        ! Cooperation among islands
        if req < bestSol AND |(bestSol - req) / bestSol| < ϵ then
            bestSol = req
            MPI_ISend(bestSol,slaves)
            ! Score update
            slaveComm(slave)++
            slaveScore(slave) = slaveComm(slave) * Elapsed_Time()
        else
            nRefuse++
            if too many refused solutions then
                ! Adapt threshold
                ϵ = ϵ/2
                nRefuse = 0
            end if
        end if
    else if req is a reconfiguration request then
        ! Send most promising settings
        bestSettings = Sort(slaveScore(:))
        MPI_ISend(bestSettings,slave)
    else if req is a termination request then
        ! Broadcast termination request
        MPI_ISend(stop, slaves)
    end if
until stopping criteria
```

Pseudocode for the slaves:

```
! Best solution initialization
bestSol = DBL_MAX
! Prepare to listen to messages from master
MPI_IRecv(buffer,...,&req)
repeat
    ! Iterate the serial metaheuristic (DE/eSS/other) once
    stepSol = Metaheuristic_Step()
    ! Check for promising solution to cooperate
    if stepSol significantly improves bestSol then
        bestSol = stepSol
        MPI_ISend(bestSol,master)
    end if
    ! Check progress before request for reconfiguration
    if execution is not progressing then
        MPI_ISend(reconfReq,master)
    end if
    ! Check stopping criteria
    if stopping criteria is fulfilled then
        MPI_ISend(stop,master)
    end if
    ! Check for pending messages from master
    repeat
        MPI_Test(&req,&pendingMsg,&status)
        if pendingMsg then
            if req is a promising solution then
                ! Cooperation from islands
                if req <bestSol then
                    bestSol = req
                    Replace_Solution(bestSol,cooperativeEntry)
                end if
            else if req is a reconfiguration response then
                ! Change metaheuristic configuration settings
                Reconfigure_Metaheuristic(req)
            else if req is a termination request then
                Stop()
            end if
            ! Continue listening
            MPI_IRecv(buffer,...,&req)
        end if
    until NOT(pendingMsg)
until stopping criteria
```

and all the rest of the processes (slaves) exchange information through the master. The distance[1] between any two slaves is always two, therefore reducing communication delays that would harm the cooperation between processes. This topology is particularly appropriate for the execution in cloud environments, since it enables to quickly spread the solutions despite the low-bandwith and high-latency of the virtualized network.

[1]The distance between two nodes in a topology is defined by the minimum number of nodes that must be traversed to join them.

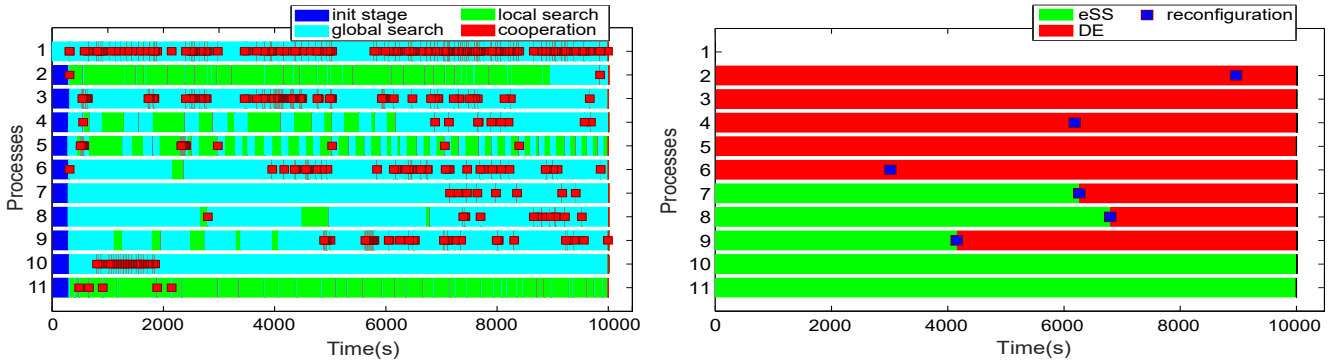

FIGURE 1 Gantt diagrams representing the processes and cooperation among them during the execution progress. Process 1 is the central master, and processes 2-11 the islands. At the left diagram, red dots represent asynchronous cooperation communications between master and islands, light blue areas represent global search steps and green areas represent local search steps. At the right diagram, red areas represent DE islands, green areas represent eSS islands and dark blue dots represent reconfiguration requests.

The communication protocol is designed to avoid processes' stalls when messages don't arrive during an external iteration, allowing the execution to keep progressing in every individual process. Both the emission and reception of messages are performed using non-blocking operations (see the use of `MPI_ISend`, `MPI_IRecv`, `MPI_Wait` and `MPI_Test` in Algorithm 2) to allow the overlapping of communications and computation. This is crucial in the success of the multimethod since the diversification introduced in the different islands would result in a computationally unbalanced scenario. In addition, the proposed protocol minimizes the impact of communication delays on the performance when executing in cloud environments.

Self-adaptive procedure

The master process controls the long-term behavior of the parallel searches and their cooperation, and dynamically changes, during the execution, several configuration parameters to improve the success of the parallel cooperative scheme.

First, the master is in charge of the threshold used to select, from the received cooperative solutions, which are qualified to be spread to the islands. With a larger threshold value, cooperation will only occur sporadically, reducing its efficiency. By the contrary, a smaller threshold value will increase the number of similar solutions being spread, affecting negatively the parallel implementation efficiency. Moreover, too much cooperation would often be counterproductive. On the one hand, it might decrease the diversity of the explored regions of the search space, leading to many islands searching in the same region and early converging to a non-optimal solution. On the other hand, it could degenerate into a tightly-coupled parallelization that would be inefficient for cloud environments. To overcome those drawbacks, an adaptive procedure is proposed that starts with a large threshold value and decreases it as the search progresses.

Second, the master is used as a scoreboard to dynamically tune the settings of the search in the different islands. As commented before, each island performs a different metaheuristic. The performance of a metaheuristic is still problem-dependent and, besides, the fine-tuning of the metaheuristic parameters have a great impact in both the search and the quality of the results. Since the exact nature of the problem at hand is unknown, a range of metaheuristics and settings that yields conservative, aggressive, and intermediate islands should be selected at the time of starting the search. Then, the proposed procedure will adaptively change the algorithm and settings in the islands during the execution, leaning toward those that exhibit the highest success, further improving the efficiency of the search.

To decide the most promising islands, the master serves as a scoreboard, ranking them according to their potential. In rating the islands, two facts have to be considered: (1) the number of total communications received by the master from each island, to identify those that cooperate more intensively with new *promising* solutions; and (2) for each island, the moment when its last solution was received, to emphasize those islands that have more recently cooperated. A more accurate scoreboard is obtained with a good balance of these two factors.

Identifying the *worst* islands is also complicated. Those islands at the bottom of the scoreboard can be there because they do not communicate sufficient solutions or because a considerable amount of time has passed since their last communication. However, they can be either non-cooperating (*less promising*) islands or more *aggressive* ones. An aggressive island often performs longer iterations because it frequently calls local solvers, and thus it is unable to communicate results as often as *conservative* islands do. Thus, each island decides by itself whether it is evolving in a promising mode or not. If an island detects that it is receiving cooperative solutions from the master but it is not improving its own results, it will send the master a reconfiguration request. The master will send back to it the settings of the island on the top of the scoreboard.

Figure 1 shows, as an example, a Gantt diagram corresponding to a real execution for one of the benchmarks (B3) used in the experimental evaluation. Five of the ten islands initiate DE with different configurations, from aggressive ones (that perform frequent local searches) to conservative

ones (that do not perform local searches or perform them only sporadically). Another five islands initiate eSS, also with different configurations. The red dots representing the cooperation between islands show that DE outperforms eSS for this benchmark, and also that conservative configurations, with few local searches, outperform aggressive islands. Thus, when the unsuccessful islands request for a reconfiguration, the master sends them the configuration of the promising islands. In this example processes 2, 4 and 6, executing DE with a poor evolution (absence of cooperations during a long time), request a reconfiguration to the master, and change from an aggressive to a conservative configuration, clearly improving their evolution (they cooperate again). Also processes 7, 8 and 9, executing eSS with an unprofitable progress, request for a reconfiguration, and they end up executing a conservative DE. Since the most promising configuration can change during the execution progress, not all the processes executing a particular method are allowed to be reconfigured. In this example, two eSS islands are not reconfigured, one conservative (process 10) and the other aggressive (process 11).

4 | EXPERIMENTAL EVALUATION

The proposed saCMM method has been evaluated with a set of benchmarks from the BioPreDyn-bench suite (44), to assess its efficiency in challenging parameter estimation problems in computational system biology:

- *Problem B1*: genome-wide kinetic model of *S. cerevisiae*. It contains 276 dynamic states, 44 observed states and 1759 parameters.
- *Problem B2*: dynamic model of the central carbon metabolism of *E. coli*. It consists of 18 dynamic states, 9 observed states and 116 estimable parameters.
- *Problem B3*: dynamic model of enzymatic and transcriptional regulation of the central carbon metabolism of *E. coli*. It contains 47 dynamic states (fully observed) and 178 parameters to be estimated.
- *Problem B4*: kinetic metabolic model of Chinese Hamster Ovary (CHO) cells, with 34 dynamic states, 13 observed states and 117 parameters.
- *Problem B5*: signal transduction logic model, with 26 dynamic states, 6 observed states and 86 parameters.

Experiments were deployed with default settings in the North Europe region of the Microsoft Azure public cloud using a virtual cluster with A9 instances. The A9 instances are for compute-intensive workloads having 16 cores each, Intel Xeon E5-2670 @2.60GHz processors, and 112GB of RAM.

The results shown in this section were analyzed both from a horizontal view (45), that is, assessing the performance by measuring the time needed to reach a given target value, and from a vertical view, that is, assessing the performance for a predefined effort. To evaluate the efficiency from a horizontal view, a stopping criteria based on a *value-to-reach (VTR)* is used. The VTR used was the optimal fitness value reported in (44). To evaluate the efficiency from a vertical view, the stopping criteria used is a predefined execution time. Thus, the experiments combine the two stopping criteria, a VTR and a maximum execution time. Due to the substantial dispersion of the results, because of the stochastic nature of these methods, each experiment was performed 20 times and a statistical study was carried out.

Different experiments have been conducted to compare the proposed saCMM multimethod versus single method performance, to assess its scalability when the number of available computational resources increases, and to determine the impact of diversification against intensification in its hybrid MPI+OpenMP implementation.

4.1 | Assessing the performance of saCMM

In this first experiment the saCMM performance has been compared with three different parallel strategies:

- an embarrassingly-parallel non-cooperative multimethod (*np*-MM). This algorithm consists of *np* independent runs (being *np* the number of available cores) performed in parallel without cooperation between them and reporting the best execution time of the *np* runs. Diversity is introduced alike in saCMM, that is, each run executes a separate metaheuristic (DE and eSS are mixed half and half) with different strategies.
- a self-adaptive cooperative strategy using only DE (saCDE). Diversity is introduced alike in saCMM but using only DE, that is, each island executes a separate DE with different configuration parameters (i.e. mutation factor and/or crossover constant).
- a self-adaptive cooperative strategy using only eSS (saCeSS). Diversity is introduced alike in saCMM but using only eSS, that is, each island executes a separate eSS with different configuration parameters (i.e. dimension of the reference set or balance parameter for the selection of initial points for the local solvers).

TABLE 1 Performance of the proposed self-adaptive multimethod compared with other parallel approaches.

	method	VTR	avg. fbest	iter.	max. time (s)	avg. exec. time (s)	%hits
	10-MM	13753	14325,67	87,05	6000	5357,09	65%
	saCDE	13753	43184,94	49,50	6000	VTR not reached	0%
B1	saCeSS	13753	13130,74	50,05	6000	3469,76	95%
	saCMM	13753	13695,37	73,20	6000	3166,17	100%
	10-MM	250	249,87	1178,85	6000	2278,40	100%
	saCDE	250	249,95	3781,20	6000	3180,58	100%
B2	saCeSS	250	249,97	780,15	6000	1383,65	100%
	saCMM	250	249,97	1534,65	6000	1908,27	100%
	10-MM	0.37	149,36	435,90	10000	VTR not reached	0%
	saCDE	0.37	37,69	545,05	10000	VTR not reached	0%
B3	saCeSS	0.37	184,88	222,95	10000	VTR not reached	0%
	saCMM	0.37	13,50	770,50	10000	VTR not reached	0%
	10-MM	55	49,03	211,00	6000	339,08	100%
	saCDE	55	47,76	189,35	6000	255,71	100%
B4	saCeSS	55	49,48	210,60	6000	676,66	100%
	saCMM	55	46,69	117,10	6000	228,62	100%
	10-MM	4200	4155,46	40,50	6000	995,85	100%
	saCDE	4200	4151,14	63,40	6000	1095,92	100%
B5	saCeSS	4200	4171,15	13,25	6000	735,03	100%
	saCMM	4200	4152,31	38,60	6000	961,08	100%

Table 1 shows, for each benchmark and each method, the VTR and the maximum time (*max. time*) used as stopping criteria, the average value of the best value obtained (*avg. fbest*), the average number of iterations performed in each experiment (*iter.*), the final execution time when the VTR is reached (*avg. exec. time*), and the percentage of executions that achieve the VTR before the maximum time allowed (*%hits*). For these experiment 10 single-threaded islands were used. Using different benchmarks allows for comprehensive comparison of the results. It can be observed that different methods perform different for different benchmarks. For instance, DE performs very poorly for B1 but very well for B4. The same happens with eSS that performs very well for B2 but quite bad for B3 (note the best value achieved). As explained before, the self-adaptive cooperative multimethod is a very attractive approach since it avoids the need of selecting the method to use in advance. In general, results in Table1 demonstrate that the proposed self-adaptive multimethod outperforms the rest of the approaches. For all the benchmarks, saCMM outperforms 10-MM and either outperforms both saCDE and saCeSS or performs closely to the best suited for the benchmark at hand. For problem B1, saCMM is the only method that achieves a 100% of successful executions, in opposite to saCDE that never achieves the VTR and 10-MM that achieves a 65%. For benchmark B3, a very challenging problem, the maximum execution time used was not enough to reach the VTR thus, taking a look at the best value obtained (*avg. fbest*), saCMM clearly outperforms the rest of the approaches, obtaining a very promising result. For benchmark B5, the difference between saCMM and the rest is not representative because the number of iterations needed to achieve convergence is too small. This has two main consequences. On the one hand, the cooperation among islands is done at the end of each iteration so, in this benchmark, islands have few opportunities to cooperate. On the other hand, for an island to be reconfigured, it has to identify if it is evolving promisingly and this takes several iterations, therefore in benchmark B5 there are less opportunities to reconfigure the islands. The saCeSS method beats the rest in B5, as well as in B2, simply because all the islands perform eSS, which is the best suited for these problems, while saCMM performs eSS in only half of the islands.

Since the results reported in Table 1 hide the underlying distribution, that in this kind of stochastic problems is very important, in Figure 2 the distribution of the results is shown combining boxplots with violinplots. The figure illustrates that saCMM reduces the variability of the results, both when assessing the performance from an horizontal view and from a vertical view. The violin/boxplot for benchmark B3 shows the results for the best value achieved from a vertical view, that is, when all the runs stop by a predefined maximum time. As it can be seen, DE performs better than eSS for this problem, but still having a large variability. However, saCMM presents considerable less variability, being an important feature in the performance of this solver. The violin/boxplot for benchmark B4 shows the results for the execution time from a horizontal view, that is, when all the executions reach the VTR. Again saCMM achieves an important reduction in the variability of the results, most of them are placed in the lower part of the violinplot. It must be noted that this problem presents, from its roots, a very large dispersion in the results (e.g. from 24s to 1800s in the non cooperative 10-MM multimethod).

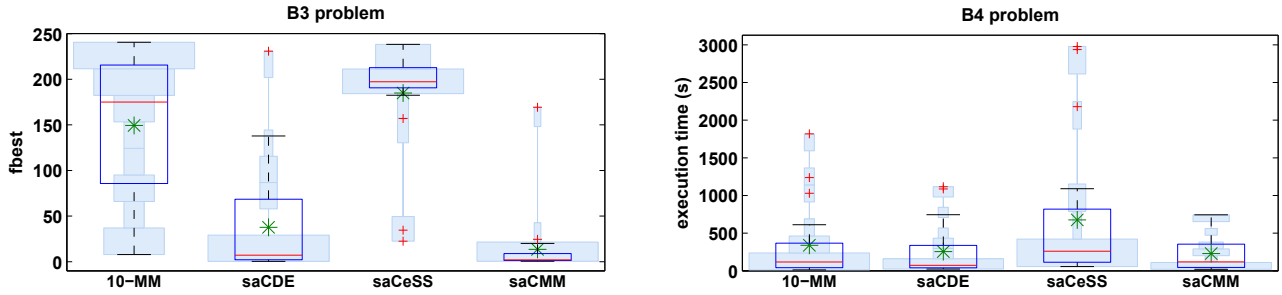

FIGURE 2 Violin/boxplots for experiments B3 and B4 reported in Table 1 . The green asterisks correspond to the mean and light blue boxes show the distribution of the results.

TABLE 2 Impact in saCMM of the diversification by increasing the number of islands.

	#Islands	VTR	avg. fbest	iter.	max. time (s)	avg. exec. time (s)	%hits
B1	10	13753	13695,37	73,20	6000	3166,17	100%
	20	13753	13657,58	36,40	6000	3154,23	100%
	30	13753	13681,14	38,95	6000	2499,97	100%
	40	13753	13644,21	30,65	6000	2302,63	100%
B2	10	250	249,97	1534,65	6000	1908,27	100%
	20	250	249,97	753,00	6000	1750,82	100%
	30	250	249,93	717,65	6000	1437,96	100%
	40	250	249,92	593,65	6000	1422,67	100%
B3	10	0.37	13,50	770,50	10000	VTR not reached	0%
	20	0.37	2,37	497,10	10000	8411.98	5%
	30	0.37	1,03	427,20	10000	7360,29	20%
	40	0.37	0,87	406,75	10000	6396,10	20%
B4	10	55	46,69	117,10	6000	228,62	100%
	20	55	32,55	22.70	6000	68,41	100%
	30	55	45,32	24,36	6000	48,36	100%
	40	55	41,89	16,30	6000	38,87	100%
B5	10	4200	4152,31	38,60	6000	961,08	100%
	20	4200	4157,22	34,05	6000	939,73	100%
	30	4200	4167,34	26,50	6000	711,34	100%
	40	4200	4136,84	29,20	6000	770,58	100%

4.2 | Impact of island diversification

Table 2 shows for saCMM the impact in the search of increasing the diversity by increasing the number of islands. Different experiments were performed using 10, 20, 30 and 40 islands to assess the scalability of the proposal. Initially, for all the experiments, half of the islands perform DE and the other half eSS with different initial populations and configuration parameters. As can be seen, an increase in the number of islands impacts positively the average execution time for all the experiments. Note also the positive effect on benchmark B3, that reaches now 20% of hits. However, the scalability begins to stagnate when we go on increasing the number of islands (above 30 islands for most of these benchmarks).

Specially significant is the impact of the island diversification in benchmarks B3 and B4. Benchmark B3 is a very challenging problem that does not converge in the maximum time allowed when using 10 islands. But when using 20 islands, 5% of the executions converge, and that percentage rises to 20% for 30 and 40 islands. Note also the improvement in the average best function value and in the average execution time of runs that converge. As already commented, benchmark B4 presents a very large dispersion of results in the sequential case. Thus, an increase in the diversity

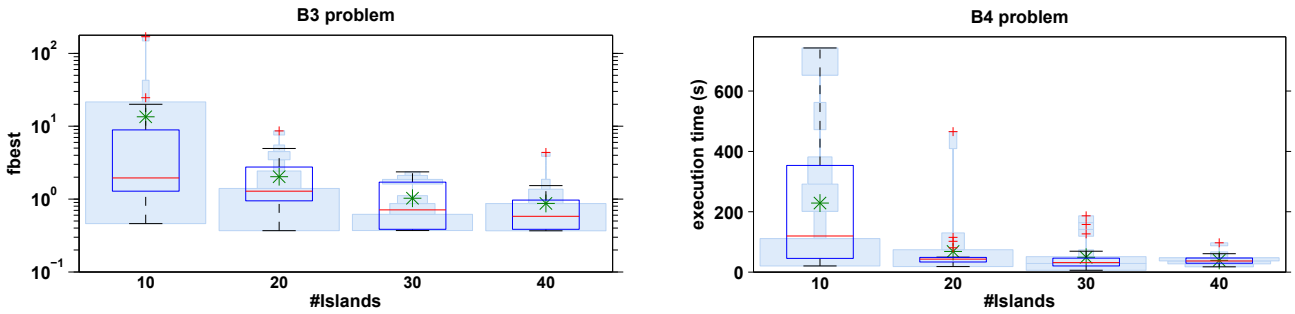

FIGURE 3 Violin/boxplots for experiments B3 and B4 reported in Table 2 .

TABLE 3 Impact in saCMM of the fine-grained parallelization within each island.

	#Islands x #threads	VTR	avg. fbest	iter.	max. time (s)	avg. exec. time (s)	%hits
B1	10×1	13753	13695,37	73,20	6000	3166,17	100%
	10×2	13753	13675,55	66,30	6000	1607,18	100%
	10×4	13753	13714,55	76,35	6000	1132,28	100%
B2	10×1	250	249,97	1534,65	6000	1908,27	100%
	10×2	250	249.97	1043,70	6000	826,41	100%
	10×4	250	249,93	873,40	6000	523,49	100%
B3	10×1	0.37	13,50	770,50	10000	VTR not reached	0%
	10×2	0.37	0,60	1526,53	10000	8535,63	40%
	10×4	0.37	0,54	2344,36	10000	6194,94	40%
B4	10×1	55	46,69	117,10	6000	228,62	100%
	10×2	55	48,02	188,50	6000	179,37	100%
	10×4	55	50,29	378,95	6000	208,82	100%
B5	10×1	4200	4152,31	38,60	6000	961,08	100%
	10×2	4200	4167,57	33,75	6000	500,28	100%
	10×4	4200	3972,16	37,55	6000	350,32	100%

is particularly convenient for this problem. Figure 3 shows violin/boxplots for the results of B3 and B4 in Table 2 , illustrating how saCMM reduces the variability of the results when more diversity is introduced by increasing the number of islands.

4.3 | Impact of island intensification

We have also evaluated the impact in saCMM of intensifying the search within each island by modifying the fine-grained parallelization using OpenMP, as explained in Section 3. Table 3 shows the results when 10 islands are used and the number of threads ranges from 1 to 4. As it can be seen, for the same number of islands increasing the number of threads in each island improves the execution time, because more evaluations of the benchmarking function are done in parallel without modifying the macroscopic behavior of the algorithm. However, as already commented in Section 3, the scalability of the fine-grained parallelization is limited. For instance, the speed-up obtained in the average execution time of the 10×2 configuration versus the 10×1 configuration is almost 2, while it hardly reaches 3 when using 10×4 configuration.

From the results of Table 2 and Table 3 we can conclude that, for a fixed number of resources, a compromise between diversification and intensification is necessary to obtain the best results. The most suitable configuration will be dependent on the problem at hand. Table 4 shows results for all the benchmarks using 40 cores in different configurations of islands and threads. In general, for these problems, only B1 benchmark benefits from using more than four threads, because these problems prefer intensification to diversification. In contrast, B4 is strongly benefited by diversity, obtaining better results if all the resources are used for running different islands instead of intensifying each individual search.

TABLE 4 Impact in saCMM of the configuration with a fixed number of resources.

	#Islands x #threads	VTR	avg. fbest	iter.	max. time (s)	avg. exec. time (s)	%hits
B1	40×1	13753	13644,21	30,65	6000	2302,63	100%
	20×2	13753	13658,44	31,30	6000	1737,25	100%
	10×4	13753	13714,55	76,35	6000	1132,28	100%
	5×8	13753	13630,28	36,00	6000	1089,27	100%
B2	40×1	250	249,92	593,65	6000	1.422,67	100%
	20×2	250	249,92	718,15	6000	887,43	100%
	10×4	250	249,93	873,40	6000	523,49	100%
	5×8	250	249,90	1357,35	6000	981,78	100%
B3	40×1	0.37	0,87	406,75	10000	6396,10	20%
	20×2	0.37	0,90	785,05	10000	7609,36	30%
	10×4	0.37	0,54	1209,00	10000	6194,94	40%
	5×8	0.37	0,50	2844,35	10000	6909,07	45%
B4	40×1	55	41,89	38,87	6000	38,87	100%
	20×2	55	45,17	57,75	6000	70,71	100%
	10×4	55	50,29	378,95	6000	208,82	100%
	5×8	55	50,61	1053,80	6000	555,45	100%
B5	40×1	4200	4.136,84	29,20	6000	770,58	100%
	20×2	4200	4128,06	32,10	6000	531,62	100%
	10×4	4200	3972,16	37,55	6000	350,32	100%
	5×8	4200	4143,15	38,05	6000	369,91	100%

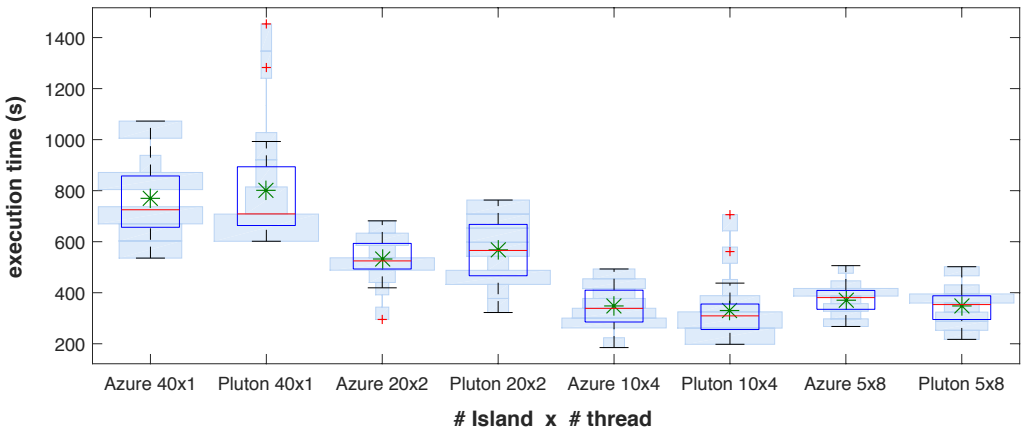

FIGURE 4 Violin/boxplots for experiment B5 in the local cluster Pluton and in the Azure virtual cluster.

4.4 | Impact of executing using public cloud resources

Finally, to assess the impact that the use of virtualized resources (virtual machines, network and storage) in the cloud have on the performance of saCMM, the same previous experiments were performed in an HPC local cluster, named Pluton. Pluton consists of 16 nodes powered by two octa-core Intel Xeon E5-2660 @2.20GHz with 64GB of RAM and interconnected with an InfiniBand FDR network. The distribution of the results for benchmark B5 are shown in the violin/bloxplots of Figure 4 . It can be seen that execution times in the Azure cluster not only are competitive with those obtained in Pluton, but also outperforms them. It is important to note that the dispersion of the results is also smaller in the Azure cluster, because computation is faster in the A9 instances due to their hardware features. As a consequence, the dissemination of promising results among islands and the reconfiguration mechanism further improve the behavior of the method. These results probe that the proposed loosely-coupled implementation is suitable to be used in cloud environments.

5 | CONCLUSIONS AND FUTURE WORK

In this paper, we propose a self-adaptive cooperative multimethod (saCMM) for global optimization. We evaluate its performance in a public cloud using challenging problems from computational systems biology. Our results reveal that saCMM is a competitive solution approach, outperforming other competitive methods based on traditional single-method and non-cooperative schemes. Our results also show that the class of problems considered would benefit from the scalable provision of computational resources, giving support to multiple concurrent searches looking for a compromise between diversification and intensification.

In the light of these results, we think that extending the proposal to allow for dynamic balancing during the search between diversity (number of islands) and intensity (number of threads per island) taking into account changes in the available resources would be of special interest. However, this is something that cannot be done using the hybrid MPI+OpenMP approach. Therefore, as future work, we will focus on the implementation of an enhanced saCMM using a cloud framework like Spark. With this new approach we expect saCMM to take further advantage of cloud capabilities like elasticity and scalability.

The source code of saCMM is publicly available at `https://bitbucket.org/DavidPenas/sacmm-library`.

ACKNOWLEDGMENTS

This research received financial support from the Spanish Government through the projects DPI2014-55276-C5-2-R and TIN2016-75845-P (AEI/FEDER, UE), and from the Galician Government under the Consolidation Program of Competitive Research Units (Network Ref. R2016/045 and Project Ref. ED431C 2017/04), all of them co-funded by FEDER funds of the EU. We also acknowledge Microsoft Research for being awarded with a sponsored Azure account.

References

[1] Villaverde A.F., Banga J.R.. Reverse engineering and identification in systems biology: Strategies, perspectives and challenges. *Journal of the Royal Society Interface*. 2014;11(91):art. no. 20130505.

[2] Banga Julio R. Optimization in computational systems biology. *BMC systems biology*. 2008;2(1):47.

[3] Reed Daniel A, Dongarra Jack. Exascale computing and big data. *Communications of the ACM*. 2015;58(7):56–68.

[4] Armbrust Michael, Fox Armando, Griffith Rean, et al. A view of cloud computing. *Communications of the ACM*. 2010;53(4):50–58.

[5] Buyya Rajkumar, Broberg James, Goscinski Andrzej M. *Cloud computing: Principles and paradigms*. John Wiley & Sons; 2010.

[6] Rodero I, Parashar M, Quiroz A, Guim F, Poole S W. Energy-efficient Online Provisioning for HPC Workloads. In: Ahmad I, Ranka S, eds. *Handbook of Energy-Aware and Green Computing-Two Volume Set*, Chapman and Hall/CRC 2012 (pp. 795–816).

[7] Jackson Keith R, Ramakrishnan Lavanya, Muriki Krishna, et al. Performance analysis of high performance computing applications on the amazon web services cloud. In: The 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom):159–168IEEE; 2010.

[8] Crainic Teodor Gabriel. *Parallel Meta-Heuristic and Cooperative Search*. technical report: CIRRELT-2017-58, Universite du Quebec a Montreal; 2017.

[9] Chen Xianshun, Ong Yew-Soon, Lim Meng-Hiot, Tan Kay Chen. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*. 2011;15(5):591–607.

[10] Yang Pengyi, Hwa Yang Yee, B Zhou Bing, Y Zomaya Albert. A review of ensemble methods in bioinformatics. *Current Bioinformatics*. 2010;5(4):296–308.

[11] Peng Fei, Tang Ke, Chen Guoliang, Yao Xin. Population-based algorithm portfolios for numerical optimization. *IEEE Transactions on Evolutionary Computation*. 2010;14(5):782–800.

[12] Burke Edmund K, Gendreau Michel, Hyde Matthew, et al. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*. 2013;64(12):1695–1724.

[13] Qin A Kai, Suganthan Ponnuthurai N. Self-adaptive differential evolution algorithm for numerical optimization. In: The 2005 IEEE Congress on Evolutionary Computation (CEC), vol. 2: :1785–1791IEEE; 2005.

[14] Olorunda Olusegun, Engelbrecht Andries Petrus. An analysis of heterogeneous cooperative algorithms. In: The 2009 IEEE Congress on Evolutionary Computation (CEC):1562–1569IEEE; 2009.

[15] Vrugt Jasper A, Robinson Bruce A, Hyman James M. Self-adaptive multimethod search for global optimization in real-parameter spaces. *IEEE Transactions on Evolutionary Computation.* 2009;13(2):243–259.

[16] Grobler Jacomine, Engelbrecht Andries P, Kendall Graham, Yadavalli VSS. Alternative hyper-heuristic strategies for multi-method global optimization. In: The 2010 IEEE Congress on Evolutionary Computation (CEC):1–8IEEE; 2010.

[17] Alba E., Luque G., Nesmachnow S. Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research.* 2013;20(1):1-48.

[18] Dean Jeffrey, Ghemawat Sanjay. MapReduce: simplified data processing on large clusters. In: The 6th USENIX Symposium on Operating Systems Design and Implementation; 2004.

[19] Zaharia Matei, Chowdhury Mosharaf, Das Tathagata, et al. Resilient Distributed Datasets: a Fault-Tolerant Abstraction for In-Memory Cluster Computing. In: The 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI; 2012.

[20] McNabb Andrew W, Monson Christopher K, Seppi Kevin D. Parallel PSO using MapReduce. In: The 2007 IEEE Congress on Evolutionary Computation (CEC):7–14IEEE; 2007.

[21] Jin Chao, Vecchiola Christian, Buyya Rajkumar. MRPGA: an extension of MapReduce for parallelizing genetic algorithms. In: The 2008 IEEE Fourth International Conference on eScience:214–221IEEE; 2008.

[22] Verma Abhishek, Llora Xavier, Goldberg David E, Campbell Roy H. Scaling genetic algorithms using MapReduce. In: The Ninth International Conference on Intelligent Systems Design and Applications, ISDA'09:13–18IEEE; 2009.

[23] Radenski Atanas. Distributed simulated annealing with MapReduce. In: Applications of Evolutionary Computation. Springer 2012 (pp. 466–476).

[24] Lee Wei-Po, Hsiao Yu-Ting, Hwang Wei-Che. Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC systems biology.* 2014;8(1):5.

[25] Teijeiro Diego, Pardo Xoán C, González Patricia, Banga Julio R, Doallo Ramón. Implementing Parallel Differential Evolution on Spark. In: Applications of Evolutionary Computation. Lecture Notes in Computer Science, Vol. 9598:75–90Springer; 2016.

[26] Teijeiro Diego, Pardo Xoán C, González Patricia, Banga Julio R, Doallo Ramón. Towards cloud-based parallel metaheuristics: A case study in computational biology with Differential Evolution and Spark. *International Journal of High Performance Computing Applications.* 2016;.

[27] Teijeiro Diego, Pardo Xoán C, Penas David R, González Patricia, Banga Julio R, Doallo Ramón. Evaluation of Parallel Differential Evolution Implementations on MapReduce and Spark. In: Euro-Par 2016: Parallel Processing Workshops:397–408Springer; 2017.

[28] Evangelinos C, Hill C. Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. In: 1st Workshop on Cloud Computing and its Applications (CCA'08):1–6; 2008.

[29] Napper Jeffrey, Bientinesi Paolo. Can cloud computing reach the top500?. In: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop:17–20ACM; 2009.

[30] Ostermann Simon, Iosup Alexandru, Yigitbasi Nezih, Prodan Radu, Fahringer Thomas, Epema Dick. An early performance analysis of cloud computing services for scientific computing. *Delft University of Technology, Tech. Rep.* 2008;.

[31] Hazelhurst Scott. Scientific computing using virtual high-performance computing: a case study using the Amazon elastic computing cloud. In: Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology:94–103ACM; 2008.

[32] Keahey Kate, Freeman Tim, Lauret Jerome, Olson Doug. Virtual workspaces for scientific applications. *Journal of Physics: Conference Series.* 2007;78(1):012038.

[33] Ramakrishnan Lavanya, Jackson Keith R, Canon Shane, Cholia Shreyas, Shalf John. Defining future platform requirements for e-Science clouds. In: Proceedings of the 1st ACM symposium on Cloud computing:101–106ACM; 2010.

[34] Li Jie, Humphrey Marty, Van Ingen Catharine, Agarwal Deb, Jackson Keith, Ryu Youngryel. e-Science in the cloud: A modis satellite data reprojection and reduction pipeline in the Windows Azure platform. In: The 2010 IEEE International Symposium on Parallel & Distributed Processing:1–10IEEE; 2010.

[35] Expósito Roberto R, Taboada Guillermo L, Ramos Sabela, Touriño Juan, Doallo Ramón. Performance analysis of HPC applications in the cloud. *Future Generation Computer Systems.* 2013;29(1):218–229.

[36] González P., Pardo X. C., Penas D. R., Teijeiro D., Banga J. R., Doallo R.. Using the Cloud for parameter estimation problems: comparing Spark vs MPI with a case-study. In: The 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2017.

[37] Penas David R, González Patricia, Egea Jose A, Doallo Ramón, Banga Julio R. Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC bioinformatics.* 2017;18(1):52.

[38] Jin Haoqiang, Jespersen Dennis, Mehrotra Piyush, Biswas Rupak, Huang Lei, Chapman Barbara. High performance computing using MPI and OpenMP on multi-core parallel systems. *Parallel Computing.* 2011;37(9):562–575.

[39] Storn R., Price K. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization.* 1997;11(4):341-359.

[40] Penas David R, Banga Julio R, González Patricia, Doallo Ramón. Enhanced parallel Differential Evolution algorithm for problems in computational systems biology. *Applied Soft Computing.* 2015;33:86–99.

[41] Egea Jose A, Balsa-Canto Eva, García María S G, Banga Julio R. Dynamic optimization of nonlinear processes with an enhanced scatter search method. *Industrial & Engineering Chemistry Research.* 2009;48(9):4388–4401.

[42] Toulouse Michel, Crainic Teodor Gabriel, Thulasiraman Krishnaiyan. Global optimization properties of parallel cooperative search algorithms: a simulation study. *Parallel Computing.* 2000;26(1):91–112.

[43] Toulouse Michel, Crainic Teodor Gabriel, Sansó Brunilde. Systemic behavior of cooperative search algorithms. *Parallel Computing.* 2004;30(1):57–79.

[44] Villaverde Alejandro F, Henriques David, Smallbone Kieran, et al. BioPreDyn-bench: a suite of benchmark problems for dynamic modelling in systems biology. *BMC Systems Biology.* 2015;. In press.

[45] Hansen N., Auger A., Finck S., Ros R. *Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup.* RR-6828: INRIA; 2009.