Towards cloud-based parallel metaheuristics: A case study in computational biology with Differential Evolution and Spark

Diego Teijeiro[1], Xoán C Pardo[1], Patricia González[1], Julio R Banga[2] and Ramón Doallo[1]

Abstract

Many key problems in science and engineering can be formulated and solved using global optimization techniques. In the particular case of computational biology, the development of dynamic (kinetic) models is one of the current key issues. In this context, the problem of parameter estimation (model calibration) remains as a very challenging task. The complexity of the underlying models requires the use of efficient solvers to achieve adequate results in reasonable computation times. Metaheuristics have been the focus of great consideration as an efficient way of solving hard global optimization problems. Even so, in most realistic applications, metaheuristics require a very large computation time to obtain an acceptable result. Therefore, several parallel schemes have been proposed, most of them focused on traditional parallel programming interfaces and infrastructures. However, with the emergence of cloud computing, new programming models have been proposed to deal with large-scale data processing on clouds. In this paper we explore the applicability of these new models for global optimization problems using as a case study a set of challenging parameter estimation problems in systems biology. We have developed, using Spark, an island-based parallel version of Differential Evolution. Differential Evolution is a simple population-based metaheuristic that, at the same time, is very popular for being very efficient in real function global optimization. Several experiments were conducted both on a cluster and on the Microsoft Azure public cloud to evaluate the speedup and efficiency of the proposal, concluding that the Spark implementation achieves not only competitive speedup against the serial implementation, but also good scalability when the number of nodes grows. The results can be useful for those interested in using parallel metaheuristics for global optimization problems benefiting from the potential of new cloud programming models.

Keywords

Cloud computing, differential evolution, metaheuristics, Microsoft Azure, spark

1 Introduction

Global optimization problems arise in many areas of science and engineering (Banga, 2008; Floudas and Pardalos, 2013; Grossmann, 2013). In the particular case of biology, global optimization methods are playing an increasingly important role in key areas like computational biology (Greenberg et al., 2004), bioinformatics (Larrañaga et al., 2006) and systems biology (Banga, 2008).

Building the mathematical models that allow understanding complex biological systems is an iterative process that proposes a mathematical structure with a set of non-measurable parameters that have to be estimated in order to obtain quantitative predictions. The model is then (in)validated with new experiments, obtaining feedback which can be subsequently used in a refinement process. The parameter estimation step is key in this iterative process and can be formulated as a mathematical optimization problem subject to the dynamic constraints which describe the time-dependent behavior of the system.

[1]Computer Architecture Group, Universidade da Coruña, Spain
[2]BioProcess Engineering Group, IIM-CSIC, Spain

Corresponding author:
Patricia González, Universidade da Coruña, Facultade de Informatica, Campus de Elviña s/n, 15071 A Coruña, Spain.
Email: patricia.gonzalez@udc.es

Most biological models are highly non-linear dynamical systems, resulting in challenging multi-modal problems which are very difficult to solve (Villaverde and Banga, 2014). The computation effort when using deterministic global optimization methods might be extremely large, making them impractical. Thus, many research efforts have focused on developing metaheuristic methods which are able to locate the vicinity of the global solution in reasonable computation times. Moreover, in order to reduce the computational cost of these methods, a number of researchers have studied parallel strategies for metaheuristics (Crainic and Toulouse, 2003; Alba, 2005). In the area of computational biology, parallel methods have already shown promising results (Perkins et al., 2006; Jostins and Jaeger, 2010; Penas et al., 2015). However, all these efforts are focused on traditional parallel programming interfaces and traditional parallel infrastructures.

With the advent of cloud computing effortless access to a large number of distributed resources has become more feasible. But developing applications that execute at such a big scale is hard. New programming models are being proposed to deal with large scale computations on commodity clusters and cloud resources. Distributed frameworks like MapReduce (Dean and Ghemawat, 2008) or Spark (Zaharia et al., 2012) provide high-level programming abstractions that simplify the development of distributed applications including implicit support for deployment, data distribution, parallel processing and run-time features like fault tolerance or load balancing. We wonder how much benefit can we expect from implementing parallel metaheuristics using these new programming models because, besides the many advantages, they also have some shortcomings. Cloud-based distributed frameworks prefer availability to efficiency, so that the speedup and distributed efficiency are frequently lower than in traditional parallel frameworks due to the underlying multitenancy of virtualized resources.

The aim of this paper is to explore this direction further considering a parallel implementation of Differential Evolution (DE) (Storn and Price, 1997), probably one of the most popular heuristics for global optimization, to be executed in the cloud. In order to illustrate its performance we considered here a set of challenging parameter estimation problems in systems biology. A thorough evaluation has been carried out both in a local cluster and in the Microsoft Azure public cloud.

The organization of this paper is as follows. Some new programming models in the cloud are described in Section 2. Section 3 presents a brief overview of the DE algorithm. Section 4 describes the proposed implementation of the DE using Spark. The performance of the proposal is evaluated in Section 5, demonstrating its good efficiency and scalability. Section 6 covers the related work. Finally, Section 7 summarizes the main conclusions of this work.

2 New programming models in the cloud

From the new programming models that have been proposed to deal with large scale computations on cloud systems, MapReduce (Dean and Ghemawat, 2008) is the one that has attracted more attention since its appearance in 2004. In short, MapReduce executes in parallel several instances of a pair of user-provided *map* and *reduce* functions over a distributed network of *worker* processes driven by a single *master*. Executions in MapReduce are made in batches, using a distributed filesystem (typically HDFS) to take the input and store the output. MapReduce has been applied to a wide range of applications, including distributed pattern-based searching, distributed sorting, graph processing, document clustering and statistical machine translation, among others.

When it comes to iterative algorithms MapReduce has shown serious performance bottlenecks (Ekanayake et al., 2010), mainly because there is no way of reusing data or computation from previous iterations efficiently. New proposals, not based on MapReduce, like Spark (Zaharia et al., 2012) or Flink, which has its roots in Stratosphere (Alexandrov et al., 2014), are designed from the very beginning to provide efficient support for iterative algorithms.

Spark provides a language-integrated programming interface to resilient distributed datasets (RDDs), a distributed memory abstraction for supporting fault-tolerant and efficient in-memory computations. According to Zaharia et al. (2012) the performance of iterative algorithms can be improved by an order of magnitude when compared to MapReduce (using Hadoop). Formally, an RDD is a read–only fault–tolerant partitioned collection of records. Users can manipulate them using a rich set of operators, control their partitioning to optimize data placement and explicitly persist intermediate results (in memory by default but also to disk). RDDs are created from other RDDs or from data in stable storage by applying coarse-grained transformations (e.g. map, filter or join) that apply the same operation to many data items. Once created, RDDs are used in actions (e.g. count, collect or save) which are operations that return a value to the application or export data to a storage system.

3 Differential Evolution

DE is an iterative mutation algorithm where vector differences are used to create new candidate solutions. Starting from an initial population matrix composed of NP D-dimensional solution vectors (individuals), DE attempts to achieve the optimal solution iteratively

Algorithm 1: Differential Evolution algorithm (seqDE).

input: A population matrix P with size D x NP
output: A matrix P whose individuals were optimized
repeat
  for *each element i of the P matrix* **do**
    choose randomly different $r_1, r_2, r_3 \in [1, NP]$
    choose randomly an integer $j_r \in [1, D]$
    for $j \leftarrow 1$ **to** D **do**
      choose a randomly real $r \in [0, 1]$
      if $r \leqslant CR$ *or* $j = j_r$ **then**
        $u_i^{G+1}(j) \leftarrow x_{r1}^G(j) + F \cdot (x_{r2}^G(j) - x_{r3}^G(j))$
      else
        $u_i^{G+1}(j) \leftarrow x_i^G(j)$
      end
    end
    evaluate (u_i^{G+1})
    if $f(u_i^{G+1}) < f(x_i^G)$ **then**
      $x_i^{G+1} \leftarrow u_i^{G+1}$
    else
      $x_i^{G+1} \leftarrow x_i^G$
    end
  end
until *Stop conditions;*

through changes in its vectors. From now on we will describe and use the DE basic algorithm reported in Storn and Price (1997), specifically the DE/rand/1/bin scheme, however, the parallel implementation proposed in the next section can be applied to other DE schemes. Algorithm 1 shows the basic pseudocode for the specific version of the DE algorithm used in this paper. For each iteration, individuals are generated in a new population matrix through operations performed among individuals of the current population (mutation (F)), with old solutions replaced (crossover (CR)) only when the fitness value of the objective function is better than the current one. A population matrix with optimized individuals is obtained as the output of the algorithm. The best of these individuals is selected as solution close to optimal for the objective function of the model.

In some real applications, such as parameter estimation in dynamic models, the performance of the classical sequential DE is not acceptable due to the large number of objective function evaluations needed. As a result, typical runtimes for realistic problems are in the range of hours to days. The parallelization of metaheuristics pursues one or more of the following goals: increase the size of the problems that can be solved, speed-up the computations or attempt a more thorough exploration of the solution space. The solution explored in this work pursues the development of an efficient parallel variant of the serial DE, focused on both the acceleration of the computation by performing separate evaluations in parallel, and the convergence improvement through the stimulation of the diversification in the search and the cooperation between the parallel threads.

In the literature, different parallel models can be found (Alba et al., 2013) aiming to improve both computational time and number of iterations for convergence. The *master–slave* and the *island-based* models are the most popular. In the *master–slave* model, the behavior of the sequential DE is preserved by parallelizing the inner-loop of the algorithm. A master processor distributes computation operations between the slave processors. Therefore, the parallel algorithm has the same behavior as the sequential one. In the *island-based* model the population matrix is divided in subpopulations, (*islands*), where the algorithm is executed in isolation. Sparse individual exchanges are performed among islands to introduce diversity into the subpopulations, preventing search from getting stuck in local optima.

4 Implementing Differential Evolution on Spark

To understand the Spark-based parallel implementation of the DE algorithm, some previous insight into the way data is distributed and processed by Spark is needed. Spark uses the RDD abstraction to represent fault-tolerant distributed data. RDDs are immutable sets of records that optionally can be in the form of key–value pairs. Spark programs are run by a driver (the master in Spark terminology) which partitions RDDs and distributes the partitions to workers (the slaves in Spark terminology), that persist and transform them and return results to the driver. There is no communication among workers. Shuffle operations (i.e. join, groupBy) that need data movement among workers through the network are expensive and should be avoided.

With the aim of better understanding Spark intricacies and assessing the performance of different alternatives when implementing DE, in Teijeiro et al. (2016) we have presented a preliminary evaluation of (a) three different variants of the master–slave parallel implementation (SmsPDE), and (b) an island-based parallel implementation (SiPDE). The main conclusion of that work is that the island-based parallel implementation is the best suited to the distributed nature of Spark and obtains the best performance results. The main issue found in the DE master–slave model was the implementation of the mutation strategy because the population is partitioned and distributed among workers. For the mutation of each individual, random different individuals have to be selected from the whole population. How to access to individuals of other partitions from a given worker, having the constraint that only the driver has access to the complete population, was the main difficulty to be tackled, and all the solutions proposed to deal with this problem introduced an unfeasible communications overhead.

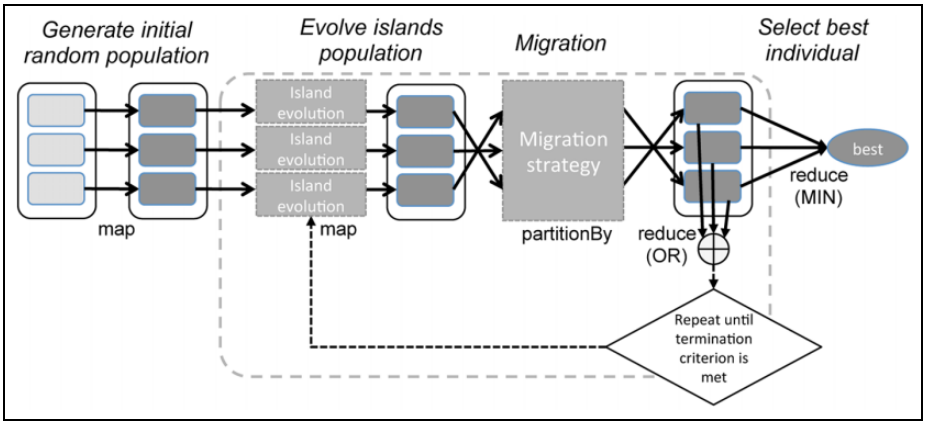

Figure 1. Spark-based island implementation of the DE algorithm (SiPDE).

The Spark island-based parallel DE implementation (SiPDE) proposed follows the scheme shown in Figure 1. In the figure, boxes with solid outlines are RDDs. Partitions are shaded rectangles, darker if they are persistent in memory. A key–value pair RDD has been used to represent the population where each individual is uniquely identified by its key. Some steps in the main flow of the algorithm are executed in a distributed fashion.

- The random generation and initial evaluation of individuals that form the population, implemented as a Spark map transformation.
- The evolution of the population. Every partition of the population RDD is considered to be an island, all with the same number of individuals. Islands evolve in isolation during a number of evolutions. This number can be configured and is the same for all islands. During these evolutions every worker calculates mutations picking random individuals from its local partition only.
- The migration strategy, which introduces diversity by exchanging selected individuals among islands every time the evolution of the islands ends.
- The checking of the termination criterion, implemented as a Spark reduce action (a distributed OR operation).

The evolution–migration loop is repeated until the termination criterion is met, after which the best individual is selected by means of a Spark reduce action (a distributed MIN operation).

For implementing the migration strategy a Spark feature known as *partitioner* has been used. In Spark the partitioner is responsible for assigning key–value pair RDD elements to partitions based on their keys. By default partitioner implements a hash-based partitioning using the key hash. For this work we have implemented a custom partitioner that randomly and evenly shuffles elements among partitions. It must be noted that this partitioner leads to a migration strategy that randomly shuffles individuals among subpopulations without replacement. This partitioner proposal is intended to evaluate the migration communications overhead and not to improve the searching quality of the algorithm. Adding migration strategies with that purpose in mind is left for future work.

Although the implementation of the *island-based* model in Spark drastically reduces the communications between different islands, the scalability is heavily restrained by the small size of the population matrix in the DE method. Having taken into account that in Storn and Price (1997) a guideline is given where the setting of the DE population size to about 10 times the dimensionality of the problem is proposed, reducing the already small population matrix by dividing it between the different islands will negatively impact the convergence of the DE. Thus, founded on the ideas outlined in Penas et al. (2015), the proposed implementation is extended to allow for the execution of a different DE in each island, using a different population matrix and different combinations of CR and F values to enhance diversity, while they cooperate through sparse migrations modifying the systemic properties of the individual searches. Thus, we have extended the SiPDE implementation to allow users to launch either homogeneous or heterogeneous islands, that is, having different combinations of CR and F values to enhance diversity.

5 Experimental results

In order to evaluate the efficiency of the Spark-based parallel implementation of the island DE algorithm (SiPDE), different experiments have been carried out. Its behavior, in terms of convergence and total execution time, has been first compared with the sequential implementation (seqDE). Then, we have also compared

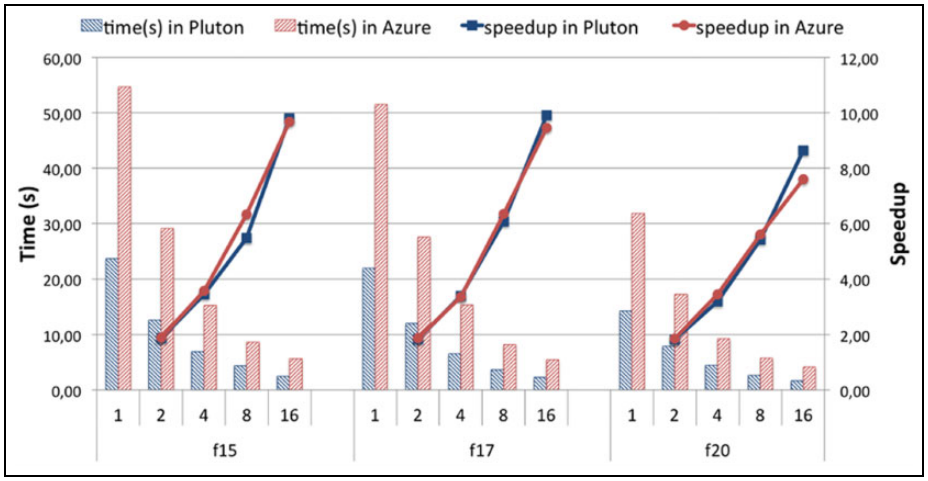

Figure 2. Execution time and speedup results in local cluster Pluton and Microsoft Azure public cloud. $D = 50$, $NP = 512$, $CR = 0.8$, $F = 0.9$, $\text{Nevals}_{max} = 500{,}000$.

the proposed Spark implementation with a MapReduce implementation.

For the experimental testbed two different platforms have been used. First, experiments were conducted in our local cluster Pluton, that consists of 16 nodes powered by two octa-core Intel Xeon E5-2660 CPUs with 64 GB of RAM, and connected through an InfiniBand FDR network. Second, experiments were deployed with default settings in the Microsoft Azure public cloud using a standard HDInsight Spark cluster with A3 standard instances (four cores, 7 GB) for head and worker nodes. It must be noted that, since Spark runs on the Java Virtual Machine (JVM), usual precautions (i.e. warm-up phase, effect of garbage collection) have been taken into account to avoid distortions on the measures. Also, because of the stochastic properties inherent to these algorithms, each experiment was executed over a number of 20 independent runs, and the average and standard deviation of the execution times are reported.

The performed experiments used two sets of benchmark problems: a set of problems out of an algebraic black-box optimization testbed, the Black-Box Optimization Benchmarking (BBOB) data set (Hansen et al., 2009); and three challenging parameter estimation problems in systems biology (Locke et al., 2005). On the one hand, the experiments over the BBOB data set were carried out to evaluate the efficiency of the proposed parallelization in a popular and accessible benchmarking testbed. On the other hand, the aim of the experiments with the parameter estimation problems in systems biology is to demonstrate the potential of the proposed techniques for improving the convergence and execution time of very hard problems. In these benchmarks, the execution of seqDE can take hours or even days to complete one only test.

5.1 Performance evaluation in general optimization problems

Three well known benchmark problems from the BBOB data set were evaluated: the Rastrigin function (f_{15}), the Schaffer function (f_{17}) and the Schwefel function (f_{20}). There are many configurable parameters in the classical DE algorithm, such as F (the mutation scaling factor) and CR (the crossover constant), or the mutation strategy and crossover type, whose selection may have significant impact on the algorithm performance. For the selection of the settings in these experiments, the suggestions in Storn and Price (1997) have been followed. Tests have been performed using DE/rand/1/bin scheme with $CR = 0.8$ and $F = 0.9$. Besides the problem dimension was set to $D = 50$ and the population size to $NP = 512$. As stopping criterion, a maximum number of evaluations of 500,000 has been used in these experiments.

Results obtained in the local cluster Pluton and in the Microsoft Azure public cloud are shown in Figure 2. In the primary axis of the figure the bars report the mean execution time of the 20 independent runs for each experiment, while in the secondary axis the speedup against the sequential implementation is shown. Comparing the sequential and the parallel metaheuristics is not an easy task; therefore, the guidance of Hansen et al. (2009) and Alba and Luque (2006) has been followed when analyzing the results of these experiments. The behavior of the proposed solution was compared with the sequential classic version of DE (seqDE); therefore, speedups calculated as $T_{\text{seqDE}}/T_{\text{SiPDE}}$ are reported in this section. This figure shows that the proposed SiPDE method accelerates the computation of seqDE by performing the same number of evaluations in parallel. As it can be seen, although the speedups achieved are similar in Azure and Pluton,

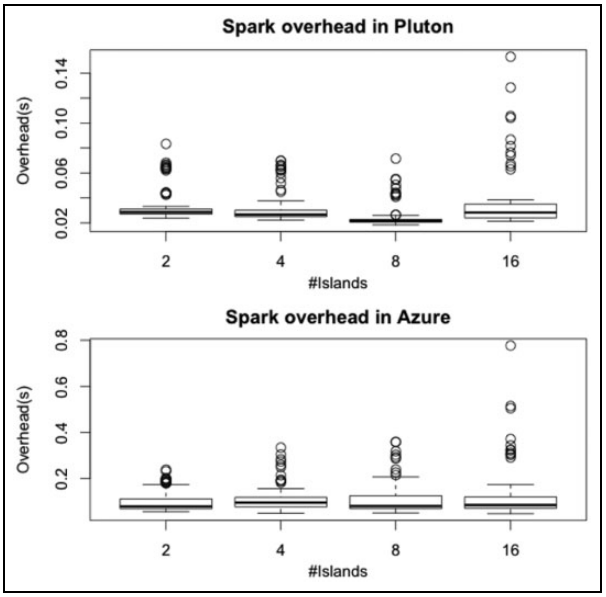

Figure 3. Box plot of the overhead times per evolution–migration iteration both in Pluton and Azure.

the execution times are longer in Azure, being between $2\times$ and $2.5\times$ the times obtained in Pluton. Virtualization overhead, use of non-dedicated resources in a multitenant platform and differences in node characteristics can explain these results. Even so, the Spark implementation achieves good results in terms of speedup versus the sequential implementation in Azure.

The speedup achieved in previous experiments deviates from the ideal one because of the overhead introduced by the communications in the parallel strategy and the overhead introduced by Spark. To evaluate this overhead we have used a modified version of our implementation in which the evolution of the population was removed. This modified implementation was executed for a total of eight *evolution–migration* iterations and the overhead of each iteration was measured separately in order to assess differences between them. Figure 3 shows the results obtained both in the local cluster and the Azure public cloud. As can be seen, the overhead in Azure is larger than in the local cluster (note that different scales are used in the figures). The first iteration in the Spark implementation is always the most time consuming (it corresponds to the outliers in the box plots). However, the rest of the iterations present even lower overhead and lower dispersion in the results, being the mean overhead of each iteration of 0.027 ± 0.006 s in Pluton and 0.092 ± 0.050 s in Azure. The overhead of the communications would barely affect the performance when the execution time between two migrations is significant.

5.2 Results for parameter estimation problems in systems biology

In order to evaluate the Spark implementation in real challenging problems, three difficult parameter estimation problems from the domain of computational biology were considered.

- *Circadian* model: Parameter estimation in a dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in Locke et al. (2005). The model consists of seven ordinary differential equations with 27 parameters (13 of them were estimated) with data sets from two experiments.
- *NFKB* model: This problem is based on the model in Lipniacki et al. (2004) and consists of 15 ordinary differential equations with 29 parameters and data sets from two experiments.
- *3-step pathway* model: problem considering a 3-step generic and highly non-linear pathway with eight differential equations and 36 parameters, and data sets from 16 experiments, as presented in Moles et al. (2003).

These problems are known to be particularly hard due to their ill-conditioning and non-convexity (Moles et al., 2003; Villaverde and Banga, 2014); thus, they are particularly appealing in assessing the performance of the homogeneous versus the heterogeneous configuration in the proposed implementation. The homogeneous configuration launches islands with the same CR and F values, while the heterogeneous configuration allows different combination of CR and F values to enhance diversity. For testing the homogeneous configuration of SiPDE $F = 0.9$ and $CR = 0.8$ were used, while for the heterogeneous configuration different combination of $CR = \{0.7, 0.8, 0.9\}$ and $F = \{0.8, 0.9\}$ values were randomly selected for each island. Nevertheless, the proposal can be applied to any other configuration parameters. Also, it is worth noting that further performance improvements can be achieved by further fine-tuning settings.

Since the aim is to decrease the execution time required for convergence in complex parameter estimation problems, the best way to fairly assess the performance of the proposal is to use as stopping criterion a value-to-reach (VTR). However, in the 3-step pathway and the NFKB benchmarks the execution of only one test could take several days to complete. Thus, we decided to use as stopping criterion: (a) a VTR $= 1e-5$ for the Circadian benchmark, evaluating its performance from a horizontal view; and (b) a predefined effort of maximum execution time $T_{\max} = 1000$ s for the Three-step pathway and the NFKB benchmarks, assessing their performance from a vertical view.

Results for the Circadian benchmark are shown in Table 1, and results for 3-step pathway and NFKB benchmarks are shown in Table 2. Table 1 displays, for each experiment, the number of cores (#Np) used, the

Table 1. Performance evaluation of the Circadian benchmark in Pluton. Stopping criterion: quality of the solution. Parameters: $D = 13$, $NP = 256$, VTR $= 1e-5$.

Method	#Np	#Mig.	Time (s)	Sp
seqDE	1	-	40,883.39±3,712.56	-
SiPDE (homo)	2	116	19,275.65±1,281.63	2.12
	4	112	9,305.30±1,038.59	4.39
	8	74	3,319.33±296.88	12.32
	16	36	790.97±90.50	51.69
SiPDE (hetero)	2	55	9,082.62±4,716.09	4.50
	4	37	3,179.47±1,339.96	12.86
	8	20	819.59±174.91	49.88
	16	19	403.39±84.66	101.35

Table 2. Performance evaluation of the 3-step pathway and NFKB benchmarks in Pluton. Stopping criterion: predefined effort, $T_{max} = 1000$s. Parameters for 3-step pathway: $D = 36$, $NP = 512$. Parameters for NFKB: $D = 29$, $NP = 512$.

	Method	#Np	#Mig.	#Evals	fbest
3-step pathway	seqDE	1	-	90,624	820.54
	SiPDE (homo)	2	8	191,232	753.52
		4	15	358,912	711.55
		8	27	653,312	690.06
		16	47	1,179,392	632.65
	SiPDE (hetero)	2	8	187,392	745,08
		4	15	352,512	698.28
		8	26	634,112	658.04
		16	44	1,094,912	530.00
NFKB	seqDE	1	-	21,274	0.06868
	SiPDE (homo)	2	10	44,032	0.06051
		4	17	81,408	0.05472
		8	30	143,104	0.05208
		16	47	239,104	0.04980
	SiPDE (hetero)	2	10	44,800	0.05959
		4	17	84,736	0.05419
		8	30	146,176	0.05220
		16	46	231,936	0.04732

mean number of migrations in the island-model algorithm (#Mig.), the mean execution times (Time (s)), and the speedup achieved versus the seqDE (Sp). Results show that the parallelization improves the execution time required for convergence by performing the evaluations in parallel, but also the cooperation between islands modifies the systemic properties of the algorithm improving the convergence rate. For complex problems, like the Circadian benchmark, the number of migrations clearly decreases with the number of nodes, demonstrating the potential of the parallel algorithm for improving the convergence of the DE method. The harder the problem is, the more improvement is achieved by the parallel algorithm, since the diversity introduced by the migration phase, although using a naive strategy as explained in Section 4, actually improves the effectiveness of the DE algorithm. Thus, for the Circadian benchmark superlinear speedups are obtained. Moreover, the diversification introduced in the heterogeneous approach outperforms the homogeneous approach, specially when the number of islands grows.

Table 2 displays, for each experiment, the number of cores (#Np) used, the mean number of migrations in the island-model algorithm (#Mig.), the average of the evaluations needed (#Evals), and the average of the best value for each run (fbest). Results show that the parallelization improves the convergence rate, since, in the same amount of time, more evaluations are executed in parallel and more migrations between islands are performed, thus, achieving better quality solutions. To better illustrate the improvement in convergence time, Figure 4 shows the convergence curves for the three benchmarks using the sequential algorithm and the parallel implementations with 16 islands. The convergence curves depicted are those that fall in the median values of the results distribution. It can be observed that at the beginning of the execution the performance of the homogeneous configuration is comparable with the heterogeneous one, since it is not until several migrations have passed that the effects of the heterogeneous search are noticeable. However, in the long-term behavior the heterogeneous configuration exhibits better

Table 3. Performance evaluation of the Circadian benchmark in Azure. Stopping criterion: quality of the solution. Parameters: $D = 13$, $NP = 256$, $VTR = 1e-5$.

Method	#Np	#Mig.	Time (s)	Sp
seqDE	1	-	9,529,470±5,623.22	-
SiPDE (homo)	2	121	47,895.80±5,091.57	1.99
	4	110	21,206.12±1,549.87	4.52
	8	77	11,449.79±1,951.18	8.32
	16	37	3,246.46±376.03	29.35
SiPDE (hetero)	2	40	16,026.56±8,858.46	4.72
	4	32	6,595.61±3,121.84	11.46
	8	21	3,065.93±663.73	24.66
	16	19	1,652.05±377.73	45.76

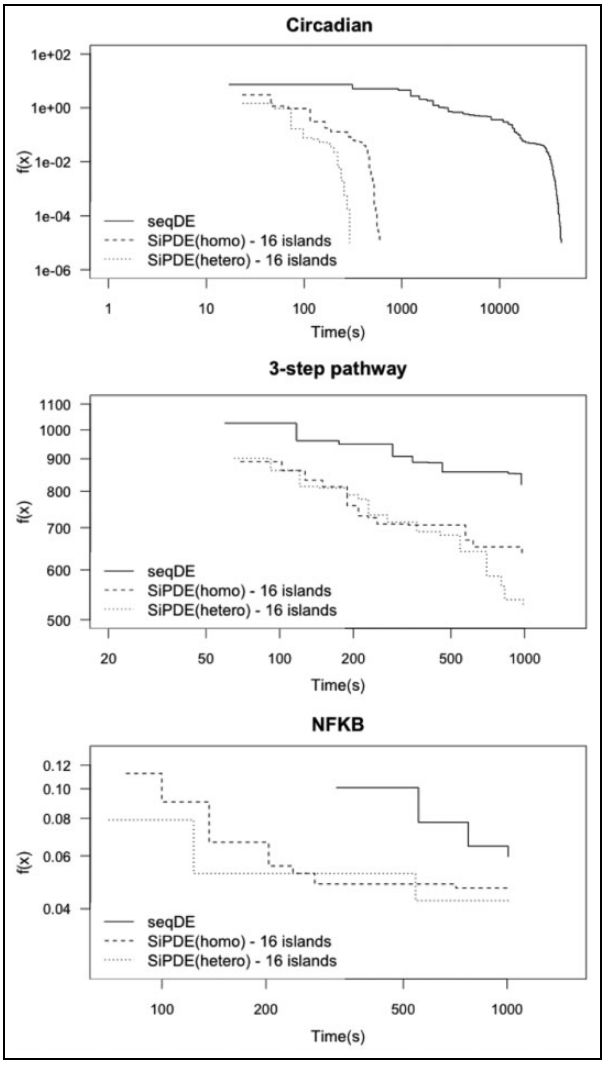

Figure 4. Convergence curves. Parameters: $D_{circadian} = 13$, $NP_{circadian} = 256$, $D_{3-step} = 36$, $NP_{3-step} = 512$, $D_{NFKB} = 29$ and $NP_{NFKB} = 512$, Circadian using as stopping criterion a $VTR = 1e-5$, while 3-step pathway and NFKB using as stopping criterion a predefined effort of $T_{max} = 1000$s.

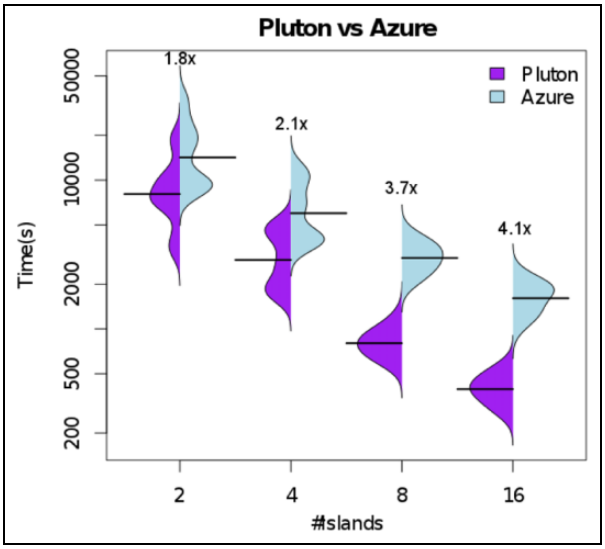

Figure 5. Bean plots comparing the results in Pluton and Azure for the heterogeneous configuration reported in Tables 1 and 3. The speedup achieved in Pluton vs Azure is displayed on top of each bean.

performance than the homogeneous one (note the logarithmic scale in y-axis).

Finally, in order to evaluate the performance of the proposal with these challenging problems in a public cloud, some experiments were conducted in the Azure public cloud. As can be seen in Table 3, the proposal achieves similar results in Azure as the ones obtained in the local cluster in terms of convergence and scalability; however, the overheads introduced in Azure due to virtualization and use of non-dedicated resources in a multitenant platform are not negligible. Since the values in the table hide the underlying distribution, which in these kind of stochastic problems is very important, Figure 5 shows the bean plots to compare the distribution of the heterogeneous configuration in both the local cluster and Azure. The mean of each distribution is also shown in each bean. This figure clearly shows, not only the larger execution time but also the larger dispersion in the results obtained in Azure (note the logarithmic scale in the y-axis).

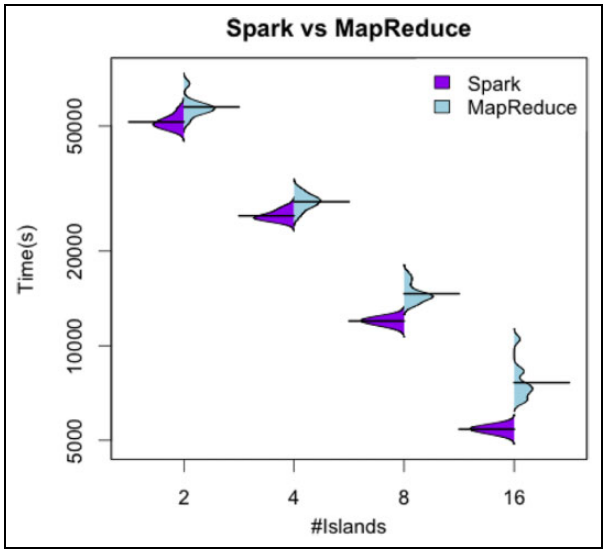

Figure 6. Bean plots comparing Spark vs MapReduce implementation in cluster Pluton for the Circadian benchmark. Parameters: $D = 13$, $NP = 640$, VTR $= 1e-5$.

5.3 Comparison with a MapReduce implementation

Finally, we have performed several tests to assess how competitive the Spark parallel implementation can be with respect to a MapReduce implementation, since MapReduce is still the de-facto standard for large-scale data-intensive applications.

For the comparison we have developed a MapReduce implementation of the homogeneous island-based parallel DE and we have repeated the same previous experiments with the Circadian benchmark in Pluton but this time using Hadoop (v2.7.1). Figure 6 shows a bean plot that allows for an easy comparison of the execution times obtained using the MapReduce and the Spark implementations in the local cluster. Note that not only is the execution time larger for the MapReduce implementation but also the dispersion of the results obtained is bigger.

The experimental results show that MapReduce has significantly higher overhead per iteration than Spark, mainly caused by longer task initialization times and HDFS access. Figure 7 shows the overhead of MapReduce per evolution–migration iteration. In opposition to Spark, in the case of MapReduce there is no significant difference between the first and the subsequent iterations, and the figures clearly indicate a higher overhead and large dispersion in the results, being the mean overhead of each iteration 17.95 ± 2.50s versus the 0.027 ± 0.006s in Spark (see Figure 3).

6 Related work

The parallelization of metaheuristics methods has received much attention with regards to reducing the run time for solving large-scale problems (Alba et al., 2013). Many parallel algorithms have been proposed in the literature, most of them being parallel implementations based on traditional parallel programming interfaces. Some of these proposals are focused on the parallelization of DE. A parallel synchronous approach was proposed in Tasoulis et al. (2004), based on the distribution of the population data among different processors (slaves), which communicate through data migrations, all of them managed by a central processor (master). A simple approach to asynchronous parallelization was proposed in Ntipteni et al. (2006), consisting of a master–slave architecture with several independent processes, where the communications were not established directly but through the filesystem. Another asynchronous proposal based on a master–slave approach is the parallel metaheuristic based on DE and simulated annealing proposed in Olenšek et al. (2011). In Apolloni et al. (2014) another distributed DE was presented, in this case exploiting an island-model with asynchronous communication.

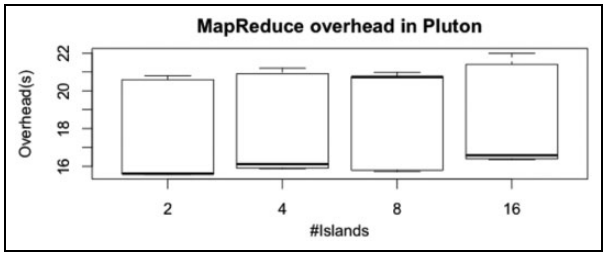

Figure 7. Box plot of the MapReduce overhead times per evolution–migration iteration in Pluton.

Several other works studied improvements to island-model schemes. Migration strategy plays an important role in the performance of distributed DE schemes. In Rucinski et al. (2010), a complete study about the impact on the performance of different communication topologies of the islands was presented. These authors used a synchronous parallel DE on a set of standard benchmarks with different topologies, concluding that ring topology was the best option. The influence of synchronous and asynchronous migration on the performance of adaptive DE algorithms has been also investigated in Bujok (2013). In De Falco et al. (2014) a novel migration model is implemented through a multistage process that involves invading subpopulations and their competition with native individuals. Recently, in Kozlov et al. (2016), a parallel technique called DEEP (Differential Evolution Entirely Parallel) has been implemented and publicly released. They propose a new migration scheme in which the best member of one population substitutes for the oldest member of a target population, organized in a ring.

Several studies suggest that randomization of the control parameters can be a propitious mechanism for

enhancing the DE performance (Brest et al., 2006). Different randomization schemes have been proposed to develop self-adaptive DE frameworks and investigate the effect of changing control parameters in distributed DE (Weber et al., 2011b, 2013). Two mechanisms to avoid the loss of diversity when the size of the population is small are described in Weber et al. (2011a). The first one was based on shuffling: the individuals from a specific subpopulation were randomly reorganized. The second one, an update mechanism, changed and adapted scaling factors for each subpopulation. The results indicated that these techniques obtained a very significant performance improvement when the dimensionality of the functions grew. In Dorronsoro and Bouvry (2011) several DE variants using different decentralized population schemes were proposed and evaluated, demonstrating that the population scheme has a marked influence on the behavior of DE algorithms.

Since there is an extensive bibliography on parallel schemes and improvements for the DE algorithm, those readers interested in further information can find a detailed review in Das and Suganthan (2011) and more recently in Das et al. (2016). Note that most of these different island-based DE schemes can be easily supported by the Spark-based implementation proposed in this work.

Research on cloud-oriented parallel metaheuristics based mainly on the use of MapReduce has also received increasing attention in recent years. MRPSO (McNabb et al., 2007) uses the MapReduce model to parallelize Particle Swarm Optimization (PSO). MRPGA (Jin et al., 2008) attempts to combine MapReduce and genetic algorithms (GA). They properly claim that GAs cannot be directly expressed in MapReduce due to their specific characteristics, so they extend the model featuring a hierarchical reduction phase. However, they only perform in parallel the fitness evaluation, so this approach shows poor scalability. A different approach is followed in Verma et al. (2009), which tries to hammer the GAs into the MapReduce model. In Radenski (2012) the applicability of MapReduce to distributed simulated annealing (SA) was also investigated. They design different algorithmic patterns of distributed SA with MapReduce and evaluate their proposal on the Azure public cloud. Recently, in Lee et al. (2014), a practical framework to infer large gene networks through a parallel hybrid GA–PSO optimization method using MapReduce has also been proposed.

Some proposals are more specific in studying how to apply MapReduce to parallelize the DE algorithm to be used in the cloud. In Zhou (2010) the fitness evaluation in the DE algorithm is performed in parallel using Hadoop (the well-known open-source MapReduce framework). However, the experimental results reveal that the extra cost of Hadoop DFS I/O operations and the system bookkeeping overhead significantly reduces the benefits of the parallelization. In Tagawa and Ishimizu (2010), a concurrent implementation of the DE based on MapReduce is proposed; however, it is a parallelized version of a neoteric DE based on the steady-state model instead of the generation alternation model. While the generational model holds two populations and generates all individuals for the second population from those of the current population, the steady-state model holds only one population and each individual of the population is updated one by one. Compared with the generational model, the parallelization of the steady-state model is simpler because it does not require synchronization for replacing the current population by newborn individuals simultaneously. On the other hand, the experiments reported in that paper were conducted on a multi-core CPU; thus, their implementation takes advantage of the shared-memory architecture, sharing the population among the different threads, which is not possible in a distributed cloud environment. In Daoudi et al. (2014) a parallel implementation of DE based clustering using MapReduce is also proposed. This algorithm was implemented in three levels, each of which consists of different DE operations.

An attempt to parallelize the DE algorithm based on RDDs is presented in Deng et al. (2015). However, in that work only the computation of the fitness values of the individuals was performed in parallel following a master–slave approach. An entire Spark-based parallelization of the DE algorithm was explored in Teijeiro et al. (2016). In that paper Spark-based implementations of two different parallel schemes of the DE algorithm, the master–slave and the island-based, are proposed and evaluated. Results showed that the island-based solution is by far the best suited to the distributed nature of Spark.

7 Conclusions

In order to explore how parallel metaheuristics could take advantage of the recent advances in cloud programming models, in this paper an island-based Spark implementation (SiPDE) of the DE algorithm is proposed and evaluated. A thorough evaluation of this implementation was conducted both on a local cluster and on the Microsoft Azure public cloud, using both synthetic and real biology-inspired benchmarks. The experimental results show that the proposal achieves not only competitive speedup against the serial implementation, but also good scalability when the number of nodes grows. Results using the Azure cloud resources show similar behavior in terms of convergence and scalability as using resources from a local cluster, but at the expense of a not negligible overhead.

Due to the complexity of the problems being considered in this work, SiPDE implementation was extended to allow users to launch either homogeneous or heterogeneous islands to enhance diversity and improve the convergence rate. The results obtained show that the heterogeneous configuration achieves an adequate balance between exploration and exploitation, thus, it outperforms the homogeneous one for the class of problems considered.

A comparison with a MapReduce implementation has been also carried out. The experimental results show that MapReduce has significantly higher overhead per iteration than Spark, mainly caused by longer task initialization times and HDFS access, and that Spark has the best support for iterative algorithms, as it reduces the overhead between the first and subsequent iterations.

Although this cloud-based implementation was designed and tested with focus on the field of parameter estimation problems in computational biology, it can also be directly applied to solve arbitrary global optimization problems. In particular, we believe that both the description of the Spark implementation and the results obtained in this work can be useful for those researchers interested in the potential of new cloud programming models for developing parallel metaheuristic methods.

The source code is made publicly available at https://bitbucket.org/xcpardo/sipde.

Acknowledgement

The authors would like to acknowledge Microsoft Research awarding this project with a sponsored Azure account.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This was supported by the Spanish Ministerio de Economía y Competitividad (and the FEDER) through the Project SYNBIOFACTORY (Grant number DPI2014-55276-C5-2-R), by the Spanish Ministerio de Ciencia e Innovación (and the FEDER) projects TIN2013-42148-P and TIN2016-75845-P and by the Galician Government (Xunta de Galicia) under the Consolidation Program of Competitive Research Units (Network Ref. R2014/041) cofunded by FEDER funds of the EU.

References

Alba E (2005) *Parallel Metaheuristics: A New Class of Algorithms*. Berlin-Heidelberg: Wiley-Interscience.

Alba E and Luque G (2006) Evaluation of parallel metaheuristics. In: *Proceedings of the 9th international conference on parallel problem solving from nature (PPSN-EMAA'06)*, Reykjavik, Iceland, pp.9–14.

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

Alexandrov A, Bergmann R, Ewen S, et al. (2014) The stratosphere platform for big data analytics. *The VLDB Journal* 23(6): 939–964.

Apolloni J, García-Nieto J, Alba E, et al. (2014) Empirical evaluation of distributed differential evolution on standard benchmarks. *Applied Mathematics and Computation* 236: 351–366.

Banga JR (2008) Optimization in computational systems biology. *BMC Systems Biology* 2(1): 47.

Brest J, Greiner S, Boskovic B, et al. (2006) Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation* 10(6): 646–657.

Bujok P (2013) Synchronous and asynchronous migration in adaptive differential evolution algorithms. *Neural Network World* 23(1): 17.

Crainic TG and Toulouse M (2003) Parallel strategies for meta-heuristics. In: Crainic TG and Toulouse M (eds) *Handbook of Metaheuristics*, pp.475–513. Springer.

Daoudi M, Hamena S, Benmounah Z, et al. (2014) Parallel differential evolution clustering algorithm based on MapReduce. In: *6th international conference of soft computing and pattern recognition (SoCPaR)*, Tunis, Tunisia, 11–14 August 2014, pp.337–341. IEEE.

Das S and Suganthan PN (2011) Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation* 15(1): 4–31.

Das S, Mullick SS and Suganthan P (2016) Recent advances in differential evolution–An updated survey. *Swarm and Evolutionary Computation* 27: 1–30.

De Falco I, Della Cioppa A, Maisto D, et al. (2014) An adaptive invasion-based model for distributed differential evolution. *Information Sciences* 278: 653–672.

Dean J and Ghemawat S (2008) MapReduce: Simplified data processing on large clusters. *Communications of the ACM* 51(1): 107–113.

Deng C, Tan X, Dong X, et al. (2015) A parallel version of differential evolution based on resilient distributed datasets model. In: *Bio-Inspired Computing-Theories and Applications*. Springer, pp.84–93.

Dorronsoro B and Bouvry P (2011) Improving classical and decentralized differential evolution with new mutation operator and population topologies. *IEEE Transactions on Evolutionary Computation* 15(1): 67–98.

Ekanayake J, Li H, Zhang B, et al. (2010) Twister: A runtime for iterative MapReduce. In: *Proceedings of the 19th ACM international symposium on high performance distributed computing*, Chicago, Illinois, USA, 20–25 June 2010, pp.810–818. ACM.

Floudas CA and Pardalos PM (2013) *Optimization in Computational Chemistry and Molecular Biology: Local and Global Approaches*. Springer Science & Business Media.

Greenberg HJ, Hart WE and Lancia G (2004) Opportunities for combinatorial optimization in computational biology. *INFORMS Journal on Computing* 16(3): 211–231.

Grossmann IE (2013) *Global Optimization in Engineering Design*. Springer Science & Business Media.

Hansen N, Auger A, Finck S, et al. (2009) Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA.

Jin C, Vecchiola C and Buyya R (2008) MRPGA: An extension of MapReduce for parallelizing genetic algorithms. In: *IEEE fourth international conference on eScience (eScience '08)*, Indianapolis, Indiana, USA, 10–12 December 2008, pp.214–221. IEEE.

Jostins L and Jaeger J (2010) Reverse engineering a gene network using an asynchronous parallel evolution strategy. *BMC Systems Biology* 4(1): 17.

Kozlov K, Samsonov AM and Samsonova M (2016) A software for parameter optimization with differential evolution entirely parallel method. *PeerJ Computer Science* 2: e74.

Larrañaga P, Calvo B, Santana R, et al. (2006) Machine learning in bioinformatics. *Briefings in Bioinformatics* 7(1): 86–112.

Lee WP, Hsiao YT and Hwang WC (2014) Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC Systems Biology* 8(1): 5.

Lipniacki T, Paszek P, Brasier A, et al. (2004) Mathematical model of nf-κb regulatory module. *Journal of Theoretical Biology* 228(2): 195–215.

Locke J, Millar A and Turner M (2005) Modelling genetic networks with noisy and varied experimental data: The circadian clock in arabidopsis Thaliana. *Journal of Theoretical Biology* 234(3): 383–393.

McNabb AW, Monson CK and Seppi KD (2007) Parallel PSO using MapReduce. In: *IEEE congress on evolutionary computation, CEC2007*, Singapore, 25–28 September 2007, pp.7–14. IEEE.

Moles C, Mendes P and Banga JR (2003) Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Research* 13(11): 2467–2474.

Ntipteni MS, Valakos IM and Nikolos IK (2006) An asynchronous parallel differential evolution algorithms. In: *Proceedings of the ERCOFTAC conference on design optimisation: Methods and application*, Las Palmas de Gran Canaria, Spain 5–7 April 2006.

Olenšek J, Tuma T, Puhan J, et al. (2011) A new asynchronous parallel global optimization method based on simulated annealing and differential evolution. *Applied Soft Computing* 11(1): 1481–1489.

Penas D, Banga J, González P, et al. (2015) Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing* 33: 86–99.

Perkins TJ, Jaeger J, Reinitz J, et al. (2006) Reverse engineering the gap gene network of drosophila Melanogaster. *PLOS Computational Biology* 2(5): e51.

Radenski A (2012) Distributed simulated annealing with MapReduce. In: *Applications of Evolutionary Computation*. Springer, pp.466–476.

Rucinski M, Izzo D and Biscani F (2010) On the impact of the migration topology on the island model. *Parallel Computing* 36(10-11): 555–571.

Storn R and Price K (1997) Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11(4): 341–359.

Tagawa K and Ishimizu T (2010) Concurrent differential evolution based on MapReduce. *International Journal of Computers* 4(4): 161–168.

Tasoulis DK, Pavlidis NG, Plagianakos VP, et al. (2004) Parallel differential evolution. In: *IEEE congress on evolutionary computation (CEC2004)*, Portland, Oregon, USA, 20–23 June 2004, volume 2, pp.2023–2029. IEEE.

Teijeiro D, Pardo XC, González P, et al. (2016) Implementing parallel differential evolution on spark. In: *Applications of Evolutionary Computation* (*Lecture Notes in Computer Science*, vol. 9598). Springer, pp.75–90.

Verma A, Llora X, Goldberg DE, et al. (2009) Scaling genetic algorithms using MapReduce. In: *Ninth international conference on intelligent systems design and applications (ISDA '09)*, Pisa, Italy November 30–December 2 2009. Pisa, Italy, 30 November–2 December 2009, pp.13–18. IEEE.

Villaverde A and Banga JR (2014) Reverse engineering and identification in systems biology: Strategies, perspectives and challenges. *Journal of the Royal Society Interface* 11(91): 20130505.

Weber M, Neri F and Tirronen V (2011a) Shuffle or update parallel differential evolution for large-scale optimization. *Soft Computing* 15(11): 2089–2107.

Weber M, Neri F and Tirronen V (2011b) A study on scale factor in distributed differential evolution. *Information Sciences* 181(12): 2488–2511.

Weber M, Neri F and Tirronen V (2013) A study on scale factor/crossover interaction in distributed differential evolution. *Artificial Intelligence Review* 39(3): 195–224.

Zaharia M, Chowdhury M, Das T, et al. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: *Proceedings of the 9th USENIX conference on networked systems design and implementation*, San José, California, USA, 25–27 April 2012, p.2. USENIX Association.

Zhou C (2010) Fast parallelization of differential evolution algorithm using MapReduce. In: *Proceedings of the 12th annual conference on genetic and evolutionary computation*, Portland, Oregon, USA, 7–11 July 2010, pp.1113–1114. ACM.

Author biographies

Diego Teijeiro received an MS (2016) degree in Computer Science from the University of A Coruña, Spain. Currently he is a postgraduate student in the Computer Architecture Group at the Department of Electronics and Systems in the University of A Coruña. His research focuses on cloud frameworks and programming paradigms for scientific computing.

Xoán C Pardo received his BS (1994) and MS (1995) degrees in Computer Science and his PhD (2004) degree in Computer Engineering from the University of A Coruña, Spain. He has been an associate professor in the Department of Electronics and Systems at the University of A Coruña since 2008. His main research interests are in the area of high performance computing focused on new cluster and cloud frameworks and programming paradigms for scientific computing.

Patricia González received MS (1996) and PhD (2001) degrees in Physics from the University of Santiago de Compostela, Spain. Currently she is an associate professor in the Department of Electronics and Systems at the University of A Coruña. Her main research interests are in the area of high performance computing, focused on parallel and distributed computing and fault-tolerance for parallel applications.

Julio R Banga is a research professor at CSIC (Spanish Council for Scientific Research). He works at the Bio-Process Engineering Group, located at the IIM-CSIC, Vigo (Spain), doing research in the area of computational systems biology. He obtained a PhD in Chemical Engineering from the University of Santiago de Compostela (Spain) in 1991. He has been a postdoc at the University of California, Davis, and a visiting researcher at the University of Pennsylvania and MIT. He has supervised over 10 PhD students, and is the author of more than 160 archival publications. He has been involved in over 40 major research projects and contracts. Currently he is a member of the Editorial Board of BMC Systems Biology, the IFAC Technical Committee on Control of Biotechnological Processes, and several international advisory boards.

Ramón Doallo received his BS (1987), MS (1987) and PhD (1992) degrees in Physics from the University of Santiago de Compostela, Spain. In 1990 he joined the Department of Electronics and Systems at the University of A Coruña, Spain, where he became a full professor in 1999. He has extensively published in the areas of computer architecture, and parallel and distributed computing. He is coauthor of more than 140 technical papers on these topics.