Multi-Objective Big Data Optimization with jMetal and Spark

Cristóbal Barba-Gonzaléz, José García-Nieto, Antonio J. Nebro*, and José F. Aldana-Montes

Dept. de Lenguajes y Ciencias de la Computación, University of Malaga,
ETSI Informática, Campus de Teatinos, Malaga - 29071, Spain
cbarba@lcc.uma.es, jnieto@lcc.uma.es,
antonio@lcc.uma.es, jfam@lcc.uma.es

Abstract. Big Data Optimization is the term used to refer to optimization problems which have to manage very large amounts of data. In this paper, we focus on the parallelization of metaheuristics with the Apache Spark cluster computing system for solving multi-objective Big Data Optimization problems. Our purpose is to study the influence of accessing data stored in the Hadoop File System (HDFS) in each evaluation step of a metaheuristic and to provide a software tool to solve these kinds of problems. This tool combines the jMetal multi-objective optimization framework with Apache Spark. We have carried out experiments to measure the performance of the proposed parallel infrastructure in an environment based on virtual machines in a local cluster comprising up to 100 cores. We obtained interesting results for computational effort and propose guidelines to face multi-objective Big Data Optimization problems.

Keywords: Multi-objective Optimization, Big Data, jMetal, Spark, Parallel Computing.

1 Introduction

Over the past few years, Big Data technologies have attracted more and more attention, leading to an upsurge in research, industry and government applications. There are multiple opportunities and challenges in Big Data research. One area in particular where Big Data is promising is Global Optimization [26]. The issue is that Big Data optimization problems may need to access a massive amount of data to be solved, which introduces a new dimension of complexity apart from features such as non-linearity, uncertainty and conflicting objectives.

Focusing on multi-objective optimization, metaheuristic search methods, such as evolutionary algorithms, have been widely applied to a great number of academic and industry optimization problems [6]. Depending on the problem, a metaheuristic may need to perform thousands or even millions of solution evaluations. In the case of complex system optimization, the computational effort, in

* Corresponding author antonio@lcc.uma.es

terms of time consumption and resource requirements, to evaluate the quality of solutions can make it impracticable to apply current optimization strategies to such problems. This issue is even harder when dealing with Big Data environments, where a huge volume of data hast to be accurately and quickly managed.

One strategy to cope with these difficulties is to apply parallelism [17]. In the last few years, a number of approaches consisting in adapting metaheuristic techniques to work in parallel on Hadoop, the de facto Big Data software platform, have been proposed. These proposals are related to data mining or data management applications, such as: feature selection [1], data partitioning [12], dimension reduction [23], pattern detection [5], graph inference [16], and task scheduling [22]. Most of these approaches are based on the MapReduce programming model [15].

However, MapReduce entails a series of drawbacks that make it unsuitable to be properly integrated with metaheuristics in particular and with global optimization techniques in general. Chief among them are: high latency queries, non-iterative programming model, and weak real-time processing. Therefore, there is a demand for new challenging approaches to integrate Big Data based technologies with global optimization algorithms in order to cope with all these issues.

In this paper, our approach is to address the parallelization of metaheuristics in Hadoop-based systems with Apache Spark [25], which is defined as a fast and general engine for large-scale data processing. Our proposal is to use the jMetalSP framework[1], which combines Spark with the jMetal multi-objective optimization framework [11][2]. Concretely, we have included support in jMetalSP to parallelize metaheuristics with Spark in an almost transparent way, hence avoiding the intrinsic shortcomings of the usual MapReduce model when applied to global optimization.

We aim to consider two scenarios: first, to use Spark as an engine to evaluate the solutions of a metaheuristic in parallel, and, second to study the influence of accessing a massive amount of data in each evaluation of a metaheuristic algorithm. Instead of focusing on a particular optimization problem, we have defined a generic scenario, in which a benchmark problem is modified to artificially increase its computing time and to read data from the Hadoop file system (HDFS). We have carried out a number of experiments to measure the performance of the proposed method in a parallel virtualization infrastructure of multiple machines in an in-house cluster.

The main contributions of this paper are as follows:

- We provide a software solution for parallelizing multi-objective metaheuristics included in the jMetal framework to take advantage of the high performance cluster computing facilities provided by Spark. This way, developers and practitioners are provided with an attractive tool for Big Data Optimization.

[1] In URL https://github.com/jMetal/jMetalSP
[2] In URL http://jmetal.github.io/jMetal/

– We perform a thorough experimentation of our proposal from three different viewpoints. First, by measuring the performance of algorithms in terms of computational effort in a Hadoop parallel environment; second, by analyzing the influence of accessing data stored in HDFS in each evaluation of a metaheuristic; and third, by combining both approaches. To carry out this study, we define different data access/processing tasks to be done when evaluation a solution, so we can measure the performance of the algorithms according to computational effort and size of data. This allows us to compute the speedups that can be obtained and identify the system's limits, to determine whether or not it is worth using more resources to solve the problem.

The remainder of this article is organized as follows. The next section presents an overview of the related work in the literature. Section 3 details our Big Data optimization approach. In Section 4, the experimental framework and parameters settings are described. Section 5 details the experimental results and analyses. Finally, Section 6 outlines some concluding remarks and plans for future work.

2 Related Work

Large amounts of data and high dimensionality characterizes many optimization problems in interdisciplinary domains such as, biomedical sciences, engineering, finance, and social sciences. This means that optimization problems handling such spatio-temporal restrictions often deal with tens of thousands of variables or features extracted from documents, images and other objects.

To tackle such challenging problems, a series of proposals have appeared in the last decade, which combine metaheuristics with data mining or data management applications, and adapt them to perform in Hadoop environments. Concretely, in [1] a swarm intelligence approach is adapted to optimize the features that exist in large protein sequences using a two-tier hybrid model by applying both filter and wrapper methods. A similar approach is proposed in [12], where a particle swarm optimization algorithm (PSO) is used to discover clusters in data that are continuously captured from students' learning interactions. In [2], intrusion detection is managed with a MapReduce strategy based on a PSO clustering algorithm.

An interesting method has been reported in [23], in which, by using sensor data to generate a PCA (Principal Component Analysis) model to forecast photovoltaic energy, it is possible to reduce the dimensionality of data with the collaboration of other artificial intelligence techniques, such as: fuzzy interference, neural networks, and genetic algorithms.

In the case of biomedical sciences, the reconstruction of gene regulatory networks is a complex optimization problem that is attracting particularly special from the research community, since it is considered to be a potential Big Data problem in the specialized literature [26,3]. Along the same lines, in the study carried out in [16], the authors proposed a parallel method consisting in a hybrid genetic algorithm with PSO by means of the MapReduce programming model. The resulting approach was tested in different cloud computing environments.

Most of these approaches are based on the MapReduce (MR) programming model [15], which yields a competitive performance in comparison with other parallel (and sequential) models for the specific problems tackled. In addition, other algorithmic adaptations to MapReduce operations can be found in metaheuristics, such as: PSO [18], Differential Evolution [9] and Ant Colony Optimization [24]. However, as stated, the MapReduce model entails a series of drawbacks that make it unsuitable to be integrated with metaheuristics in particular and with global optimization techniques in general. These are directly related to: high-latency queries, a non-iterative programming model, and weak real-time processing. For instance, the following issues which we aim to cover with our proposal, combining jMetal and Spark:

- MapReduce uses coarse-grained tasks to do its work, which can be too heavyweight for iterative algorithms, like metaheuristic algorithms. In the proposals analyzed, developers use various MapReduce hacks or alternative tools to overcome these limitations, but this highlights the need for a better computation engine that supports these algorithms directly, while continuing to support more traditional batch processing of large datasets. Our software proposal follows an iterative programming model, which eases the adaptation of algorithms and the integration with software classes managing a multitude of optimization problems.
- Another problem with current optimization algorithms using MapReduce is that they have no awareness of the total pipeline of Map plus Reduce steps, so they cannot cache intermediate data in memory for faster performance. Instead, they flush intermediate data to disk between each step. With Spark the managed data can be cached in memory explicitly, thus improving performance significantly.
- Existing proposals in the literature were not evaluated on well-grounded Big Data environments. Most of them were tested to show their ability to solve a given optimization problem in a parallel infrastructure composed of up to ten machines, and therefore critical aspects such as data volume and variable computational effort remain open issues. In the present study, a thorough experimentation is carried out to measure the performance of algorithms in terms of scalability in a 100-core Hadoop-based cluster, and to analyze the influence of accessing a large amount of data in each evaluation of a multi-objective metaheuristic.

3 Big Data Optimization Approach

To develop Big Data optimization applications it is necessary to have software tools capable of coping with the requirements of such applications. Our contribution in this sense is to propose jMetalSP [4][7], an open-source platform[3] combining the jMetal optimization framework [11] with the Apache Spark cluster computing system [25]. Below we describe the adopted Big Data Optimization scheme in the context of jMetal, Spark, and jMetalSP.

[3] URL: https://github.com/jMetal/jMetalSP

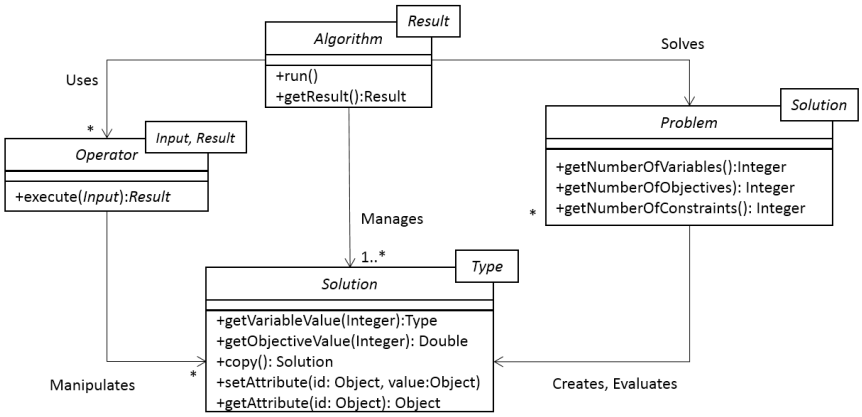

Fig. 1. Class diagram with core classes and interfaces of jMetal 5.

Algorithm 1 Template of a metaheuristic

1: $A(0) \leftarrow$ GenerateInitialSolutions()
2: $t \leftarrow 0$
3: Evaluate($A(0)$)
4: **while not** StoppingCriterion( ) **do**
5: $S(t) \leftarrow$ Generation($A(t)$)
6: Evaluate($S(t)$)
7: $A(t+1) \leftarrow$ Update($A(t)$, $S(t)$)
8: $t \leftarrow t+1$
9: **end while**

jMetal is an algorithmic framework, which includes a number of optimization metaheuristics of the state of the art [11]. It mostly centers in multi-objective optimization, although it also provides single-objective algorithms. In the work presented here, we use jMetal 5 [19], which follows the architecture depicted in Fig. 1. The underlying idea is that an algorithm (metaheuristic) manipulates a number of solutions with some operators to solve an optimization problem.

jMetal 5 provides algorithm templates mimicking the pseudo-code of a generic metaheuristic like that shown in Algorithm 1, where a set A of some initial solutions is iteratively updated by generating a set S of new solutions until a stopping condition is achieved. Another feature of jMetal is that it offers an interface (called `SolutionListEvaluator`) to encapsulate the evaluation of a list of solutions (i.e., a population in the context of evolutionary algorithms):

```
public interface SolutionListEvaluator<S> {
  List<S> evaluate(List<S> solutionList, Problem<S> problem);
  void shutdown();
}
```

The encapsulated behavior of this interface is that the method `evaluate` of the problem (see Fig. 1) is applied to all the solutions in the list, yielding to a new list of evaluated solutions. Metaheuristics using this interface can be empowered with different evaluator implementations, so that the current way of evaluating solutions is transparent to the algorithms. For example, many algorithms incorporate a method similar to this one (which corresponds to step 6 of the template shown in Algorithm 1):

```
protected List<S> evaluatePopulation(List<S> population) {
    population = evaluator.evaluate(population, problem);
    return population;
}
```

In this way, the actual evaluator is instantiated when configuring the settings of the metaheuristic, so no changes in the code are needed. jMetal 5 currently includes two implementations of `SolutionListEvaluator`: sequential and multi-threaded. Our approach has been then to develop an evaluator based on Spark.

Apache Spark [25] is based on the concept of Resilient Distributed Datasets (RDD), which are collections of elements that can be operated in parallel on the nodes of a cluster by using two types of operations: transformations (e.g., map, filter, union, etc.) and actions (e.g., reduce, collect, and count). The Spark based evaluator in jMetal creates an RDD with all the solutions to be evaluated, and a map transformation is used to evaluate each solution. The evaluated solutions are then collected and returned to the algorithm.

It is worth noting that algorithms do not need to be modified to use Spark, although the problem to be solved must fulfill the requirements imposed by this platform, as the algorithms run map processes. For example, the evaluate method of the problem must not modify variables outside the scope of the RDD containing the list of solutions to be evaluated.

Currently, five multi-objective metaheuristics in jMetal 5 use evaluators, so all of them can take advantage of the one based on Spark: NSGA-II [10], SPEA2 [27], SMPSO [20], GDE3 [14] and PESA2 [8]. This scheme can be also used in a number of single-objective algorithms: generational genetic algorithm (gGA), differential evolution (DE), and two PSO algorithms.

jMetalSP is a new project for Big Data Optimization with multi-objective metaheuristics [4] based on jMetal and Spark. It is currently intended to solve dynamic multi-objective optimization problems in Hadoop environments by using the streaming data processing capabilities of Spark, while jMetal provides the optimization infrastructure for implementing the dynamic problems and the dynamic algorithms to solve them. We have extended jMetalSP with the Spark evaluator, so it can be used also to solve non-dynamic optimization problems.

The attractive point of the adopted approach is that a number of single and multi-objective metaheuristics can be executed in parallel without requiring any modification. If we look closely at steps 3 and 6 in Algorithm 1 we can see that the set S can be evaluated in parallel by the Spark-based evaluator, so the resulting parallel model is a heartbeat algorithm: a parallel step is alternated with a sequential one (for the rest of the phases in the main loop of the metaheuristic).

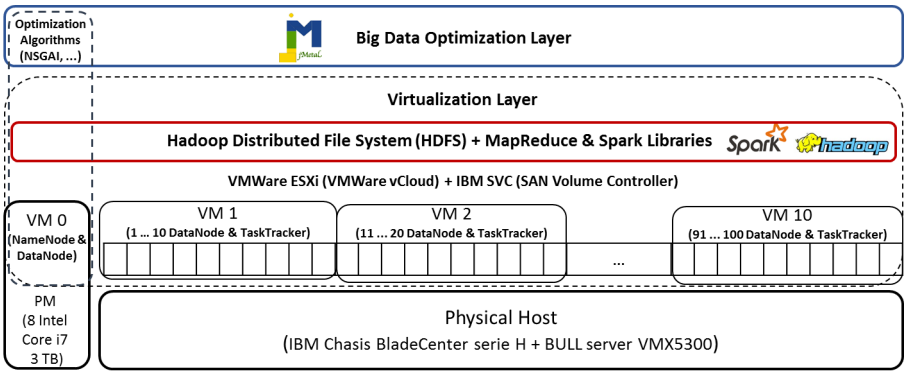

Fig. 2. Computational environment for Big Data Optimization used to test the performance of the proposed jMetal+Spark software solution, under different conditions of computational effort and Big Data management

This obviously prevents linear speedups, and this is the price to pay for having a very simple mechanism that uses jMetal's algorithms in a Big Data infrastructure. In this paper, our main interest is to measure the effective time reductions that can be achieved in different contexts.

4 Experimental Framework

To evaluate the performance of the proposed approach, a series of experiments have been conducted from three points of view: (1) computational effort, in terms of which we measure the performance of the parallel model; (2) data management, which is focused on testing the ability to manage a large number of data files; and (3) a combination of (1) and (2). Therefore, for each, we follow a different problem configuration involving: time consuming delays, different data block sizes, and different cluster sizes. We are seeking the maximum limit that a multi-objective algorithm can manage when handling Big Data without a loss in performance.

For this reason, with the aim of guiding us and to simplify this experimentation, we have centered on one optimization problem and one algorithm to solve it, although without emphasizing on the solution quality, as the behavior of the parallel algorithm is the same as its sequential counterpart. Specifically, we have selected the NSGA-II algorithm [10] and the multi-objective optimization problem ZDT1 [13].

From an algorithmic point of view, we have used a common parameter setting of NSGA-II to test our proposal in all the experiments. The variation operators are SBX crossover and Polynomial mutation, with crossover and mutation rates $p_c = 0.9$ and $p_c = 1.0/L$, respectively (L being the number of decision variables of the problem), and both having a distribution index value of 20. The selection

strategy is binary tournament and the population size has been set to 100 individuals. Finally, the stopping condition is met when the total number of 25,000 candidate solutions have been evaluated. In others words, the NSGA-II performs 250 evolution steps or iterations of the population throughout the running time.

All the experiments have been conducted in a virtualization environment running on a private high-performance cluster computing platform. This infrastructure is located at the *Ada Byron Research Center* at the University of Málaga (Spain), and comprises a number of IBM hosting racks for storage, units of virtualization, server compounds and backup services.

Our virtualization platform is hosted in this computational environment, whose main components are illustrated in Fig. 2. Concretely, this platform is made up of 10 virtual machines (VM1 to VM10), each one with 10 cores, 10 GB RAM and 250 GB virtual storage (adding up to 100 cores, 100 GBs of memory and 2.5 TB HD storage). These virtual machines are used as Slave nodes with the role of TaskTracker (Spark) and DataNode (HDFS) to perform fitness evaluations of algorithmic candidate solutions in parallel. The Master node, which runs the core algorithm (NSGA-II), is hosted in a different machine (VM0) with 8 Intel Core i7 processors at 3.40 GHz, 32 GB RAM and 3 TB storage space. All these nodes are configured with a Linux CentOS 6.6 64-bit distribution. The whole cluster is managed with Apache Ambari 1.6.1 and executes the Apache Hadoop version 2.4.0. This Hadoop distribution integrates HDFS and Apache Spark 1.6.

The jMetalSP framework is then deployed on this infrastructure, providing optimization algorithms with Spark methods to evaluate candidate solutions in parallel, in addition to managing HDFS files.

5 Experiments

This section describes the set of experiments and the analyses carried out to test our approach. We focus on the speedup and efficiency analysis in terms of computational effort and data management.

5.1 Speedup and Efficiency

One of the most widely used indicators for measuring the performance of a parallel algorithm is the *Speedup* (S_N). The standard formula of the speedup is represented in Equation 1 and calculates the ratio of T_1 over T_N, where T_1 is the running time of the analyzed algorithm in 1 processor and T_N is the running time of the parallelized algorithm on N processing units (processors or cores).

$$S_N = \frac{T_1}{T_N} \tag{1}$$

$$E_N = \frac{S_N}{N} \times 100 \tag{2}$$

Table 1. Experimental results of NSGA-II (jMetalSP) executed on 1, 10, 20, 50, and 100 cores with different time delays in each problem evaluation

	Running Time (hours)					Speedup				Efficiency			
Delay	T_1	T_{10}	T_{20}	T_{50}	T_{100}	S_{10}	S_{20}	S_{50}	S_{100}	E_{10}	E_{20}	E_{50}	E_{100}
10 s	76.50	10.90	8.10	5.00	3.50	7.02	9.44	15.30	21.85	70.20%	47.22%	30.60%	21.85%
30 s	216.70	34.10	24.40	16.30	10.80	6.35	8.88	13.29	20.06	63.50%	44.41%	26.59%	20.06%

A related measure is the *Efficiency* of a parallel algorithm, which is calculated with the formula of Equation 2. An algorithm scales linearly (ideal) when it reaches a speedup $S_N = N$ and hence, the parallel efficiency is $E_N = 100\%$.

5.2 Computational Effort

As stated, to measure the parallel computing performance of our approach we have used the NSGA-II algorithm to solve a modified version of the ZDT1 problem. The running time of NSGA-II with those settings in a laptop equipped with an Intel i7 processor is less than a second, so we have artificially increased the computing time of the evaluation functions of ZDT1 (by adding an idle loop) to simulate a real scenario where the total computing time would be in the order of several hours. After a number of preliminary experiments, we set the evaluation time to two values, 10 and 30 seconds; this way, we estimated the total running time of the sequential NSGA-II algorithm to be around 76.5 and 216.7 hours, respectively.

Table 1 shows the running time in hours used by the NSGA-II approach of jMetalSP running on 1, 10, 20, 50, and 100 cores, with regards to the two delays considered, applied in each solution evaluation. This table also contains the corresponding speedup and efficiency values to the resulting times. As mentioned, T_1 (one core) is 76.5 hours or 3.19 days in the case of the 10 seconds delay and 216.7 hours or 9.03 days with a delay of 30 seconds. As expected, these times are reduced in relation to the increase in the number of cores used in the parallel model. The highest reductions in time are obtained when our approach is configured with 100 cores in parallel, for which the running time is reduced to 3.5 hours (95.42%) in the case of a delay of 10 seconds and to 10.8 hours (95.02%) using a 30 seconds delay.

In terms of efficiency, the highest percentage 70.2% is reached with 10 cores and it decreases as the number of resources gets larger, to reach 47.22%, 30.6%, and 21.85% with 20, 50, and 100 cores, respectively.

This behavior was somewhat expected as the parallel model is based on alternating parallel and sequential steps. Considering the results, it is worth mentioning that both problem configurations yield similar speedup and efficiency values, which indicates that the bottleneck is due to both the parallel model and the parallel infrastructure, so increasing the evaluation time does not compensate the synchronization and communication costs.

Table 2. Experimental results of NSGA-II (jMetalSP) executed on 1, 10, 20, 50, and 100 cores with different block sizes management in each problem evaluation

	Running Time (hours)					Speedup				Efficiency			
Block	T_1	T_{10}	T_{20}	T_{50}	T_{100}	S_{10}	S_{20}	S_{50}	S_{100}	E_{10}	E_{20}	E_{50}	E_{100}
32MB	41.47	2.47	1.94	2.18	2.56	16.78	21.37	19.02	16.19	167.79%	106.8%	32.38%	16.19%
64MB	41.28	4.20	3.72	3.99	4.52	9.82	11.09	10.34	9.13	98.20%	55.45%	20.68%	9.13%
128MB	48.45	7.89	8.94	9.92	9.47	6.14	5.41	4.88	5.11	61.40%	27.05%	9.76%	5.11%

5.3 Data Block Size

We now turn to a number of experiments to measure the influence of using different data block sizes (file sizes) on the evaluation process of NSGA-II. For this purpose, we have set NSGA-II to manage data files with different sizes in each problem evaluation. This way, we have centered on file sizes of 32MB, 64MB, and 128MB; hence, each complete algorithm's execution manages a total volume of 0.8TB, 1.6TB, and 3.2TB, respectively. We have arranged a pool of files stored in HDFS totalling 1TB, so the pool size is 1TB/file size.

Once a problem solution has been evaluated, the optimization algorithm randomly (uniformly) selects a file from the pool. Neither additional computing tasks nor delays are performed in the evaluation step for this analysis.

Table 2 shows the running time in hours used by the NSGA-II approach of jMetalSP running on 1, 10, 20, 50, and 100 processors, for the three different configurations of file size. As in the first experiments, the sequential time T_1 is an estimation. In addition, this table contains the speedup and efficiency with regards to the resulting computing times. As we can observe, the running time with one core T_1 took approximately 1.72 days in the case of 32MB (41.47 hours) and 64MB (41.28 hours) file sizes, and it was close to 2 days (48.45 hours) in the case of 128MB. In fact, in the context of one processing unit, for the 32MB file size the optimization algorithm took longer than for 64MB, which could be due to the optimal block size in HDFS, which is 64MB [21] since no extra operations of either wrapping or splitting are required with this file size.

As expected, the running time is reduced in general when using a higher number of nodes in the parallel model. However, the highest reductions in running time are not obtained with 100 cores, but rather with 10 and 20. For the latter, the execution time (T_{20}) is 1.94 hours in the case of 32 MBfile size and 3.72 hours when using 64MB, which means a reduction of 95.32% and 90.98%, respectively, with regards to the algorithm running time in one processor (T_1). In the case of 128MB file size, the highest reduction in computing time (83.71%) is reached when running NSGA-II in parallel with 10 nodes (T_{10}). Although this behavior could be counter-intuitive, the fact of focusing only on data management without spending extra computational effort in evaluating the solutions, causes the network throughput in the parallel model to impair the overall performance of the optimization approach, since the greater the number of cores used, the higher the transference of data between nodes.

Table 3. Experimental results of NSGA-II (jMetalSP) executed on 1, 10, 20, 50, and 100 processors with different time delays and 64MB block size management in each problem evaluation

	Running Time (hours)					Speedup				Efficiency			
Delay	T_1	T_{10}	T_{20}	T_{50}	T_{100}	S_{10}	S_{20}	S_{50}	S_{100}	E_{10}	E_{20}	E_{50}	E_{100}
10 s	85.90	19.50	11.80	6.80	7.40	4.40	7.28	12.63	11.61	44%	36.40%	25.26%	11.61%
30 s	237.5	39.90	31.70	17.30	12.50	5.95	7.49	13.73	19.01	59.50%	37.45%	27.46%	19.01%

In terms of speedup, a number of 10 and 20 cores in the parallel model reach close to linear (even superlinear) speedup values, although they get worse with a higher number of nodes. In fact, the most efficient configuration is obtained with 10 nodes for the three file sizes. As our computational environment is composed of 10 virtual machines with 10 cores each (see Fig. 2), it is clear that the algorithmic configuration with 10 nodes is able to launch all jobs within the same machine, thereby avoiding having to use the underlying network and reaching the highest efficiency in terms of data management. The achievement of superlinear speedups is related to the application of data cache and file replication techniques in the underlying Hadoop system.

5.4 Computing Data Plus Access Performance

Following the two experiments in the context of parallel numerical performance and parallel data access performance, the next step is to combine both aspects to simulate a real Big Data optimization problem, which involves both heavy data access and data processing. Consequently, we have designed a new experiment in which each evaluation task requires reading a file of 64MB size and then simulating a computational effort of 10 and 30 seconds.

The resulting times, as well as speedup and efficiency metrics, are shown in Table 3. We can observe that the running times are different depending on the computational delay in the evaluations. With 10 seconds, the highest reduction in running time is not obtained with 100 cores, but with 50: the execution time (T_{50}) is 6.8 hours which means a reduction of 92.08% with regards to the one processor running time (T_1); the efficiency value is 25.26%. In the case of using a 30 seconds delay, the best running time is with 100 cores with a reduction of 94.73% with regards to one core. In this case, the efficiency value is 19.01%.

5.5 Discussions

As a general observation, in the scope of our experimental framework, Fig. 3 plots the running times consumed by our jMetalSP approach executed on 1, 10, 20, 50, and 100 cores with all the used configurations of computational delay (10 s and 30 s) and data management (32MB, 64MB, and 128MB), plus the experiment combining computational effort and data access.

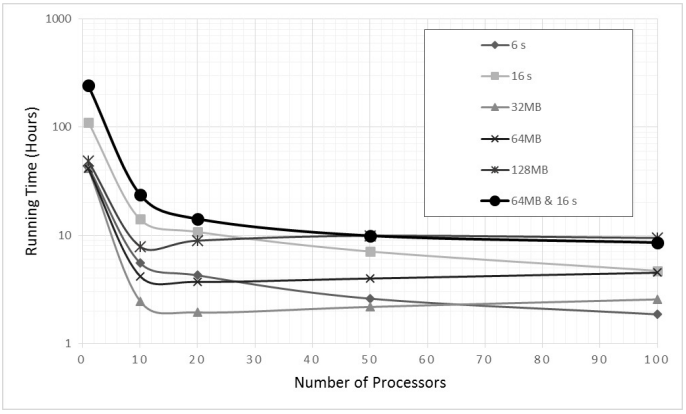

Fig. 3. Running time in hours (logarithmic scale) of jMetalSP version of NSGA-II executed on 1, 10, 20, 50 and 100 cores with all configurations of computational delay and data management

In this figure, it is clearly observable that the highest reduction in computing time is reached when the evaluation of solutions in the evolution process of NSGA-II entails a computational effort (30 s delay), as well as data management (64MB file size). So, a complete execution of the optimization algorithm requires several days (as much as 10) in a platform environment with fewer than 10 cores, although this computational time is reduced to less than a day when using more than 20 cores in our parallel model. We therefore can suggest that a good trade-off between performance and resource requirement is obtained with a configuration of jMetalSP in the range of 20 to 50 cores, since all jobs are finalized in less than half a day and it is 50% more cost efficient.

The results we have obtained in the experiments suggest demonstrates the systems limits when dealing with a real-world Big Data optimization problem. The stopping condition of the NSGA-II algorithm used is to compute 25,000 function evaluations, which is a usual value used to optimize the ZDT1 problem. However, in a real scenario the number of evaluations would be most certainly beyond that number.

If we consider the lowest times of the experiments carried out, they range from 1.86 hours (computational effort, 10 second delay, 100 cores) to 8.58 hours (combined experiment, 100 cores). Assuming that the algorithm would need to compute 100,000 function evaluations, the times would increase to 7.44 and 34.32 hours, respectively. While the first time could be acceptable, the second could be at the limit of an acceptable value. Consequently, augmenting the number of evaluations to the order of 1 million or more would go beyond a reasonable time in our Hadoop-based system, and specific algorithms should be designed instead of merely using the standard NSGA-II metaheuristic, e.g. by incorporating local

search methods or by designing problem-oriented operators to reduce the number of evaluations to achieve satisfactory solutions.

Finally, it is worth mentioning that a similar experiment has been carried out with SMPSO [20], a swarm-intelligence multi-objective approach, in the scope of the experimental framework used here. SMPSO performs a different learning procedure from NSGA-II, although they both use similar mechanisms in terms of solution evaluation and multi-objective operation, i.e., archiving strategy and crowding density estimator. As expected, SMPSO registered similar speedups to NSGA-II, which leads us to claim that our jMetalSP approach behaves efficiently with different optimizers.

6 Conclusions

In this paper, we have presented a study related to multi-objective Big Data optimization. Our goal has been twofold: first, we have extended the jMetalSP framework (which combines jMetal and Spark) in such a way that jMetal metaheuristics can make use of the Apache Spark distributed computing features almost transparently; second, we have studied the performance of running a metaheuristic based in our parallel scheme on three scenarios: parallel computation, parallel data access, and a combination of both. We have carried out experiments to measure the performance of the proposed parallel infrastructure in an environment based on virtual machines in a local cluster composed of up to 100 cores.

The results lead us to get interesting conclusions about computational effort and to propose guidelines when facing Big Data optimization problems:

- Our approach is able to obtain actual reductions in computing time from more than a week to just half a day, when addressing complex and time consuming optimization tasks. This performance has been obtained in the scope of an in-house (virtualized) computational environment with limited resources.
- In those experiments where we only focused on data management, without spending extra computational effort on evaluating the solutions, the overall performance of the parallel model is usually impaired when using a large number of nodes (more than 50), because of the network overhead and the increasing data transfer between nodes.
- A noteworthy trade-off between performance and resource requirements is obtained with a configuration of jMetalSP in the range of 50 to 100 cores.

As for future work, we plan to evaluate the performance of our proposal in the context of public in-cloud environments, with the aim of studying whether a different behavior in the optimization of Big Data problems is observed in these kinds of environments (compared with virtualized), or not. As a second line of future work, we intend to use the available optimization algorithms in jMetalSP to deal with real-world complex and data intensive optimization problems.

Acknowledgement

This work has been partially funded by Grants TIN2014-58304-R (Spanish Ministry of Education and Science) and P11-TIC-7529 (Innovation, Science and Enterprise Ministry of the regional government of the Junta de Andalucía) and P12-TIC-1519 (Plan Andaluz de Investigación, Desarrollo e Innovación). Cristóbal Barba-González is supported by Grant BES-2015-072209 (Spanish Ministry of Economy and Competitiveness).

References

1. S. Abdul-Rahman, A.A. Bakar, and Z.-A. Mohamed-Hussein. Optimizing Big Data in Bioinformatics with Swarm Algorithms. In *IEEE 16th International Conference on Computational Science and Engineering (CSE)*, pages 1091–1095, Dec 2013.
2. I. Aljarah and S.A. Ludwig. Mapreduce intrusion detection system based on a particle swarm optimization clustering algorithm. In *IEEE Congress on Evolutionary Computation (CEC 2013)*, pages 955–962, June 2013.
3. T. Spencer Angus and Y. Jin. Reconstructing biological gene regulatory networks: where optimization meets big data. *Evolutionary Intelligence*, 7(1):29–47, 2014.
4. C. Barba-González, A.J. Nebro, J.A. Cordero, J. García-Nieto, J.J. Durillo, I. Navas-Delgado, and J.F. Aldana-Montes. jMetalSP: a framework for dynamic multi-objective big data optimization. Submitted to Applied Soft Computing, 2016.
5. A. Cabanas-Abascal, E. García-Machicado, L. Prieto González, and A. de Amescua Seco. An item based geo-recommender system inspired by artificial immune algorithms. *Journal of Universal Computer Science*, 19(13):2013–2033, 2013.
6. C. Coello, G.B. Lamont, and D.A. van Veldhuizen. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc. 2nd Ed., NY, USA, 2007.
7. J.A. Cordero, A.J. Nebro, J.J. Durillo, J. García-Nieto, I. Navas-Delgado, J.F. Aldana-Montes, and C. Barba-González. Dynamic multi-objective optimization with jMetal and Spark: a case study. In *The Second International Workshop on Machine learning, Optimization and Big Data – MOD 2016*.
8. D.W. Corne, N.R. Jerram, J.D. Knowles, and M.J. Oates. PESA-II: Region-based selection in evolutionary multi-objective optimization. In *Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 283–290. Morgan Kaufmann, 2001.
9. M. Daoudi, S. Hamena, Z. Benmounah, and M. Batouche. Parallel diffrential evolution clustering algorithm based on mapreduce. In *6th International Conference of Soft Computing and Pattern Recognition (SoCPaR 2014)*, pages 337–341, 2014.
10. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
11. J. J. Durillo and A. J. Nebro. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.
12. K. Govindarajan, T.S. Somasundaram, V.S. Kumar, and Kinshuk. Continuous clustering in big data learning analytics. In *IEEE Fifth International Conference on Technology for Education (T4E)*, pages 61–64, Dec 2013.
13. E. Kitzler, K. Deb, and L. Thiele. Comparasion of multiobjective evolutionary algorithms: Empirical results. *Evol.Comput.*, 8(2):173–195, 2000.

14. S. Kukkonen and J. Lampinen. GDE3: The third evolution step of generalized differential evolution. In *IEEE Congress on Evolutionary Computation (CEC'2005)*, pages 443 – 450, 2005.
15. R. Lammel. Google's MapReduce programming model revisited. *Science of Computer Programming*, 70(1):1 – 30, 2008.
16. W. Lee, Y. Hsiao, and W. Hwang. Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC Systems Biology*, 8(1), 2014.
17. G. Luque and E. Alba. *Parallel Genetic Algorithms*. Springer-Verlag Berlin Heidelberg, 1st edition, 2011.
18. A.W. McNabb, C.K. Monson, and K.D. Seppi. Parallel PSO using MapReduce. In *IEEE Cong. on Evol. Comp. CEC 2007*, pages 7–14, Sept 2007.
19. A. J. Nebro, J. J. Durillo, and M. Vergne. Redesigning the jMetal multi-objective optimization framework. In *Genetic and Evolutionary Computation Conference (GECCO 2015) Companion*, pages 1093–1100, July 2015.
20. A.J. Nebro, J.J. Durillo, J. García-Nieto, C.A. Coello Coello, F. Luna, and E. Alba. SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In *IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MCDM 2009)*, pages 66–73. IEEE Press, 2009.
21. K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
22. W. Sun, N. Zhang, H.n Wang, W. Yin, and T. Qiu. PACO: A Period ACO Based Scheduling Algorithm in Cloud Computing. In *International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, pages 482–486, Dec 2013.
23. K. B. Tannahill and Mo. System of systems and big data analytics bridging the gap. *Computers and Electrical Engineering*, 40(1):2 – 15, 2014.
24. B. Wu, G. Wu, and M. Yang. A MapReduce based ant colony optimization approach to combinatorial optimization problems. In *8th Int. Conf. on Natural Computation (ICNC 2012)*, pages 728–732, May 2012.
25. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.
26. Z. Zhou, N.V. Chawla, Y. Jin, and G.J. Williams. Big data opportunities and challenges: Discussions from data analytics perspectives. *IEEE Computational Intelligence Magazine*, 9(4):62–74, Nov 2014.
27. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. In *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2002.