Applied Soft Computing xxx (2017) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

jMetalSP: A framework for dynamic multi-objective big data optimization

Cristóbal Barba-González [a], José García-Nieto [a], Antonio J. Nebro [a,*], José A. Cordero [b], Juan J. Durillo [c], Ismael Navas-Delgado [a], José F. Aldana-Montes [a]

[a] Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain
[b] European Organization for Nuclear Research (CERN), Switzerland
[c] Distributed and Parallel Systems Group, University of Innsbruck, Austria

ARTICLE INFO

Article history:
Received 30 May 2016
Received in revised form 25 April 2017
Accepted 1 May 2017
Available online xxx

Keywords:
Big data optimization
Multi-objective optimization
Metaheuristics
Software framework
jMetal
Apache Spark

ABSTRACT

Multi-objective metaheuristics have become popular techniques for dealing with complex optimization problems composed of a number of conflicting functions. Nowadays, we are in the Big Data era, so metaheuristics must be able to solve dynamic problems that may vary over time due to the processing and analysis of several streaming data sources. As this is a new field, there is a need for software platforms to solve dynamic multi-objective Big Data optimization problems. In this paper, we present jMetalSP, which combines the multi-objective optimization features of the jMetal framework with the streaming facilities of the Apache Spark cluster computing system. Thus, existing state-of-the-art multi-objective metaheuristics can be easily adapted to deal with dynamic optimization problems that are fed by multiple streaming data sources. Moreover, these algorithms can take advantage of the parallel computing features of Spark. We describe the architecture of jMetalSP and show how it can be used to solve a dynamic bi-objective instance of the Traveling Salesman Problem (TSP) based on New York City's real-time traffic data. We have also carried out an experimental study to assess the performance of the resultant jMetalSP application in a Hadoop cluster composed of 100 nodes.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

We are immersed in the era of Big Data, where applications must manage and analyze huge amounts of data that cannot be processed with traditional technologies. The volume of data is not the only feature of the data managed or analyzed in Big Data applications, since they also have to deal with a variety of heterogeneous data sources, most of them in streaming, and often delivering data at a great velocity. These data must also be validated and analyzed to produce a significant value to the end user [1]. These features comprise the so-called V's of Big Data: volume, velocity, variety, variability, veracity and value [2].

In this context, evolutionary algorithms and metaheuristics in general will play an important role in solving Big Data optimization problems. If we focus on multi-objective problems, i.e., those having two or more conflicting objectives, they can be found in many disciplines such as transportation, economics, medicine and bioinformatics, so Big Data variants of these problems will be common in the near future. For this reason, a key challenge nowadays is to adapt current metaheuristics to cope with Big Data optimization [3]. The ideal therefore is to have powerful tools that allow us to take advantage of the large amount of research on multi-objective optimization over the last 15 years, so as to be able to use classic and modern algorithms in combination with some Big Data platforms, thereby addressing this new class of optimization problems.

Another important issue is that some real-world multi-objective problems involve objectives, constraints, and parameters that can change over time. These problems are referred to as dynamic multi-objective optimization problems [4]. They are related to Big Data optimization as they can change due to data received continuously from different data sources.

The purpose of this paper is to present jMetalSP, a software platform for dynamic multi-objective Big Data optimization, which combines the features of the jMetal framework [5] for multi-objective optimization metaheuristics with the Apache Spark cluster computing system [6]. As the main requirements of our

* Corresponding author.
E-mail addresses: cbarba@lcc.uma.es (C. Barba-González), jnieto@lcc.uma.es (J. García-Nieto), antonio@lcc.uma.es (A.J. Nebro), j.cordero@cern.ch (J.A. Cordero), juan@dps.uibk.ac.at (J.J. Durillo), ismael@lcc.uma.es (I. Navas-Delgado), jfam@lcc.uma.es (J.F. Aldana-Montes).

http://dx.doi.org/10.1016/j.asoc.2017.05.004
1568-4946/© 2017 Elsevier B.V. All rights reserved.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

proposal, we have defined a series of features that jMetalSP should have:

- The framework must define and solve dynamic Big Data optimization problems in Hadoop/Spark clusters.
- The algorithms developed in jMetalSP must be easily adaptable to solve dynamic multi-objective optimization problems.
- The applications developed with jMetalSP must be able to include a number of different data sources by taking advantage of the streaming features of Spark.
- The framework must be easy to use, by hiding low level details as much as possible and simplifying the incorporation of different data sources.
- jMetalSP is an open source project,[1] so any interested researcher can use it freely. Furthermore, the feedback from users of the framework will help to improve and evolve it.

With the aim of testing jMetalSP, we have generated a realistic use case based on the application of a multi-objective metaheuristic (NSGA-II [7]) to solve the dynamic bi-objective Traveling Salesman Problem (TSP) [8]. This is a classical academic problem to which we have incorporated a combination of real nodes (street geo-locations) and real traffic data from the city of New York, together with simulated data that mimics streaming sources taken from Twitter API[2] and Apache Kafka.[3] A series of experiments and comparisons have been conducted on a Hadoop cluster in order to check the behavior of jMetalSP in terms of both computational performance and solution quality.

This paper is an extension of the work in Cordero et al. [9], where a very preliminary version of the architecture of jMetalSP was presented. Here the architecture is defined in more detail (see Section 4), including all the software components and how they are related. The target problem, the dynamic bi-objective TSP, was formulated in previous work with one data source and tested on a single computer whereas in the current work we use three data sources and the performance of the system is evaluated on a Hadoop cluster composed of 100 nodes. Another novelty is the inclusion of a new algorithm, MOCell [10], in addition to NSGA-II (see Section 6.4). This allows us to compare these two techniques in the scope of the same computational and data environment.

The remainder of this paper is organized as follows. Section 2 contains background concepts and related work. Sections 3 and 4 present the different software components and the architecture of jMetalSP, respectively. The case study for validation is presented in Section 5. Section 6 describes the experiments and analyses. A series of additional features of jMetalSP are included in Section 7. Finally, conclusions and future work are given in Section 8.

2. Background concepts and related work

A Multi-objective Optimization Problem (MOP) is characterized by having to minimize/maximize two or more functions or objectives which are in conflict with each other. This means that improving one objective implies a worsening in the others, so it is not usually a single solution for a MOP but rather a set of solutions known as a Pareto Optimal set (or simply Pareto set). As this set is optimal, there is no other solution capable of improving any of the solutions in the set in any of the objectives. The representation of the Pareto set in the objective space is the Pareto front, which, in general, is presented as a graph so that the problem expert, i.e., the decision maker, can choose the best trade-off solution.

Over the last 15 years, metaheuristics [11] have proven to be effective methods for solving MOPs. A subfamily of them in particular, the evolutionary algorithms, are now widely used to effectively solve real world MOPs [12]. A key reason for the popularity of these algorithms is the availability of software frameworks that have helped implement and use state-of-the-art multi-objective metaheuristics; some examples are jMetal [5], PISA [13], or Paradiseo-MOEO [14].

Most research on multi-objective optimization assumes that the MOPs are immutable or static, in the sense that they do not change during the optimization process. However, in some real-world scenarios the objective functions or the search space can vary over time [4], leading to dynamic MOPs that require dynamic algorithms to solve them.

According to [4], four kinds of DMOPs can be characterized:

- Type I: the Pareto Set changes, i.e., the set of all the optimal decision variables changes, whereas the Pareto front remains the same.
- Type II: Both Pareto set and Pareto Front change.
- Type III: Pareto set does not change, whereas Pareto front changes.
- Type IV: Neither the Pareto set nor the Pareto front change, although the problem can change.

In the context of Big Data optimization, the origin of changes in the structure of a given problem is data coming in streaming from different sources, such as files appearing in certain directory/ies, socket connections, Web services, Twitter flows and Kafka streams. This implies that not only dynamic multi-objective metaheuristics must be applied to solve these problems, but also different processing methods must be used to manage all the streaming data sources. This issue is even harder when dealing with Big Data environments, where a huge volume of heterogeneous data has to be accurately and quickly managed. Consequently, it may require the capability of parallel processing, preferably transparently and straightforwardly.

In the last few years, a number of approaches consisting in adapting metaheuristic techniques to work in parallel on Hadoop ecosystems have been proposed. These proposals are related to data mining or data management applications, such as a swarm intelligence method to optimize the feature selection in Big gene expression datasets [15], data partitioning in Big Databases [16], dimension reduction in Big Data analytics [17], pattern detection with Artificial Immune Algorithms [18], a parallel MapReduce evolutionary algorithm for graph inference [19], and a parallel artificial ant colony optimization for task scheduling in clusters environments [20]. Most of these approaches are based on the MapReduce programming model [21]. However, MapReduce entails a series of disadvantages that make it unsuitable to be properly integrated in global optimization techniques in general, and metaheuristics in particular. These drawbacks are mostly related to: high latency queries, non-iterative programming model, and weak real-time processing. Therefore, new challenging approaches are demanded to integrate Big Data based technologies with global optimization algorithms in order address all these issues.

Surprisingly, in terms of dynamic optimization, there is still a lack (to the best of our knowledge) of applied algorithms to the optimization of problems in real-time updating Big Data environments, which is a crucial feature in those applications in which high velocity and variety of incoming data have to be properly managed.

In this paper, we propose a framework that combines jMetal with Apache Spark, thereby addressing dynamic Big Data optimization problems in an almost transparent way, hence avoiding the intrinsic shortcomings of the usual MapReduce model when applied to global optimization. In successfully meeting this con-

[1] jMetalSP project in GitHub: https://github.com/jMetal/jMetalSP.
[2] Available from URL https://dev.twitter.com/overview/api.
[3] Available at URL http://kafka.apache.org/.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

crete objective, we will show the feasibility of a new software framework for Big Data Analytics which will be able to cope with scaling and dynamic optimization techniques. In addition, we generate a dynamic multi-objective algorithm to optimize a real-world bi-objective TSP instance in a changing traffic environment, by managing the ingestion of multiple data sources, in order to benefit from them.

3. jMetalSP: software components

jMetalSP is a software that has two components: First, the jMetal multi-objective optimization framework, which provides the optimization infrastructure for implementing both the Big Data problems and the dynamic metaheuristics to solve them; second, the Apache Spark parallel computing system, which is used to manage the streaming data sources, enabling the computing power of Hadoop clusters to be effectively used when processing large amounts of data. Spark also allows storing and retrieving data to/from the HDFS file system. The main features of these two components are described in the following subsections.

3.1. jMetal

jMetal is a Java-based open-source framework for multi-objective optimization with metaheuristics [5]. It started life as part of project begun in 2006 with the objective to design a tool to assist in multi-objective optimization research by a research group at the University of Málaga. It was made publicly available in 2008 at SourceForge[4] and since then, it has become a popular tool in the field. Some reasons for this are its object-oriented architecture, which makes it easy to understand and use, together with the inclusion of many multi-objective algorithms of the state-of-the-art, as well as most of the benchmark problems that are used in many studies.

The framework was completely re-designed in 2015, and it was moved to GitHub,[5] the full project being available at https://github.com/jMetal/jMetal. This version, jMetal 5, has many improvements over previous releases, including a new architecture, an improved solution representation scheme, an intensive use of Java generics to avoid many run-time errors, and a set of algorithm templates to foster code reusing and testing [22].

The architecture of jMetal 5 is generic enough to allow much needed flexibility to implement any metaheuristic, but most of the algorithms belong to well-established subfamilies, such as evolutionary algorithms, particle swarm optimization and scatter search to name but a few. These subfamilies are characterized by a common behavior that is shared by all the algorithms belonging to them. jMetal 5 provides a number of algorithm templates that include the behavior of a base metaheuristic subfamily, so developing a particular algorithm only requires implementing some specific methods. An example is shown in Algorithm 1, which represents evolutionary algorithms.

Algorithm 1. Pseudo-code of an evolutionary algorithm.

```
1:  P(0) ⟵ GenerateInitialSolutions()
2:  t ⟵ 0
3:  Evaluate(P(0))
4:  while not StoppingCriterion() do
5:    Q(t) ⟵ Variation(P(t))
6:    Evaluate(Q(t))
7:    P(t+1) ⟵ Update(P(t), Q(t))
8:    t ⟵ t+1
9:  end while
```

[4] http://jmetal.sourceforge.net.
[5] http://jmetal.github.io/jMetal/.

jMetal's evolutionary algorithm template closely mimics this pseudo-code, which is used to implement popular multi-objective techniques, such as NSGA-II [7], SPEA2 [23] or SMS-EMOA [24].

Using algorithm templates fosters code reusability (e.g., to implement an algorithm variant only those methods differing from the base algorithm have to be written). This feature is exploited by jMetalSP to facilitate the development of dynamic versions of multi-objective metaheuristics (see Section 5.1).

3.2. Apache Spark

Apache Spark is a general-purpose distributed computing system [6] based on the concept of Resilient Distributed Datasets (RDDs). RDDs are collections of elements that can be operated in parallel on the nodes of a cluster by using two types of operations: transformations (map, filter, union, etc.) and actions (reduce, collect, count, etc.). Notable features of Spark are: high level parallel processing programming model, machine learning algorithms, graph processing, multi-programming language API, runs on different systems (Hadoop, Mesos, standalone, cluster), and streaming processing. Spark is becoming highly popular and is replacing MapReduce as the dominant technology for developing Big Data applications.

As jMetalSP is intended for Big Data optimization, we are interested in Spark's stream processing capabilities; in particular, the ability to include different data sources (Kafka, Flumme, Twitter, TCP sockets, files, etc.) and process them in parallel. The basic abstraction used in Spark is the discretized stream (DStream), which is a sequence of RDDs. The idea is that input streams are discretized in small batches, which are then processed by the Spark engine.

4. jMetalSP architecture

The conceptual architecture of jMetalSP is depicted in Fig. 1. A jMetalSP application is intended to solve a dynamic optimization problem with a dynamic algorithm by analyzing multiple streaming data sources, so there will be one or more data consumers that obtain the results being computed by the algorithm.

The central point of the architecture is the dynamic problem, which is basically a jMetal problem, but endowed with methods to:

1. Change some of its components to meet the requirements of the processing and analysis of the streaming data sources.
2. Allow the dynamic algorithm to detect changes and to react according to them (for example, by applying a re-start strategy).

The processing of streaming data sources use the Spark features mentioned in the previous section. This processing involves the reception of data that appears continuously and must be analyzed. The analysis is usually a costly operation (e.g., detecting significant changes in a given context after performing a sentiment analysis of information from a social network), and it can be carried out automatically in parallel by the Spark runtime by taking advantage of the underlying distributed cluster computing system. This can be based not only on Hadoop, but also on different platforms like Apache Mesos or the stand-alone mode offered by Spark.

The dynamic algorithm is, as the dynamic problem, a jMetal algorithm modified to be able of checking the state of the problem and to react when the changes happen. The dynamic algorithm is executed in a separate thread, in parallel with the streaming data source processing entities. Compared with a standard static algorithm, which ends after the stopping criterion has been fulfilled, the dynamic algorithm runs forever, producing result data (e.g., a

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

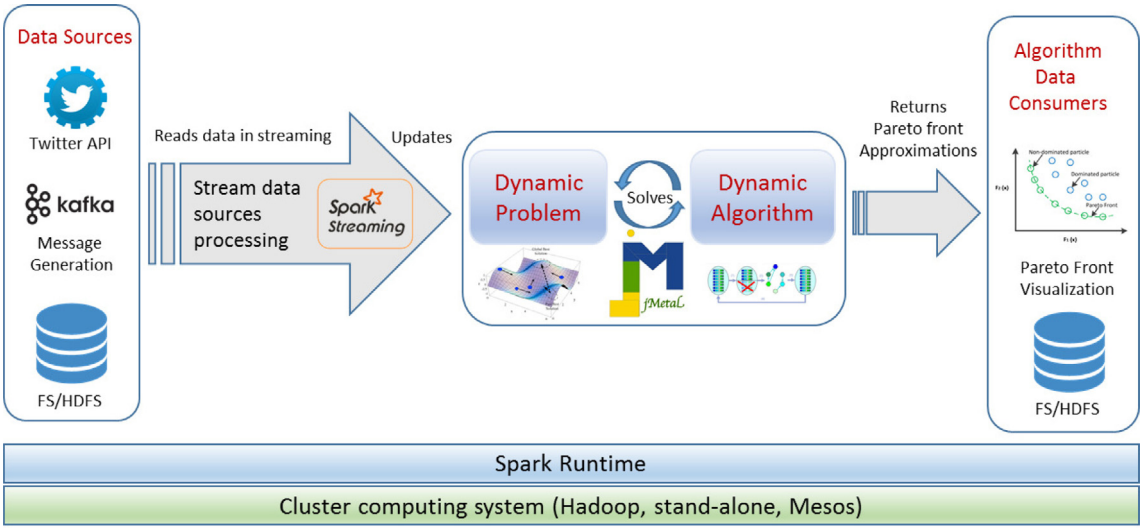

Fig. 1. Architecture overview of jMetalSP.

Pareto front approximation). These data are intended to be received by one or more algorithm data consumers. These components can perform simple tasks, such as merely plotting the fronts, or they can be complex applications that allow to analyze the output solutions by a decision maker.

It is worth noting that we are omitting implementation details to simplify for the sake of a better understanding of the overall package, but it is worth noting that the actual code is structured with Java generics to ensure all the components are compatible at compiling time.

5. Case study: dynamic bi-objective TSP with NSGA-II

To evaluate our proposed architecture we have defined a case study that combines an academic problem with real-world data. The chosen problem is a dynamic version of the bi-objective Traveling Salesman Problem (TSP), to minimize the "*travel time*" and the "*distance*" to cover all the points of the instance. The algorithm for solving it is a dynamic variant of the well-known multi-objective metaheuristic NSGA-II [7].

Our purpose is twofold: first, we intend to show how jMetalSP can be used to implement both the algorithm and the problem; second, we want to carry out a performance study while solving the problem on a Hadoop cluster composed of 100 cores. The quality of the obtained Pareto front approximations are not relevant, because, as we will see later in this section, two of the streaming sources generate artificial data.

5.1. Dynamic NSGA-II

The strategy followed to implement a dynamic version of NSGA-II is to consider the implementation already provided by jMetal 5, which follows the pseudo-code shown in Section 3.1. Then, we must focus in those methods that need to be modified or added to allow the dynamic behavior.

In this regard, we have made two modifications:

- Redefine the *StoppingCriterion()* method: in the standard NSGA-II algorithm, when the predefined maximum number of evaluations are computed, the algorithm ends. However, in the dynamic version, the algorithm must notify the algorithm data consumers that a new Pareto front approximation has been found. Then, the algorithm is restarted and begins another execution.
- Add a further step at the end of each iteration: After the evaluation counter has been updated (Step 8 in Algorithm 1), the algorithm must check whether the dynamic problem has changed or not and, if so, a restart have to be carried out.

This approach to develop dynamic algorithms can be easily used with most of the metaheuristics included in jMetal 5.

5.2. Dynamic bi-objective TSP

A dynamic problem in jMetalSP is simply a jMetal 5 problem but including a method to make changes in the problem. In the case of the dynamic bi-objective TSP, which is formulated in terms of a distance matrix and a time travel matrix, the changes can affect any of them.

Our particular dynamic TSP problem instance is based on real data. Specifically, it is defined from the Open Data provided by the New York City Department of Transportation, which updates traffic information several times per minute.[6] The information is provided as a text file where each line includes the average speed to traverse the two end points defining a link in the most recent interval.

After processing the links provided by the traffic service (see detailed information in [9]), the resulting dynamic TSP problem is composed of 93 locations with 315 communications between them, which are depicted in Fig. 2. We note that the links are bi-directional, so the resulting TSP is asymmetric.

5.3. Streaming data sources

In our case study, we use three streaming data sources that are managed with Spark: directory, Twitter and Kafka. We give some details in this section.

New York's traffic data is read periodically by an external application that writes a file in a directory, whenever new data are acquired, so we have implemented a streaming data component for that purpose. This component reads periodically the new data

[6] At the time of writing this paper, the data can be obtained from this URL: http://207.251.86.229/nyc-links-cams/LinkSpeedQuery.txt.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

Fig. 2. Real dynamic TSP instance: 93 nodes of the city of New York.

appeared in the directory (this is done automatically by Spark) and makes a simple processing: if a change in a link is detected (time or distance), then the corresponding problem matrices are updated.

As commented in Section 4, the analysis of the streaming data sources can be carried out in parallel by using Spark. In our case, this processing is very simple and as the external process updates the directory two or three times per minute, there is no benefit to using parallelism. Therefore, we have implemented two other streaming data sources, based on Twitter and Kafka, to look in greater depth at this issue. In the first one, tweets are read from Twitter with the topic *New York traffic* and a processing of each tweet is simulated and, after that, the problem is updated with some random value. This way, we combine a different streaming source with the possibility of adjusting the processing time, which will serve for performance evaluation purposes (see next section). In the second one, as is the previous one, the idea to enrich the case study with another data source that will produce artificial data. Here, we have created a Kafka message producer, which generates, following uniform and normal distributions, a series of random messages with data to update the problem. Every 5 s at least 1000 messages are produced, but on average about 10,000 messages are created.

Both the Twitter and Kafka streaming source classes have the same behavior as the directory based one: they iteratively collect and analyze the data and possibly update the problem.

5.4. Data consumers

The last components of the dynamic TSP case study are the consumers of the data generated by the dynamic metaheuristic.

We have developed two data consumers: ones that stores the produced Pareto fronts in a directory, and other one that can print some information about the front (as the number of solutions and the number of generated fronts).

6. Experiments

In this section, we describe the experimental study undertaken to evaluate the performance of the dynamic bi-objective TSP application developed with jMetalSP. However, first we present the computational environment and parameter settings.

6.1. Computational environment

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

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

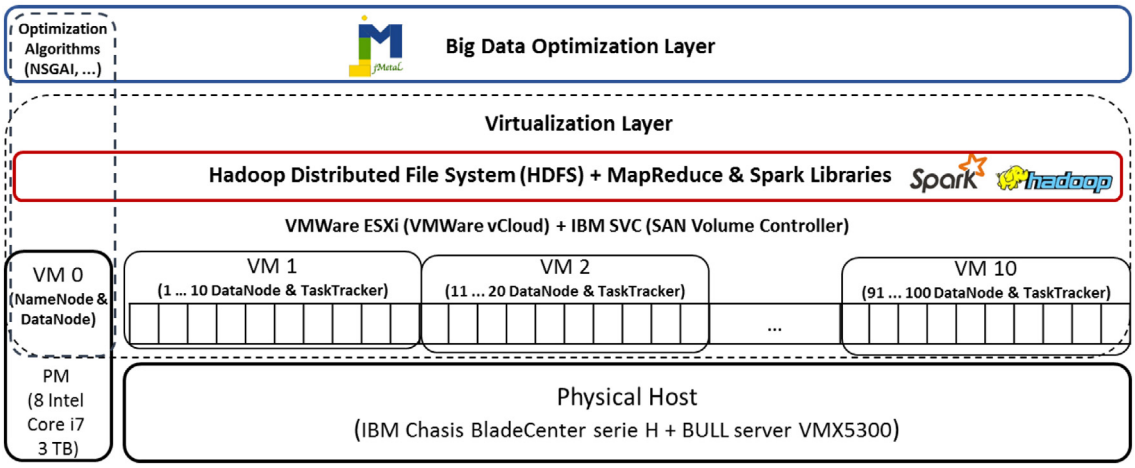

Fig. 3. Computational environment for Big Data optimization used to test the performance of jMetal-Spark under different conditions of computational effort and big data management.

The physical platform comprises an IBM Chassis BladeCenter H type and BULL server with a high-performance VMX5300, unified CPU, memory and storage. It also contains a NovaScale Blade BL265 only for storage. The virtualization layer supports computing resources through VMWare ESXi (and VMWare vCloud), storage through IBM SVC (SAN Volume Controller), backup through Veem Backup and IBM Protectier (virtualization of tapes) and desktops are managed with VMWare Horizon and Virtual Cable UDS. The general characteristics of this virtualization installation are: a processor with 256 cores and 578.94 GHz, 2.75 TB RAM, 84.41 TB storage space, 10 Gbps LAN network, 1 Gbps internet network, iSCSI 10 Gbps and FC 8 Gbps storage network.

The virtualization platform is hosted in this computing environment, whose main components are illustrated in Fig. 3. 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 (summing up 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 (Dynamic 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 Hadoop Distributed File System (HDFS), MapReduce framework libraries, and Apache Spark v1.6.1.

6.2. Parameter settings

The parameter settings of the dynamic NSGA-II algorithm are as follows: the population size is 100, the crossover operator is PMX (applied with a 0.9 probability), the mutation operator is swap (applied with a probability of 0.2), and the algorithm computes 100,000 function evaluations before writing out the front found and re-starting. We set the stop condition to 1 h, ingesting in this time an average number of messages per second in the range of [2000···20,000].

6.3. Computational performance

We have organized the experiments in to three different tests according to the way the streaming data is generated and consumed: *Uniform Moderate*, *Uniform Intensive*, and *Normal Moderate*. Our aim is to stress all the phases of the proposed approach in order to obtain insights into its actual performance. Therefore, each experimental test comprises a complete execution from which we have captured the traces of the Ganglia v3.6.2 monitoring service,[7] consisting in: data ingestion metrics, computational effort in terms of CPU resources (*Load_One*), and memory usage.

As explained in Section 5.2, in addition to data from Twitter and NY Traffic Files (which are stored in HDFS), we have incorporated a Kafka message producer which generates, following uniform and normal distributions, a series of random messages (4 KBs) with data to update the problem. Every 5 s, the Kafka producer delivers at least a number of 1000 messages, although computing an average number of 10,000 messages. Formally, the data production is formulated as follows:

$$D(t) = K(t) + T(t) + F_{NY}(t) \quad (1)$$

$$K(t) = \Phi \times \Omega(t), \quad \text{with} \quad \Omega \in \{U(0, Max), N(\mu, \sigma)\} \quad (2)$$

where $D(t)$ denotes the set of data produced each time (t) sequence of 5 s, $K(t)$ is the message generation procedure of Kafka, which is calculated according to Φ, a constant rate of message generation, and $\Omega \in \{U, N\}$ is a random number generator that can follow uniform or normal distributions. $T(t)$ denotes those messages obtained from a Twitter source (with queried key word: "*New York Traffic*"), and $F_{NY}(t)$ are the data files in HDFS with information concerning the traffic in the New York city TSP instance.

6.3.1. Test 1. Uniform moderate

In this test case, $\Phi = 1000$ mps (messages per second) and $\Omega = U$ is a uniform random generator with an upper limit of $Max = 100$. This implies a moderate uniform data flow of 50,000 incoming messages i on average, every 5 s for 1 h, which adds up to close on 140 GBs of data managed over the observation time.

Fig. 4 shows a snapshot (15 min) of plots concerning the input streaming data the dynamic NSGAII consumes in the initial running steps, on the Hadoop cluster, until it stabilizes. We can observe how the processing time is high in the initial minutes of execution, as it is subjected to the scheduling delay required to deploy all running tasks in the computational framework. In this scenario, the input data rate is stable, so not excepting some sporadic peaks (see the

[7] Ganglia Monitoring System. In URL http://ganglia.info/.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

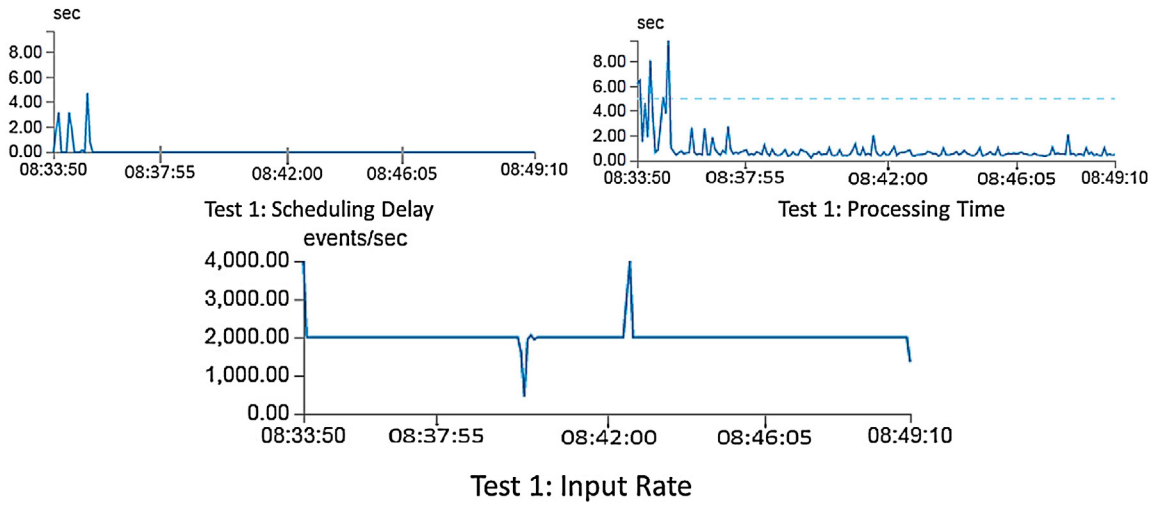

Fig. 4. Test 1. Scheduling delay, processing time, and input data during execution.

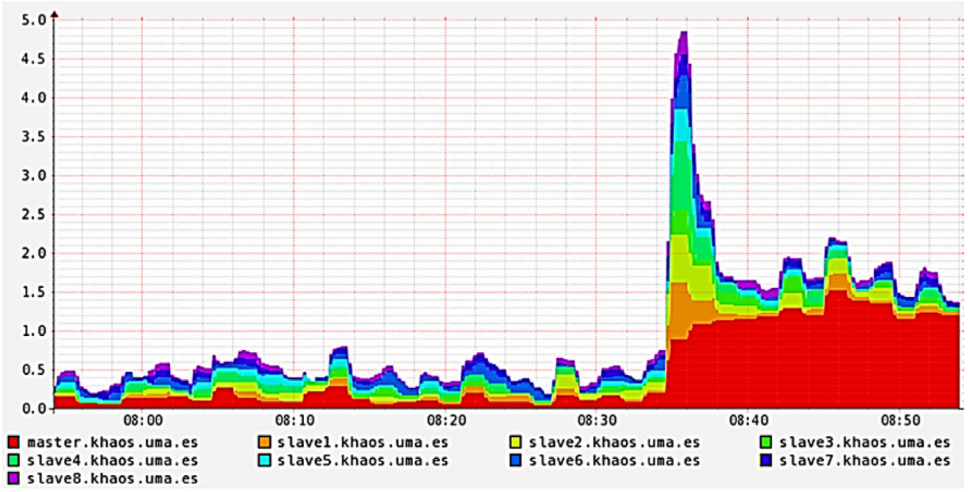

Fig. 5. Test 1. Load_one. Number of threads per node.

plot at the bottom of Fig. 4), the optimization algorithm is able to process the incoming data successfully, hence avoiding additional system overheads.

In terms of computational effort, we have plotted the *Load_one* measure of the entire cluster in Fig. 5, in order to check the overall CPU load. In particular, the *Load_one* computes the number of threads at kernel level that are running and being queued while waiting for CPU resources, averaged over the last minute. We could interpret this number in relation with the number of hardware threads available on the machine and the time it takes to drain the run queue. Fig. 5 captures a short time period (close to minute 8:35) in which the master node (Spark driver) delivers tasks to the slave nodes and they start to undertake data processing jobs.

This short system overload is also reflected in the memory usage level in Fig. 6, where we can observe how the global memory consumption scales from 500 GBs to 600 GB from minute 8:35 to 9:10. However, in this period, the use of *swap* and *buffer* memories remain constant, which prevents the system from collapsing. After this operation, both CPU and memory usage recover their moderate states and proceed properly. In other words, the dynamic NSGAII's performance is stable throughout the remaining optimization procedure.

6.3.2. Test 2. Uniform intensive

In Test 2, we increase the data injection in sources, so that $\Phi = 1000$ mps and $\Omega = U$ is a Uniform random generator with an upper limit of $Max = 1000$. This implies a intensive uniform data flow of 500,000 incoming messages on average, every 5 s for 1 h, which adds up to close on 1400 GBs (1.37 TBs) of managed data over the observation time. Our purpose in this test is to stress the data processing environment, in order to find the operation limits of the jMetalSP application.

In this case, as we can observe in Fig. 7, the scheduling delay scales so that in less than 10 min of execution, the delay is close to 12 min, probably due to buffering operations. The *Load_one* measure in Fig. 8 shows an increasing activity in minute 9:40, although lower than in the previous tests. As expected, the tasks scheduling step is lengthily delayed, which directly affects the jobs, delivery, and hence the slave nodes proceed to accept jobs.

This issue is clearly observable in terms of memory usage. As recorded in Fig. 9, the use of memory scales up to the limit of 600 GBs, therefore causing a hard overload in the computing platform.

With this in mind, we carried out a second round of tests by decreasing the constant rate of messages to $\Phi = 100$ mps, although

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

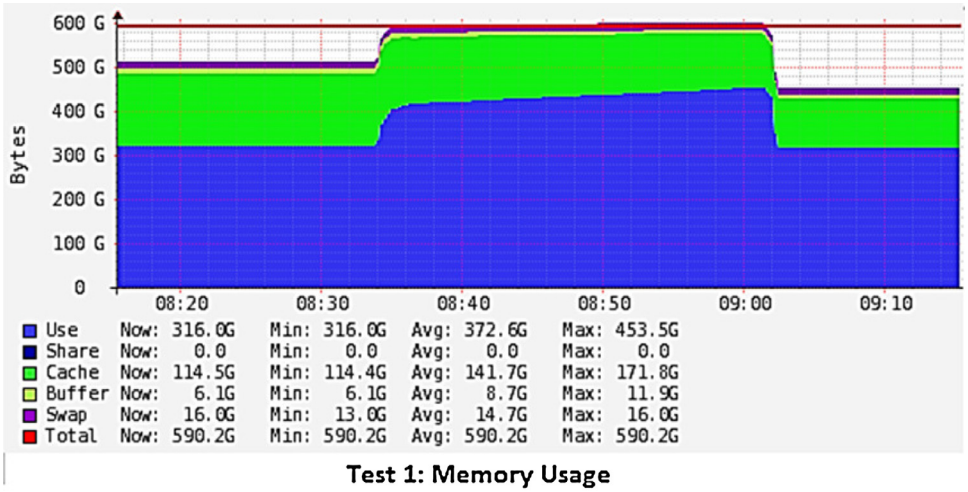

Fig. 6. Test 1. Memory usage level.

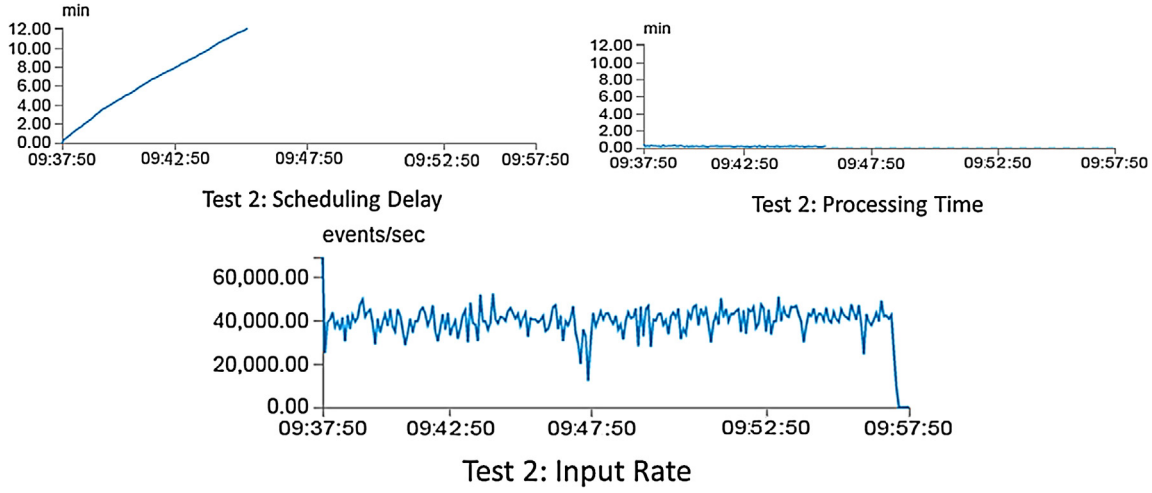

Fig. 7. Test 2. Scheduling delay, processing time, and input data during execution.

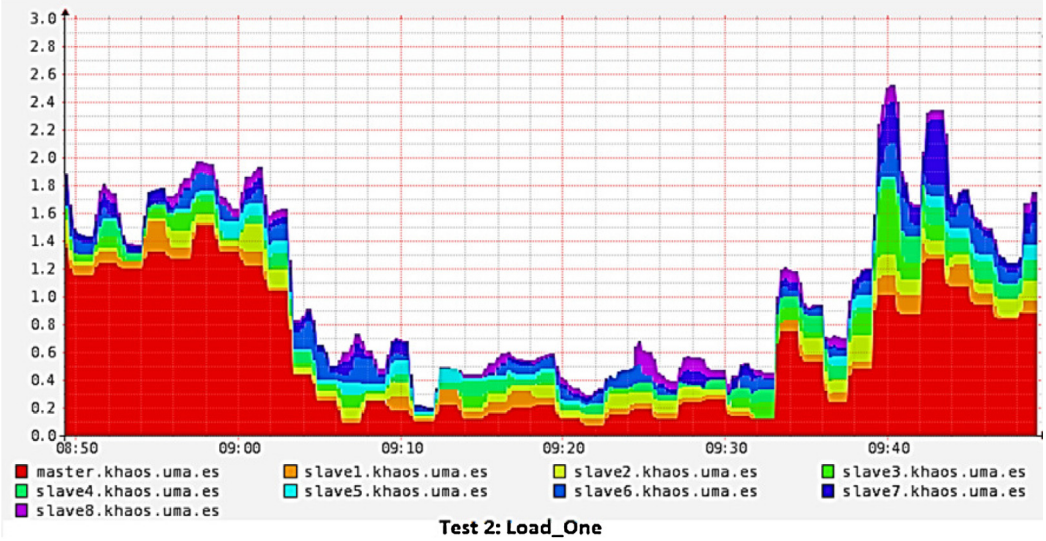

Fig. 8. Test 2. Load_one. Number of thread per node.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

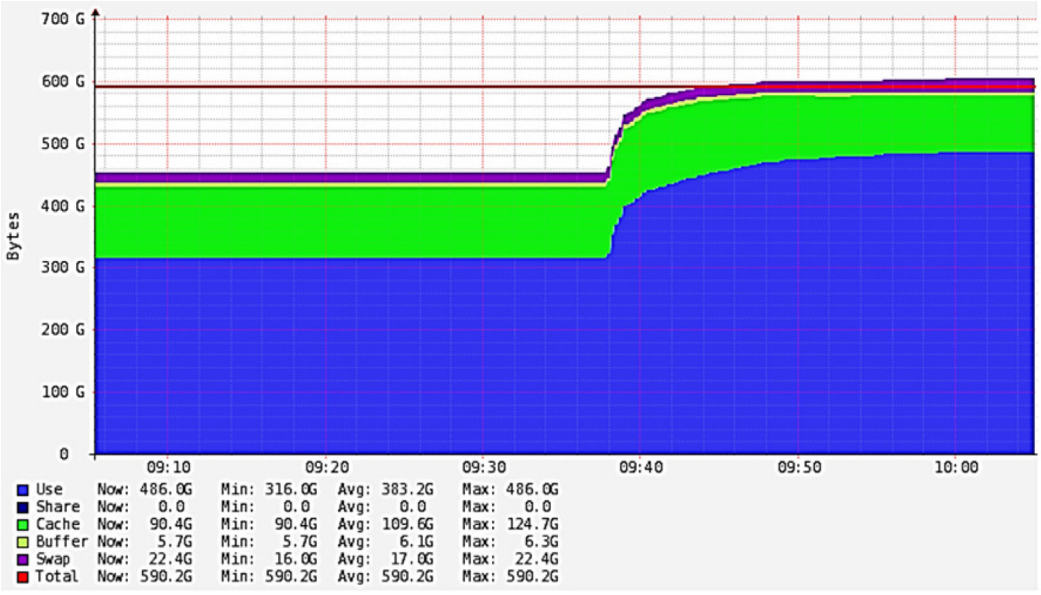

Fig. 9. Test 2. Memory usage level.

keeping $\Omega = U$ and $Max = 1000$. Interestingly, even though the average number of managed messages was similar to Test 1 (50,000), the platform was overloaded. We suspected that the uniform (random) generation of messages could produce several consecutive operations with large amounts of input data, which would provoke the memory overflow. This behavior led us to set out the third test as follows.

6.3.3. Test 3. Normal moderate

In Test 3, the input messages follow a normal distribution with the aim of better balancing the global data ingestion. This way, we look to find a good trade-off so that the dynamic NSGA-II algorithm can periodically read the data without additional computational overheads. Now the setup is $\Phi = 1000$ ms and $\Omega = N$ is a normal distribution generator with average $\mu = 100$ and standard deviation $\sigma = 10$. This implies an intensive normal data flow close to 500,000 incoming messages on average, every 5 s for 1 h (adding up to close on 1.37 TBs of managed data over the observation time).

Fig. 10 shows the input rate of data during the execution, where we can also observe how the scheduling delay increases, but it becomes stable in the range of [2.00···3.00] min. In this scenario, as captured in the plot of Fig. 11, the master node is able to schedule and deliver all jobs to the slave nodes, which consume data processing tasks properly. In this regard, as plotted in Fig. 12, the total memory registers an intensive use, but on average lower than 560 GBs, which indicates an efficient usage of the available computational resources.

To summarize, Table 1 contains the minimum, average and maximum values (in GBs) of memory usage, for each type of memory in the cluster and for each experimental test. We highlight that the use of Swap memory in Test 3 is higher than in Test 1 and 2, whereas the use of the Cache and Buffer memories is lower in Test 3 than in the other tests. This is a desirable behavior, since in the case of Test 3 the number of incoming messages is steady, so jMetalSP is able to compute them. As a result, when focusing on the total memory usage, we can observe that Test 1 and Test 2 approach's the global memory system capacity (600 GBs), which does not occur in Test 3, and therefore system overload is prevented.

6.4. Optimization results

Finally, in this section we report some interesting results with regards to the optimized solutions reported by the jMetalSP application using the dynamic NSGA-II algorithm. Fig. 13 shows the Pareto front approximations generated by the evolutionary algorithm throughout the optimization process after 10, 50, and 100 operations of problem data update. It is worth noting that each problem update implies an algorithm restarting.

We can observe that the shapes of Pareto front approximations vary in time, which is an expected behavior since we look for a reactive approach that, taking advantage of the information coming from multiple sources of data, is able to adapt itself to the changing conditions in an traffic environment. In the three depicted fronts, the extreme solutions are similar, which means that the data variations in the time slot where the data have been received have not been enough to alter the best and worst solutions for each objective.

It is worth mentioning that although we have focused on NSGA-II for this study, many of the metaheuristics included in jMetal can be adapted to jMetalSP in the same way as for NSGA-II. In order to show this, we have repeated the experimental procedure performed with NSGA-II, but in this case using a different optimizer, MOCell [10] (available in jMetalSP), which is a cellular multi-objective evolutionary algorithm. Fig. 14 shows the Pareto front approximations generated by the MOCell throughout the optimization processes, after 10, 50, and 100 operations of problem data updating. In general, we can observe that MOCell is also able to obtain a varied set of non-dominated solutions throughout the different steps of dynamic optimization. Nevertheless, in comparison with NSGA-II, the solutions obtained by MOCell show a similar performance in terms of distance, but with higher values of travel time. This indicates that NSGA-II performs better than MOCell in the scope of the data instances and the experimental framework used here.

7. Additional features of jMetalSP

In the preceding sections we have described and used jMetalSP to solve a dynamic Big Data optimization problem. However,

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

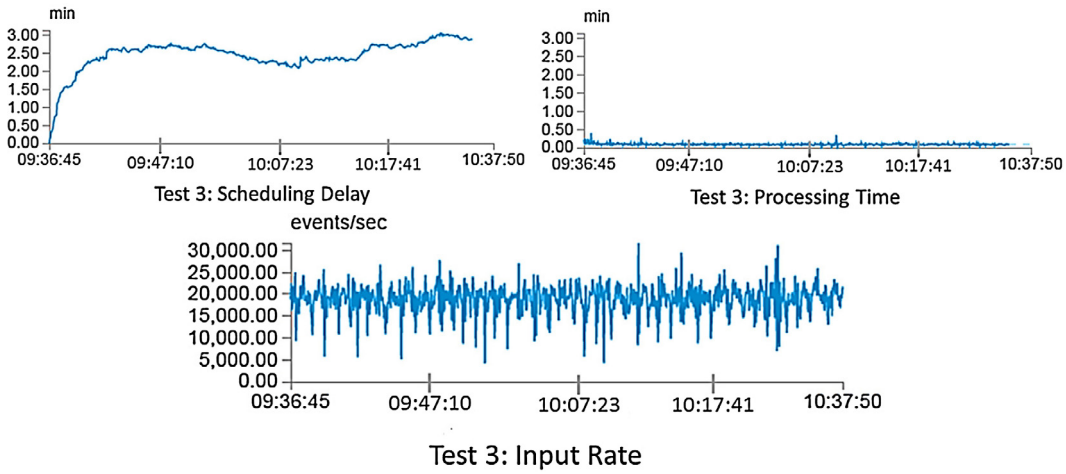

Fig. 10. Test 3. Scheduling delay, processing time, and input data during execution.

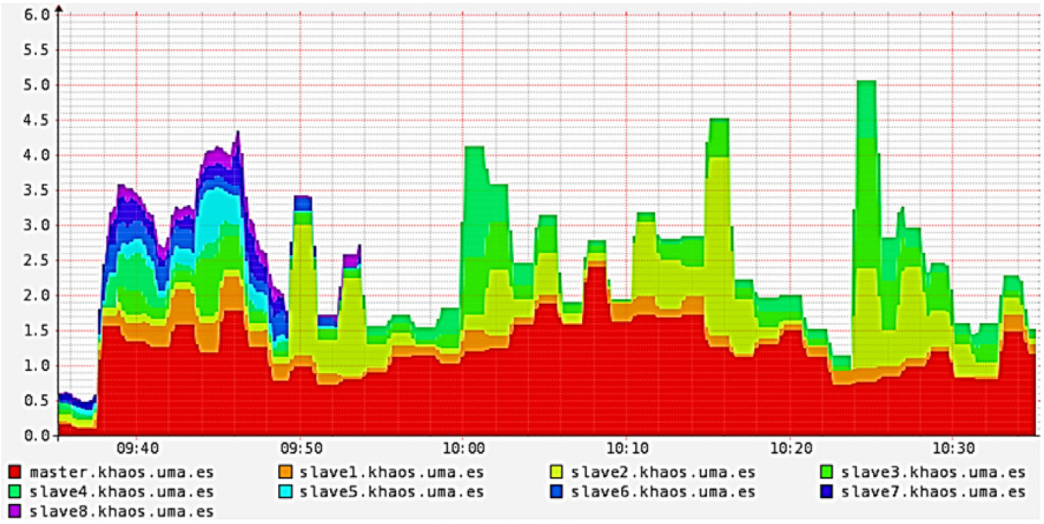

Fig. 11. Test 3. Load_one. Number of threads per node.

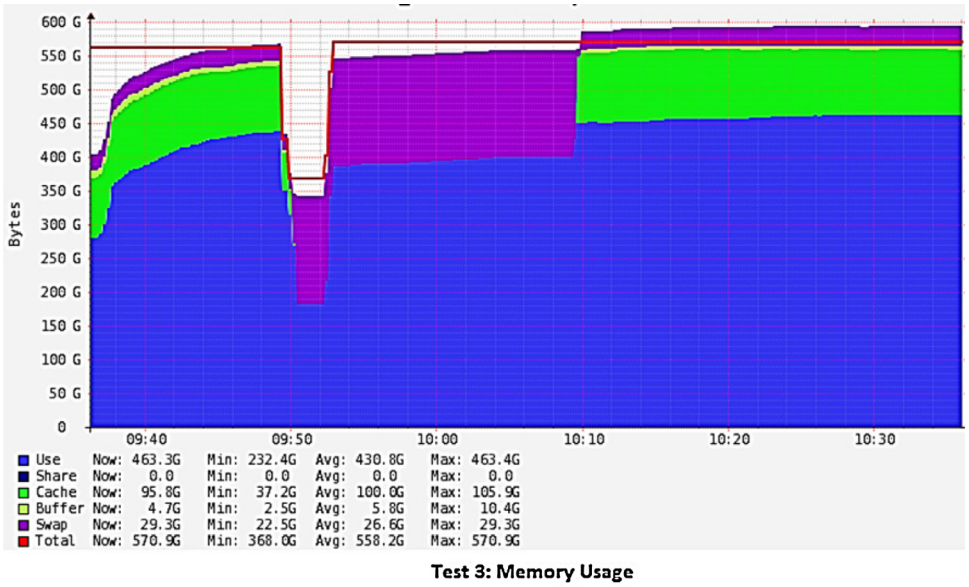

Fig. 12. Test 3. Memory usage level.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

Table 1
Minimum (Min), average (Avg), and maximum (Max) values in GigaBytes of memory usage throughout Test 1 (uniform moderate), Test 2 (uniform intensive) and Test 3 (normal moderate) with jMetalSP running NSGA-II.

Tests/type of memory	Test 1			Test 2			Test 3		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
Use	316.0	372.6	453.5	316.0	383.2	486.0	232.4	430.8	463.4
Cache	114.4	141.7	171.8	90.4	109.6	124.7	37.2	100.0	105.9
Buffer	6.1	8.7	11.9	5.7	6.1	6.3	2.5	5.8	10.4
Swap	13.0	14.7	16.0	16.0	17.0	22.4	22.5	26.6	29.3
Total	590.2	590.2	590.2	590.2	590.2	590.2	368.0	558.2	570.9

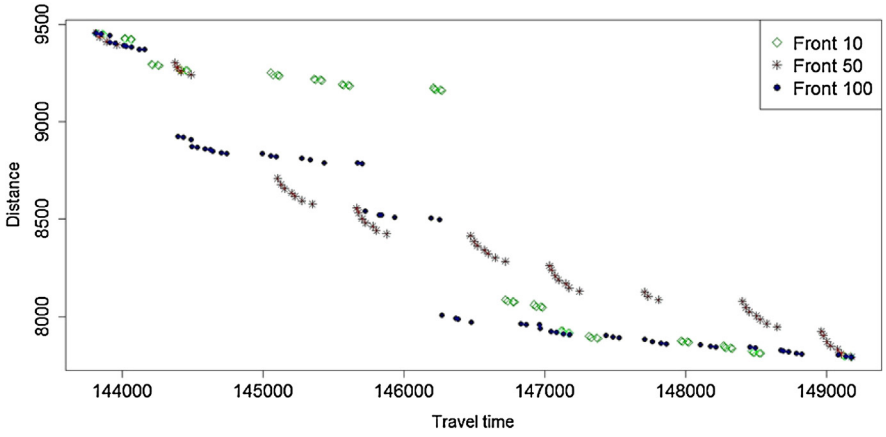

Fig. 13. Pareto front approximations obtained when solving the DTSP of New York with Dynamic NSGAII.

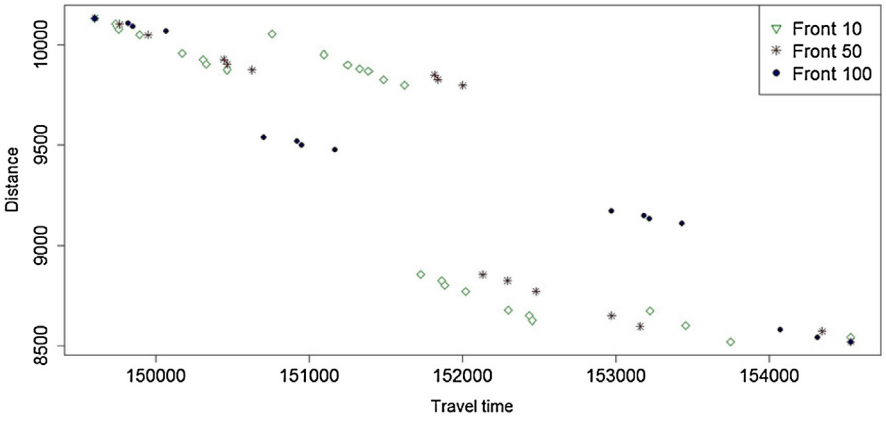

Fig. 14. Pareto front approximations obtained when solving the DTSP of New York with Dynamic MOCell.

it is worth mentioning that jMetalSP can also be used to parallelize multi-objective metaheuristics that solve conventional (static) multi-objective problems [25]. Once more, the key is to take advantage of algorithm templates.

If we take a look again to the pseudo-code of an evolutionary algorithm, the Step 6 consists on evaluating the new generated solutions. This step can be performed in parallel, because each solution evaluation is an independent task. jMetal 5 provides by default two strategies: sequential and multi-threaded. The idea in jMetalSP then is to use Spark.

The approach followed is to create an RDD with all the solutions to be evaluated and a map transformation is used to evaluate each solution in parallel. The evaluated solutions are then gathered together and returned to the algorithm.

The advantage of using this scheme is that the metaheuristics do not need to be modified, but the problem to be solved must fulfill the requirements imposed by Spark for run map processes (the closure of the tasks). 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.

The resulting parallel model alternates parallel steps (evaluating the solutions) with sequential ones (the rest of the algorithm), so we cannot expect to achieve linear speed-ups. Along the same lines, a study was carried out in [25], where different experiments are conducted and analyzed. These experiments consider the parallel performance of NSGA-II in a Hadoop cluster of 100 cores with and without accessing data files stored in HDFS. This study reveals that significant time reductions are achieved (see [25]).

8. Conclusions

In this paper, we have presented jMetalSP, a software solution to deal with dynamic Big Data Optimization problems combining the jMetal optimization framework with the Spark cluster computing system.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004

Our motivation has been driven by the rise of Spark as a distributed computing platform in clusters and Hadoop systems and the utilization of the jMetal framework as a multi-objective optimization engine. We have generated a case study that considers a bi-objective formulation of the TSP problem, where the goal is to minimize the total distance and travel time. Then, a real instance created from Open Data of the city from New York has been used to update the problem data in streaming. Two additional artificial data sources, based on Twitter and Kafka, have been added. After the experimental validation, three main conclusions can be drawn:

- jMetalSP can be easily adapted to deal with other Big Data Optimization problems, as well as to deploy and experiment with them in a current cloud computing Spark cluster.
- In the scope of the computational platform used, the proposed approach is able to efficiently manage a *normal moderate* (see Test 3) flow of data involving 1.37 TBs of information in 1 h. The system limits are reached when testing with *uniform intensive* (Test 2) amounts of data.
- The dynamic NSGA-II algorithm developed with jMetalSP is able to adapt to changing traffic conditions when solving the dynamic TSP problem. This is a desirable behavior when dealing with reactive approaches to benefit from information from multiple data sources.

As for future work, we plan to define more realistic problems, including additional data sources, as well as considering other optimization algorithms for Big Data Optimization.

Acknowledgements

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). José García-Nieto is the recipient of a Post-Doctoral fellowship of "Captación de Talento para la Investigación" Plan Propio at Universidad de Málaga.

References

[1] M. Marr, Big Data: Using SMART Big Data, Analytics and Metrics to Make Better Decisions and Improve Performance, Wiley, 2015.
[2] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S.U. Khan, The rise of big data on cloud computing: review and open research issues, Inf. Syst. 47 (2015) 98–115.
[3] Z.H. Zhou, N.V. Chawla, Y. Jin, G.J. Williams, Big data opportunities and challenges: discussions from data analytics perspectives [discussion forum], IEEE Comput. Intell. Mag. 9 (4) (2014) 62–74.
[4] M. Farina, K. Deb, P. Amato, Dynamic multiobjective optimization problems: test cases, approximations, and applications, IEEE Trans. Evol. Comput. 8 (5) (2004) 425–442.
[5] J. Durillo, A. Nebro, jMetal: a java framework for multi-objective optimization, Adv. Eng. Softw. 42 (10) (2011) 760–771.
[6] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, Berkeley, CA, USA (2010) 10.
[7] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[8] C.H. Papadimitriou, The Euclidean travelling salesman problem is np-complete, Theor. Comput. Sci. 4 (3) (1977) 237–244.
[9] J. Cordero, A. Nebro, J. Durillo, J. García-Nieto, C. Barba-González, I. Navas, J. Aldana-Montes, Dynamic multi-objective optimization with jMetal and Spark: a case study, Machine Learning, Optimization, and Big Data: Second International Workshop, MOD 2016, Volterra, Italy, August 26–29, 2016, Revised Selected Papers, Vol. 10122 of Lecture Notes in Computer Science (2016).
[10] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Design Issues in a Multiobjective Cellular Genetic Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 126–140, http://dx.doi.org/10.1007/978-3-540-70928-2_13.
[11] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Comput. Surv. 35 (3) (2003) 268–308.
[12] C. Coello, G. Lamont, D. van Veldhuizen, Multi-objective Optimization Using Evolutionary Algorithms, 2nd ed., John Wiley & Sons, Inc., NY, USA, 2007.
[13] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA – a platform and programming language independent interface for search algorithms, in: C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, L. Thiele (Eds.), Evolutionary Multi-criterion Optimization (EMO 2003), Lecture Notes in Computer Science, Springer, Berlin, 2003, pp. 494–508.
[14] A. Liefooghe, M. Basseur, L. Jourdan, E.-G. Talbi, ParadisEO-MOEO: a framework for evolutionary multi-objective optimization, in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Fourth International Conference on Evolutionary Multi-criterion Optimization, EMO 2007, Vol. 4403 of LNCS, Springer, Berlin, Germany, 2007, pp. 386–400.
[15] S. Abdul-Rahman, A.A. Bakar, Z.A. Mohamed-Hussein, Optimizing big data in bioinformatics with swarm algorithms, 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE) (2013) 1091–1095.
[16] K. Govindarajan, T.S. Somasundaram, V.S. Kumar, Kinshuk, Continuous clustering in big data learning analytics, 2013 IEEE Fifth International Conference on Technology for Education (T4E) (2013) 61–64.
[17] B.K. Tannahill, M. Jamshidi, System of systems and big data analytics, bridging the gap, Comput. Electr. Eng. 40 (1) (2014) 2–15 (40th-year commemorative issue).
[18] A. Cabanas-Abascal, E. García-Machicado, L. Prieto-González, A. de Amescua Seco, An item based geo-recommender system inspired by artificial immune algorithms, J. Univers. Comput. Sci. 19 (13) (2013) 2013–2033.
[19] W.-P. Lee, Y.-T. Hsiao, W.-C. Hwang, Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment, BMC Syst. Biol. 8 (1) (2014) 1–19.
[20] W. Sun, N. Zhang, H. Wang, W. Yin, T. Qiu, Paco: a period ACO based scheduling algorithm in cloud computing, 2013 International Conference on Cloud Computing and Big Data (CloudCom-Asia) (2013) 482–486.
[21] R. Lammel, Google's mapreduce programming model, Sci. Comput. Program. 70 (1) (2008) 1–30.
[22] A. Nebro, J.J. Durillo, M. Vergne, Redesigning the jMetal multi-objective optimization framework, in: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO Companion'15, ACM, New York, NY, USA, 2015, pp. 1093–1100.
[23] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: improving the strength Pareto evolutionary algorithm, in: K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, T. Fogarty (Eds.), EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece, 2002, pp. 95–100.
[24] N. Beume, B. Naujoks, M. Emmerich, SMS-EMOA: multiobjective selection based on dominated hypervolume, Eur. J. Oper. Res. 181 (3) (2007) 1653–1669.
[25] C. Barba-González, J. García-Nieto, A. Nebro, J. Aldana-Montes, Multi-objective big data optimization with jMetal and Spark, Evolutionary Multi-Criterion Optimization: 9th International Conference, EMO 2017, Münster, Germany, March 19–22, 2017, Proceedings, Vol. 10173 of Lecture Notes in Computer Science (2017) 16–30.

Please cite this article in press as: C. Barba-González, et al., jMetalSP: A framework for dynamic multi-objective big data optimization, Appl. Soft Comput. J. (2017), http://dx.doi.org/10.1016/j.asoc.2017.05.004