Future Generation Computer Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach

Visnja Simic [a,*], Boban Stojanovic [a,b], Milos Ivanovic [a,b]

[a] University of Kragujevac, Faculty of Science, Radoja Domanovica 12, 34000 Kragujevac, Serbia
[b] BioIRC Research and Development Center for Bioengineering, Kragujevac, Serbia

HIGHLIGHTS

- A software framework for large-scale optimization using parallel GA over cloud resources.
- Auto-scaling in the application layer for user QoS including cost and performance.
- ML surrogate model for predicting a delivery time and an execution cost.
- An intelligent decision support engine optimizes cost/performance ratio.
- Recommends optimal VM instance's parameters for the best efficiency.

ARTICLE INFO

Article history:
Received 13 September 2018
Received in revised form 8 April 2019
Accepted 17 July 2019
Available online xxxx

Keywords:
Parallel metaheuristics based optimization framework
Cloud computing
Machine learning
Surrogate model
Resource usage prediction
Resource usage optimization

ABSTRACT

The cloud computing paradigm has gained wide acceptance in the scientific community, taking a significant share from fields previously reserved exclusively for High Performance Computing (HPC). On-demand access to a large amount of computing resources provided by Cloud makes it ideal for executing large-scale optimizations using evolutionary algorithms without the need for owning any computing infrastructure. In this regard, we extended WoBinGO, an existing parallel software framework for genetic algorithm based optimization, to be used in Cloud. With these extensions, the framework is capable of elastically and frugally utilizing the underlying cloud computing infrastructure for performing computationally expensive fitness evaluations.

We studied two issues that are pertinent when dealing with large-scale optimization in the elastic cloud environment: the computing instance launching overhead and the price of engaging Cloud for solving optimization problems, in terms of the instances' cumulative uptime. To explain the usability limits of WoBinGO framework running in the IaaS environment, a comprehensive analysis of the framework's performance was given.

Optimization of both total optimization time and total cumulative uptime, leads to minimizing the cost of cloud resources utilization. In this way, we are proposing an intelligent decision support engine based on artificial neural networks and metaheuristics to provide the user with an assessment of the framework's behavior on the underlying infrastructure in terms of optimization duration and the cost of resource consumption. According to a given assessment, the user can decide upon faster delivery of results or lower infrastructure costs.

The proposed software framework has been used to solve a complex real-world optimization problem of a subsurface rock mass model calibration. The results obtained from the private OpenStack deployment show that by using the proposed decision support engine, significant savings can be achieved in both optimization time and optimization cost.

1. Introduction

Cloud computing has shown a great deal of promise as a scalable and cost-effective computing model for supporting scientific applications [1–3]. Until recently, scientists and engineers have used supercomputers, HPC clusters or computing grids to execute applications that require a large amount of resources to solve complex problems. Now, Cloud [4] offers a pay-as-you-go computing environment for the execution of largescale applications at low costs and without the need for owning any computing infrastructure.

* Corresponding author.
E-mail addresses: visnja@kg.ac.rs (V. Simic), bobi@kg.ac.rs (B. Stojanovic), mivanovic@kg.ac.rs (M. Ivanovic).

https://doi.org/10.1016/j.future.2019.07.042
0167-739X/

One of the domains that can certainly benefit from employing cloud resources is optimization based on evolutionary algorithms. Evolutionary algorithms (EAs) are a population-based metaheuristic inspired by the survival of the fittest principle, whose use has become increasingly popular over the last three decades, mainly for real-world large-scale optimization and classification tasks. Some of the most popular EAs are genetic algorithms (GAs) [5]. A GA involves thousands of objective function evaluations, which in the case of a real-world problem can be quite time-consuming, and may take days, weeks or even months for the GA to find an acceptable solution. Speeding up the optimization process is achieved by parallelization of the GA [6], which reduces the resolution times to reasonable levels. An implementation of parallel metaheuristics has previously been mainly realized using HPC clusters, computing grids and volunteer peer-to-peer systems [7–10]. Unfortunately, most individuals and institutions that want to employ evolutionary optimization to solve real-world problems do not have the access to these types of resources and/or do not possess the expertise necessary for their use, which makes it impossible for them to obtain feasible results within a reasonable time frame. Additionally, many of those in need of optimization do not need to perform it on a daily, or even monthly basis, and therefore investing in an expensive HPC cluster and its further maintenance would be completely unacceptable to them. Recent availability of Cloud on-demand massive computing resources offers new opportunities for the development of the optimization frameworks based on parallel EAs. Cloud services enable provisioning of resources beyond what is available in most research labs, allowing implementation of parallel metaheuristics at reasonable prices. An infrastructure is fully operated by the cloud provider, which completely eliminates the cost of equipment purchase. An additional benefit from Cloud exploitation is that hardware maintenance and support are transferred to the provider. The utilization of the cloud computing power makes a large-scale optimization available to a significantly wider range of users. Offering a cloud-based optimization service can make them capable to introduce parallel metaheuristics procedures, regardless of their financial, technological or knowledge level.

The aim of this paper is to present the latest development advances within the WoBinGO software framework [10]. In recent years, WoBinGO has been used for solving large-scale real-world optimization problems over heterogeneous computing resources, including HPC clusters and Grids. Examples of its successful utilization range from portfolio optimization [11], to near real-time adjustment of artificial neural network architecture in a dam health monitoring system [12]. However, since only large companies and research institutions can afford private computing infrastructure for employing WoBinGO, we have decided to upgrade WoBinGO in order to make use of private and public cloud infrastructures compatible with EC2 API, thus allowing large-scale optimization at reasonable prices. We built a set of supporting services for WoBinGO with an inherent capability of automatically launching and maintaining the pool of full-stack cloud instances. The advances to the framework were introduced to meet the following requirements:

- Speeding up the optimization process by parallelization of GA and employing multiple computing instances in the Cloud for evaluation of the individuals.
- Relieving the researcher burden of obtaining cloud resources and dealing with various provider APIs.
- Maintaining the pool of ready computing instances to enable their fast allocation and avoid waiting until requests for computing resources are processed by the provider.
- Elastic provisioning of resources in accordance with the dynamics of the users' requests, thus eliminating unnecessary costs and reducing energy consumption.
- Providing an intelligent, machine-learning based decision support engine for determining the optimal cloud instance parameters to achieve the best performance to IaaS consumption ratio.

There are two major issues to be considered when dealing with large-scale optimization in the elastic cloud environment. The first issue comes from the fact that a computing instance launching overhead is considerably higher compared to a launching overhead of a simple batch job in the HPC cluster queue. In further text, we will assess the usability limits of the WoBinGO framework running within the IaaS environment. The second issue to be considered is the price of optimization task execution in terms of computing instances' cumulative uptime. Discussing the cost of solving the hard optimization problem was not at all in the focus of [10], while the aim of this paper is to show that it is possible to provide a reliable mechanism to minimize both the total optimization time and optimization cost.

The distinctive feature of WoBinGO, as shown in [10], is the limited lifetime of worker jobs. When that time is about to expire, despite there being more evaluation work to be done, the system removes the current job and submits an appropriate number of new jobs according to the recent load. In terms of WoBinGO's cloud implementation, the workers performing evaluations are actually cloud instances. Therefore, the lifetime parameter affects a system inertia by controlling the uptime of each computing instance and, as we will show later in the manuscript, directly influences the cost of the acquired computing power. By adjusting the lifetime value, according to the requirements of a specific optimization task, it is possible to achieve savings in overall duration and cost of the optimization process.

In addition to lifetime, the factor that mostly contributes to performance is the maximum number of engaged computing instances. It can also be adjusted to influence the total optimization time and cost. The ideal combination of lifetime and a maximum number of workers for a certain optimization task depends on the performance of the underlying IaaS infrastructure, the mean evaluation time of the individual and the number of individuals within a population. Special attention was devoted to determining the optimal combination of lifetime and maximum number of workers for the optimization problem at hand, such that the total optimization time and cost are both minimal. Enhancing the user's QoS, we developed an intelligent decision support engine which uses machine learning and historical data to model the behavior of the underlying infrastructure under various load patterns. Subsequently, a user is provided with the optimal combination of an instance lifetime and the maximum number of instances to achieve the best performance to resource (and consequently energy) consumption ratio. As shown by the results of the conducted experiments, by using the proposed decision support engine, it is possible to significantly decrease IaaS cost, while keeping the same total optimization time.

The rest of the paper is organized as follows: in Section 2, we review the related work. A description of the framework is given in Section 3. In Section 4, the experimental results and discussion are given, along with a case study, followed by concluding remarks in the last section.

2. Related work

Over the past few years, porting the execution of parallel metaheuristics to Cloud has gained the growing attention of the scientific community. Probably the first example of integration of EAs with Cloud is found in [13]. The *Offspring* framework provides

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

support for researchers of combinatorial optimization in quickly deploying their algorithms on a distributed computing infrastructure. The framework relies on *Aneka* [14,15] to distribute multi-objective evolutionary algorithms on Enterprise Clouds. The distribution model of *Offspring* has been implemented on top of the task model with a plugin architecture. The other example of Cloud usage in evolutionary computation can be found in [16], which advances the contribution of [17], and presents a distributed evolutionary computation system that relies on two different cloud storage services (Dropbox and SugarSync) for the file synchronization among islands, while each island is hosted by a different computer.

Building upon the concept presented in [18] and [19], the authors in [20] introduce the *EvoSpace* population storage model for the development of pool-based EAs that can be executed over cloud computing resources. The evolving population is stored in a centralized repository, while distributed clients asynchronously extract a subset of individuals and return a new subset of individuals after performing the search operators. *EvoSpace* is configured to run on a cloud architecture using Heroku for the *EvoSpace* server and PiCloud for simulating *EvoWorkers*. The study shows how EAs can scale on Cloud and how Cloud can make EAs effective in a real-world environment, speeding up the running time and lowering the costs. This concept is to some extent similar to our work, in terms that workers are autonomous computational entities which only communicate with the *EvoStore* and not with each other. However, an important drawback is related to a researcher's tools: in Heroku only certain versions of languages are currently supported and only a certain number of services are available as Add-Ons; in PiCloud the integration with Python is transparent, but other languages or binaries need to be deployed to customizable environments.

In [21] the authors analyse research projects and important requirements in the context of optimization services, and offer the definition of a reference architecture for cloud-based optimization services. The authors also provide the prototype of a cloud-based optimization service (OaaS), which originates from their previous work [22] where it was initially presented. OaaS *HeuristicLab HIVE* defines a generic and extensible service which can be adapted to support custom optimization scenarios. The service is based on the Microsoft .NET framework, and therefore intended only for the Windows Azure cloud provider, subsequently using a Microsoft solution for automatic scaling of computational resources, which makes it highly platform dependent. Additionally, it requires users to establish a computing infrastructure on their own.

The authors of [23] introduce a *SPACE* web-based framework for distributing expensive fitness evaluations across an elastic, heterogeneous pool of computing nodes that include both the personal computers of users/volunteers running JavaScript software in their web browsers as well as a variable sized pool of cloud computing nodes running an optimized C++ version of the software.

Frameworks presented in [23] and [21] have the advantage over other solutions in that they enable concurrent execution of multiple different experiments, initiated by different users, on the same network. Other solutions (as they are implemented now) do not offer this opportunity, because they host the population on a central server and therefore would require a separate server for each concurrent experiment.

With the aim of parallelizing GAs on a commercial cloud environment and in a DevOps fashion, paper [24] presents a novel approach to distribute GAs, implementing the global parallelization model and exploiting technologies specifically devised for Cloud (i.e., Docker, CoreOS and RabbitMQ). However, there is no empirical assessment of the effectiveness of such system in terms of execution time and scalability, when solving a real problem.

Numerous authors have suggested using the MapReduce paradigm for parallelization of EAs over a cloud infrastructure, in order to relieve programmers of most of the distributed computation issues. An additional motivation for the use of this model stems from the fact that it is natively supported by several cloud infrastructures. In [25] authors utilize Cloud to execute parallel genetic algorithm (PGA) for the purpose of automated software testing. Three different models of PGA were adapted to the MapReduce paradigm and issues and concerns about them were discussed. Paper [26] provides a comprehensive analysis and a comparison of the three PGA models in terms of execution time, speedup, overhead and computational effort. To realise the PGAs, the authors exploited the *elephant56* framework [27], which is an open source project offering the possibility of distributing the GA's computation over a Hadoop MapReduce. The cost estimation of the execution on a potential cloud infrastructure is also considered, and the results suggest that the island model is worth using compared to the execution with a single machine. Although the conducted benchmark is comprehensive, the results would probably be significantly better if a considerably faster Spark MapReduce was employed instead of the IO heavy Hadoop MapReduce.

There are a number of survey papers [28–31] that classify elasticity in a cloud environment. Aforementioned *HeuristicLab HIVE* [21], *EvoSpace* [20] and *Aneka* [15], use elasticity for increasing the local resources' capacity. According to classification for elasticity mechanisms found in [32], WoBinGO employs an automatic reactive elasticity policy through replication, which consists of adding and removing virtual machine instances from the infrastructure. The elasticity controller is embedded within the framework itself in order to automatically manage the resources used by the application, as in [15]. The elasticity is used to avoid the inadequate provision of resources and consequently reduce the cost of cloud resources' exploitation. In [33], the authors report more general approach in providing fully automatic elasticity for standard HPC workloads migrating to the Cloud, considering both performance, cost and energy consumption. If referring to the Zhan's [30] taxonomy of a cloud resource scheduling, one can say that WoBinGO performs scheduling resources in the application layer for user QoS including cost and performance.

The general conclusion of all the above mentioned works is that unless for toy examples, Cloud execution of parallel EAs greatly exceeds the performance of a local server. However, none of these solutions offer elasticity in the allocation of cloud resources, or enable users to adjust a framework's parameters in accordance with their preferences in regards to the total optimization time and cost of solving the particular optimization problem.

3. WoBinGO in the cloud

WoBinGO framework has recently been successfully used for solving hard real-world optimization problems. It has been improved to enable fully automated execution of EA based optimization on IaaS, in addition to the standard HPC base. The principles upon which the WoBinGO was created have stayed the same, and all the benefits that the framework previously provided by engaging Work Binder (WB in further text) service [34] are still there. These benefits include fully automatic allocation of the computing resources, quick client-worker binding by elastic maintenance of the pool of ready workers, etc. [10]. From the users' point of view, porting the framework to the Cloud induced the following features:

- SaaS (Software as a Service) based optimization software allows the optimization methods to be exposed as an Internet service.

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

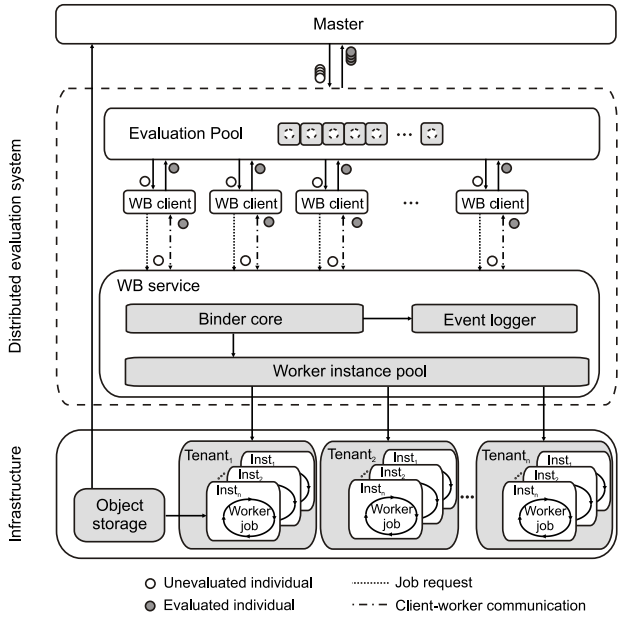

Fig. 1. WoBinGO cloud-based architecture.

- Speeding up the optimization process by employing multiple cloud instances for the evaluation of the individuals in PGA.
- Pool of ready cloud instances is elastically maintained in accordance with the dynamics of the users' requests. In this way unnecessary costs are avoided and energy consumption is reduced.
- Worker instances with limited lifetime ensure frugal utilization of the computing resources, resulting in cost savings and minimal energy consumption.
- Objective function can be written in any compiled or script language supported by an underlying OS/runtime environment.

3.1. Architecture

The architecture of the optimization framework remained intact from [10], with the addition of the possibility to automatically launch full stack virtual instances besides standard batching and grid jobs. The framework employs standard EC2 API to manage compute instances, which is compatible with a large number of IaaS providers. Another change was the location of the objective function evaluator, which now resides within the object storage service.

The basic structure of the framework is illustrated in Fig. 1. The framework consists of the optimization master and the distributed evaluation system based on WB. The distributed evaluation system is composed of the evaluation pool and the WB subsystem.

The **master** performs the main evolutionary loop to the point when a generation has to be evaluated. At that moment, the master sends all individuals in a generation to the evaluation pool, waiting for the results. After all individuals from a generation have been evaluated, the master proceeds with the rest of the evolutionary algorithm until the stopping criteria is reached. The **evaluation pool** acts as an intermediate layer between the master and the WB service. It provides asynchronous parallel evaluation of individuals. The **WB** part consists of software components distributed in three tiers: the client, the worker VM instances, and the WB service. The purpose of the WB service is to maintain the pool of ready instances and to bind them with clients that request evaluation. It launches an appropriate number of worker instances in order to satisfy incoming requests. The client establishes a connection to the WB service and requests a worker. As soon as the client and instance are successfully coupled, the WB service acts as a simple proxy. After completing the evaluation, provided that its lifetime did not expire, the VM instance reconnects to the WB service asking for more work. Each GA-based optimization process requires an **evaluator** of objective function(s), which has to be supplied in the form of a bundle. The communication interface is simple: the evaluator executable has to take an *input.xml* that contains types and values of the parameters and has to produce an *output.xml* that carries objective function values. Upon the start, each computing instance downloads evaluator bundle(s) from the object storage using *cloud-init* [35] and informs the WB service about the willingness to perform evaluations. The bundle can be supplied in the form of an archive, or a Docker image in the case of more complex dependencies.

3.2. Parameters that affect the optimization performance

The shortest optimization execution time is achieved by the fully static approach, meaning a separate VM instance is engaged for the evaluation of each individual in the generation and all worker instances are active throughout the whole optimization process. This scenario is generally not always possible in a public cloud, due to the fact that currently public cloud providers imply strict limits regarding the amount of resources that can be given to a user at once. For example, Amazon allows normal users to request simultaneously only 20 on-demand instances and 100 spot instances per region. Additionally, even if it was always possible to perform *one-to-one mapping* between individuals in the population and the computing instances, the cost-effectiveness of such an approach would be questionable. In fact, when solving real-world optimization problems, the evaluation time often varies greatly from individual to individual. For example, in optimization based on simulation with an adaptive time step, the evaluation of some solutions can last much longer than the evaluation of the rest of the population. Therefore, most of the engaged instances would complete assigned evaluations and then idly wait for the remaining few to complete their work, which causes unnecessary cost. Another scenario in which the idle worker instances are encountered occurs when the size of the population is not divisible by the current number of computing instances. In this case, a certain number of instances will be idle, until the evaluation of the remaining individuals completes. The existence of idle instances indicates the possibility of reducing the number of deployed instances without deteriorating the optimization performance.

With WoBinGO, the problem of idle worker instances is assessed by using the inherent elasticity of Work Binder (WB) service [34]. As explained in [10], WB is capable of utilizing underlying infrastructure elastically in accordance with resource availability and the present workload. An instance has a defined maximum lifetime L which is determined by a tuple (r_{max}, t_{idle}). The parameter r_{max} represents the maximum number of evaluations that a worker instance can perform during its lifetime, while t_{idle} is the time that an instance is allowed to spend in the idle state. If a worker has performed as many evaluations as specified by r_{max} or has not been given an individual to evaluate for t_{idle}, its lifetime is considered expired. After an instance completes its lifetime, it is turned off, regardless of whether there are more individuals to evaluate or not. If necessary, WB will activate new instances on the basis of an analysis of incoming client requests to replace those that have shut-off.

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

In WoBinGO's cluster/grid deployment, the worker job's limited lifetime provided friendliness towards other batching queue users and enabled them to reach a running state significantly faster compared to a static pilot-job approach. However, in terms of IaaS, the workers performing evaluations are cloud instances, while the lifetime parameter affects the elasticity inertia by controlling the uptime of each worker instance. The lower value of lifetime reduces the inertness of the system when the number of workers needs to be scaled down. At the same time, the system becomes more inert when it needs to increase the number of workers, since lower value of lifetime means more frequent instance on/off switching.

On the other hand, in the case of a long lifetime, the WB has at its disposal a larger number of ready worker instances and can quickly handle sudden rush of tasks, such as when starting the evaluation of the generation. However, the idle periods cost money and consume energy. In the case when we do not want to keep the idle instances, we must accept the overhead when raising new instances. Thus, by controlling the value of the lifetime, one can reduce the overall optimization time, but also the cumulative uptime of engaged worker instances and, consequently, the optimization task cost.

The total optimization time depends directly on the optimization problem itself defined by the population size, number of generations, and the average evaluation time of an individual on a certain type of cloud instance. Theoretically, ideal total optimization time and ideal cumulative uptime of engaged worker instances for a particular task (denoted as T^*_{opt} and T^*_{cum}, respectively) can be calculated as:

$$T^*_{opt} = \left(t_{eval} \cdot \left\lceil \frac{N}{W_{max}} \right\rceil + t_{seq} \right) \cdot G, \quad (1)$$

$$T^*_{cum} = T^*_{opt} \cdot W_{max} \quad (2)$$

where t_{eval} is an average evaluation time of an individual, N is the number of individuals in the population, W_{max} is the maximum number of VM instances used for performing evaluations, and G is the number of generations to be evaluated. In an ideal scenario, when a population can be distributed among workers evenly, each worker evaluates the same number of individuals (N/W_{max}), spending t_{eval} for each of them throughout G generations. Additionally, the time required for the execution of the sequential part of the algorithm (t_{seq}) must be taken into account for each generation. When a population size is not divisible by the maximum number of workers, the remainder of the individuals will be evaluated for additional t_{eval} in each generation.

3.3. Decision support engine

In order to improve QoS, we decided to expand WoBinGO with a predictive decision support engine. It provides end users with an assessment of the framework's behavior on the specific underlying infrastructure in terms of Pareto optimal combinations of the total time required to complete optimization, and the cost of resource consumption during that period. According to a given assessment, a user can further select one of three possible options: (i) to sacrifice time to save money; (ii) to pay more to get the results as soon as possible or (iii) to achieve a balance between the cost and the optimization time. As stated above, the total optimization time and total uptime on the IaaS platform can be influenced by adjusting the lifetime and the maximum number of engaged instances. Therefore, minimizing both execution time and cost of the optimization process is a multiobjective optimization problem with two decision variables: L and W_{max}. The total optimization time and cumulative instances' uptime are also largely influenced by the overall performance

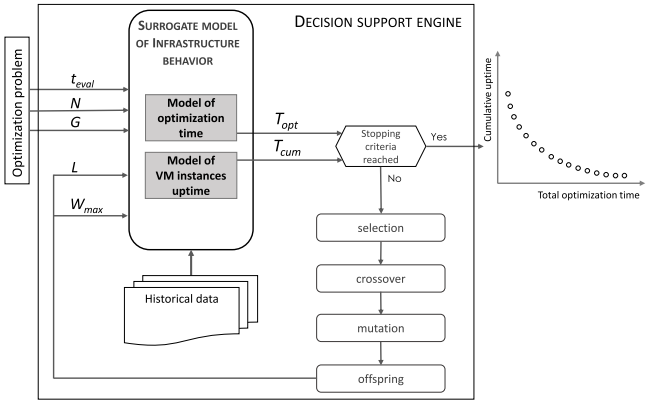

Fig. 2. The decision support engine workflow.

of an underlying infrastructure. There are various indicators of the performance of a certain IaaS provider, such as an instance boot and shutdown time, oversubscription rate, instance creation time, network latency, network throughput, object and block storage performance, etc. If we could determine how all these infrastructure factors cumulatively affect the total optimization time and the total uptime for a specific optimization problem, then we could reduce both of them. However, since relations among the quantities involved in the decision process are too complex to be modeled using a physically based model, the most effective way to estimate a solution is to build a surrogate model of IaaS performance under an optimization load by employing machine learning and historical data from previously performed optimizations.

Two surrogate models were built to estimate an infrastructure response to various combinations of L and W_{max} for the specific optimization task. Both surrogate models have the following inputs (Fig. 2): t_{idle}, r_{max}, t_{eval}, N, G, W_{max}, where t_{idle} and r_{max} are constitutive parts of the L tuple, as defined above. Output from the first model is the total optimization time and the second model outputs the cumulative uptime. The models were built using Artificial Neural Networks (ANNs). Given the previous historical data, two fully-connected neural networks were trained using the back-propagation algorithm. The first ANN is used to predict the total optimization time T_{opt} and the other ANN to predict the total cumulative uptime of the worker instances T_{cum}.

As depicted in Fig. 2, the built surrogate models of the infrastructure behavior act as objective functions evaluators in the process of determining optimal (L, W_{max}) combinations. The optimization is performed within the decision support engine using multi-objective GA (MOGA). After the initial population of solutions was randomly generated, it undergoes the evaluation using the infrastructure's surrogate models. The population is then subjected to the iterative process of selection, crossover, mutation and evaluation. The iterative process is terminated when solutions of satisfactory quality are obtained or when the maximum number of iterations is reached. After the execution of MOGA, the decision support engine produces an output in the form of a Pareto front showing possible trade-offs between T_{opt} and T_{cum} for different combinations of L and W_{max}.

3.4. The system implementation

The software system as implemented on the underlying OpenStack installation is depicted in Fig. 3. The decision support engine is a hybrid neuro-evolutionary system, but from the user's point of view it represents a black box. A user supplies the system with an optimization problem definition (Step 1) through the

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

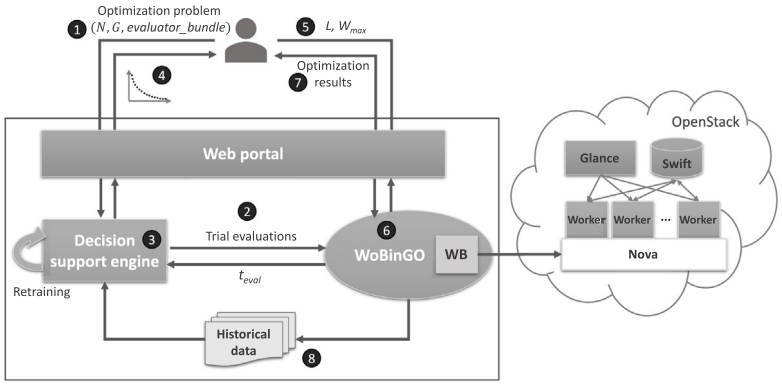

Fig. 3. The implementation of the proposed optimization system. The user supplies the optimization problem through the Web Portal, while the Decision support engine helps him choose the best worker instance parameters to perform the supplied optimization task. The optimization itself is carried out by WoBinGO framework which maintains the pool of worker instances on Openstack.

web portal. Amongst other parameters, the definition includes a population size (N), the maximum number of generations in GA (G) and an objective function evaluator. To determine the average t_{eval}, the system executes several trial evaluations of randomly generated individuals on a target type computing instance (Step 2). Upon obtaining t_{eval}, the decision support engine executes MOGA, and determines the optimal combinations of a computing instance's L and W_{max} (Step 3). They are given in the form of Pareto optimal solutions in regards to T_{opt} and T_{cum} (Step 4). A user can choose a single solution according to his preferences - faster optimization process, or lower cost, or some kind of balance between the two. The chosen values of L and W_{max} are further passed to WoBinGO (Step 5) to be used for the optimization problem described by a user (Step 6). WoBinGO performs the optimization process as described in [10], with the difference that instead of HPC batching jobs, worker instances are used for evaluations of individuals. The final result (Step 7) is a solution to a user's optimization problem, delivered in the minimum time and at lowest price possible, regarding the user's preferences. Upon each optimization, the system logs the data in the form of tuples $(N, G, t_{eval}, W_{max}, r_{max}, t_{idle}, T_{opt}, T_{cum})$ into the database (Step 8) and periodically retrains the surrogate models of infrastructure behavior in accordance with the enriched data set (Step 9).

It should be emphasized that, when an approximate model of infrastructure behavior is used for objective function evaluation within the decision support engine, a false optima can be encountered [36]. More precisely, the parametric approximation technique may not be capable of modeling the problem landscapes accurately, and can thus produce an unreliable search. There is no unique approach to prevent convergence to a false optimum. In this implementation we employed our own approach based on the assessment of ideal optimization time (T^*_{opt}) and ideal cumulative uptime (T^*_{cum}) of VM instances, using Eqs. (1) and (2). When an individual in a population contains T_{opt} and/or T_{cum} values that are better than ideal, it is automatically punished with suitably harsh fitness values for both objectives. This way, not only an individual with parameters leading to false optimum is punished, but the MOGA itself is "diverted" from the area which leads to the obvious false minima.

4. Empirical study

When employed over HPC clusters and grids, WoBinGo framework provided significant speed-up for optimizations that have computationally expensive evaluations, despite its adaptive and frugal allocation of the computing resources, as reported in [10]. Upon implementing the framework on the Cloud, our first aim was to determine to what extent WB's auto-scaling affected the pure performance in terms of the optimization time, considering a significantly higher infrastructure overhead compared to an HPC cluster. Secondly, we focused on demonstrating the influence of the instances' lifetime on the total optimization time and the cumulative uptime of the instance engagement.

Further in this section, we considered the building blocks needed to build the decision support engine. The main components of the decision support engine are the surrogate models of the infrastructure behavior under various optimization loads, whose performance is thoroughly discussed. The last experiment presented in this section was intended to test our software system through a case study in regards to a real-world optimization problem. The goal was to examine whether the use of the Pareto optimal solution suggested by the engine brings savings, either in terms of optimization time or optimization cost.

4.1. The performance of pure WoBinGO on IaaS

To accomplish the first objective, we carried out benchmarks with the aim of comparing (i) WoBinGO on the IaaS with a static number of worker instances and (ii) WoBinGO on the IaaS with its inherent elastic behavior. In the first benchmark, we consider the performance only in terms of total optimization time and achieved speedup.

Using a simple GA with real-encoded chromosomes we solved the artificial test problem. The fitness function was a dummy function that did nothing except receive an individual, sleep for n seconds, and return a random fitness value to the master. For the purpose of this study, it only mattered how long it took the instances to return their responses. The simple GA was executed for 10 generations and the results reported are the average of 10 independent runs.

To execute the experiment, we employed a private IaaS infrastructure based on *OpenStack Juno* hosted on Ubuntu 16.04 x86_64 deployed on 4 physical servers with Intel Xeon E5-2620 v2 CPUs, totalling to 96 cores, 128 GB memory and a gigabit interconnection. All VM instance images were also Ubuntu 16.04. In order to perform a fair experiment, we used the same configuration for each instance (see Table 1). It is worth noting that we did not simulate any hardware component, thus exploiting the virtualization of OpenStack only as a way to equally decompose the underlying hardware infrastructure. We completely dedicated the partial atomic resource (e.g., CPU cores, RAM) to the running instances so that they could have fully used them without overlapping with others.

The WoBinGO setup on the OpenStack closely mimics the way the framework is used within the standard HPC environment. The only crucial difference is the usage of worker instances instead

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

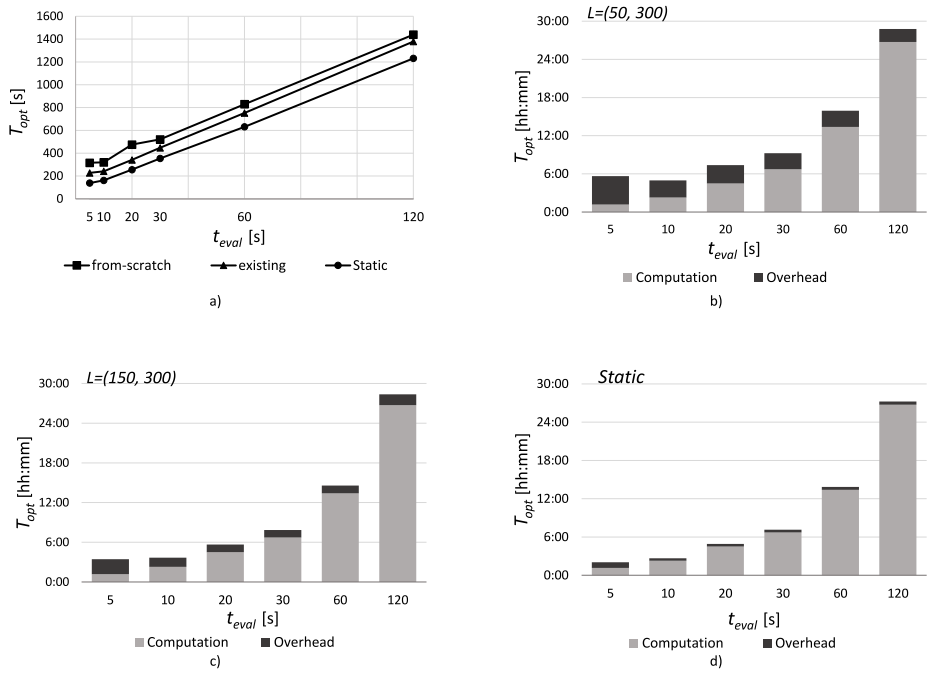

Fig. 4. Infrastructure overhead: (a) comparative display of the experimentally obtained wall-clock time for various launching policies of the instances; the ratio between the actual computational time and the overhead for different values of the lifetime (b) L = (50, 300), (c) L = (150, 300) and (d) the static run.

Table 1
Virtual machines' configuration.

Hardware		Software	
Feature	Value	Feature	Value
Architecture	64 bit	OS	Ubuntu 16.04
CPUs	1	Java	1.8
RAM	2 GB	Mono	5.10.0.160
Storage	10 GB		

of HPC batching jobs. The WB service (Fig. 1) is hosted on a separate instance which is capable to invoke OpenStack (mostly Nova) services using RESTful interface. It also exposes a NFS shared home, mimicking standard HPC setup. As soon as the WB service notices the current pool of ready workers is insufficient, it initiates a new worker instance. Then one of the following two cases may occur: (**existing**) there is a worker instance already available, and (**from-scratch**) there are no more worker instances available. In the first case, the WB service just switches an instance on, and the launched instance mounts NFS shared home, downloads the evaluator bundle from the object storage (Swift) and announces its readiness to perform evaluations. In the second case, prior to starting the worker instance up, we have to create it from the image (Glance service), call *cloud-init* to create a user, install necessary packages, and deploy WB components. Upon launch, the worker instance created from the image announces its readiness to perform evaluations to the WB service. The worker instance lives as long as prescribed by the *lifetime* (denoted as L above). When time expires, it automatically turns off.

The maximum number of worker instances (W_{max}) was set to 40. The estimated average period required for the sequential part of the algorithm was $t_{seq} = 2$ s. The expected execution time was:

$$T_{par} = t_{seq} + \frac{n \cdot t_{eval}}{\bar{p}} + O(n, \bar{p}), \tag{3}$$

where $\bar{p}$ is the average number of worker instances that performed evaluation of $n = N \cdot G$ individuals and t_{eval} is the average

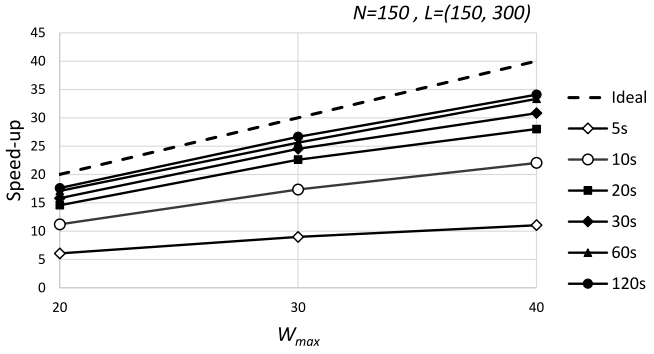

Fig. 5. Speed-up curve.

time needed to evaluate a single individual. $O(n, \bar{p})$ represents the total overhead including instance creation (only if it has to be created from the image), its launch, boot sequence, communication, etc. The time required for solving the same optimization problem in serial T_{ser} can be expressed as:

$$T_{ser} = t_{seq} + n \cdot t_{eval}. \tag{4}$$

In order to achieve any speed-up, the following must hold: $T_{ser} > T_{par}$. Combining Eqs. (3) and (4) leads to:

$$t_{eval} > \frac{\bar{p}}{n(\bar{p} - 1)} \cdot O(n, \bar{p}) \tag{5}$$

The Eq. (5) specifies the profitability limit saying that it is worthwhile to use WoBinGO only when condition Eq. (5) is satisfied. Due to the proportionally higher launch overhead in IaaS than with a simple batch job submission, we expected higher profitability limit of t_{eval} compared to a HPC cluster run [10]. The diagram in Fig. 4a gives a comparative display of the experimentally obtained wall-clock time needed to evolve 10 generations of 40 individuals with 40 worker instances performing evaluations for the following cases: (*from-scratch*) auto-scaling of limited

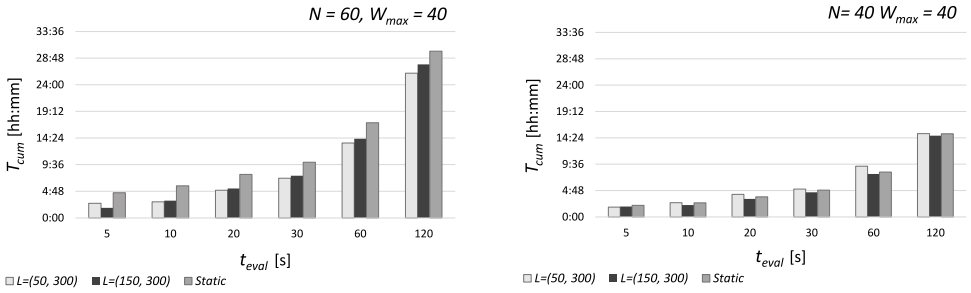

Fig. 6. Impact of the lifetime parameter on the total cumulative uptime.

lifetime worker instances created from the image; (*existing*) auto-scaling of already existing worker instances with limited lifetime; (*static*) all worker instances stay active throughout the whole optimization process. The static run was undoubtedly the fastest as it emulates the ideal setup, followed by a case that uses existing worker instances. The run which involved building instances from the image performed the worst, but the overhead was constant and predictable. In our test run, the overheads compared to the static run were 110 s for existing worker instances and 200 s for instances that had to be created from the image. Speaking in relative terms, in the case of the evaluation that lasted 120 s, the cost of the infrastructure overhead was approximately equal to a period necessary to evaluate one generation within the GA process. Having a sufficient amount of log data, these numbers could be easily estimated for each cloud provider, but this estimation is out of scope of this paper. In order to demonstrate the consistency of the overhead, we assessed the relationship between the execution time in static scenario compared to the scenarios with instances created from the image and with existing worker instances. The Pearson correlation coefficient ranges between 0.98 and 0.99, proving that the contribution of any type of IaaS overhead to the optimization time is mostly constant.

Further, we examined the ratio between the actual computational time and the overhead for different values of the lifetime and compared it to the static run used as a reference. Fig. 4b and c show results for two versions of the elastic approach which differ only in lifetime values. The static run (Fig. 4d) exhibits a much smaller overhead when compared to the elastic ones (Fig. 4b and c). It is also apparent that the overhead decreases with the increase of lifetime for the reasons discussed in Section 3.2.

The overheads are quite significant for short evaluation times, as was the case with the WoBinGO's grid deployment (reported in [10]). This is due to the fact that in the case of problems with a relatively short fitness evaluation, the frequency of client and worker requests addressed to the WB service is higher than with longer fitness evaluations. A higher frequency implies longer waiting for the requests' fulfilment. For evaluation times over 10 s, as can be noticed in Fig. 4(b), (c) and (d), the overhead was constant regardless of the evaluation time. Thus, in the case of cloud deployment, the profitability limit for evaluation time turns out to be 10 s in contrast to the 5 s limit for the HPC cluster deployment that was reported in [10].

Once we obtained the execution times and overhead involved with the WB's auto-scaling approach, we calculated the speedup by dividing the total amount of time required for sequential execution of the optimization algorithm by the amount of time required by the PGA. We compared the achieved speed-up with the ideal speed-up (number of employed parallel workers) (Fig. 5) In the case of more time consuming evaluations, the speed-up approaches the ideal value, and for short evaluations, the speed-up is poor, because of the overhead discussed above.

The presented empirical results prove that the WB's auto-scaling does not affect the system's performance significantly, especially in the cases of problems with computationally expensive evaluations, which are common in real-world applications, such as simulation-based optimizations.

4.2. The impact of instance lifetime

The performance was not the only factor that had to be taken into account when dealing with a large-scale optimization in a cloud environment. The cumulative uptime of the instances (T_{cum}) which contributed to the evaluation process was also a significant factor, since it directly influences the cost of the optimization, together with the IaaS provider payment policy. Therefore, the second set of experiments that were carried out were aimed at demonstrating how adjusting the worker instance lifetime affects T_{cum} and, consequently, the cost. The results are presented in Fig. 6. As can be seen from both diagrams, the lifetime has a significant impact on the total cumulative time of computing instances' engagement. However, it is obvious that the relation between lifetime, number of individuals, number of workers and evaluation time on one side and total uptime on the other side is complex, and this is exactly the reason why we had to employ a surrogate prediction models.

Having the uptime data, it was possible to carry out a cost–benefit analysis that compares the computational benefit to the cost of running evaluations on the cloud infrastructure (both in terms of time). We calculated WoBinGO's cost efficiency using the following equation:

$$\epsilon = \frac{T_c}{T_{cum}}, \tag{6}$$

where $T_c = t_{eval} \cdot N \cdot G$ denotes pure computational time. The ideal value of ϵ is 1, which means that instances were utilized all the time during the optimization process. It is apparent from the resulting diagrams (Fig. 7) that with the increase of an individual evaluation time (t_{eval}) our system becomes more efficient. However, the efficiency is obviously non-linearly dependent on worker instance lifetime. For various combinations of the evaluation time t_{eval}, the number of individuals N, and the maximum number of workers (W_{max}), the lifetime L influences the cost efficiency in a different way.

4.3. Training of the surrogate models with the historical data

The surrogate models of the infrastructure behaviour under the optimization load were constructed using two artificial neural networks. The first ANN estimated the overall duration of the optimization process, and the second ANN was used for the assessment of the total cumulative time of computing instances' engagement (Fig. 2). The training of the ANNs was performed using historical data in regards to T_{opt} and T_{cum} that were gathered through WoBinGO's elasticity testing. The gathered historical data counted 221 records and was split into training and test sets, with a split ratio of 80% and 20%, respectively.

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

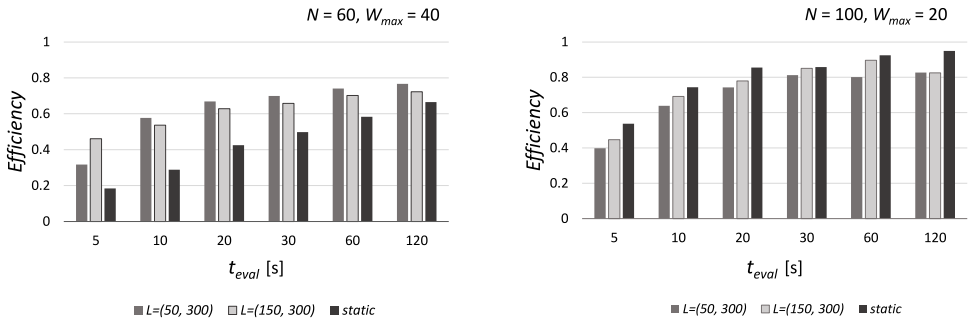

Fig. 7. Cost efficiency.

Table 2
Comparison of prediction accuracy between ANN and SVR surrogate models.

Measurement	ANN		SVR	
	T_{opt}	T_{cum}	T_{opt}	T_{cum}
RMSE [$h:mm$]	0:05	1:16	0:09	1:54
RAE	0.14	0.09	0.22	0.13
R-squared	0.98	0.99	0.93	0.98

For the ANNs' implementation, we used Keras library interfaced to *R-Project* [37]. Both ANNs consisted of 5 neurons at the input layer, one neuron at the output layer, and two hidden layers with 20 neurons each. The number of neurons in the hidden layers was selected empirically. The input vector $x = (L, W_{max}, t_{eval}, N, G)$ was identical for both ANNs. The output of the first ANN was the total optimization time T_{opt}, and the output of the second ANN was the total uptime of all engaged VMs (T_{cum}). The resilient backpropagation learning algorithm was used, with a learning rate of 10^{-3}.

Additionally, we developed the prediction models of WoBinGO's behaviour using the Support Vector Regression (SVR) approach and compared their prediction accuracy with ANN models'. To evaluate the accuracy of the fitted models the following metrics were used:

- Root mean square error:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{7}$$

- Relative absolute error

$$RAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{\sum_{i=1}^{n}|\bar{y} - y_i|} \tag{8}$$

- *R*-squared (R^2) prediction accuracy

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{9}$$

where y_i denotes the actual output, $\hat{y}_i$ is the predicted output, $\bar{y}$ is the mean of actual outputs and n is the number of observations in the dataset. The results of applied statistical measures are shown in Table 2.

From Table 2, it can be seen that all of the results go in favor of ANN prediction models. Regarding ANN accuracy measurements results, the *RMSE* for T_{opt}, was approximately 5min, which makes 3% of the full range. The RMSE of T_{cum} was 1h 16min, which makes 2% of the full range. It is also apparent from the diagrams (Fig. 8) that the trained ANNs can closely exhibit the trend of both T_{opt} and T_{cum}.

4.4. CaSe study: Calibration of the structure model

In this case study, we evaluate the reduction of the cumulative uptime (optimization cost) without affecting the duration of the optimization, using a real-world optimization problem from the field of hydroinformatics. Hydroinformatics integrates the use of numerical simulation and modelling, metaheuristic optimization algorithms, IoT, big data, data mining, HPC, cloud computing and other efficient techniques to overcome the complexity of the problems associated with hydraulics, hydrology and environmental engineering for better management of water-based systems [38,39]. Bearing in mind the importance of evolutionary algorithms for the field of hydroinformatics [40] we demonstrate how the proposed software system can be utilized for solving water-related optimization problem.

One of the most important topics in this field is the safety of hydropower facilities, which amongst others, requires calibrations of subsurface rock mass models. The purpose of a model calibration (i.e., parameter estimation) is to find the optimal parameter set for the model such that the model behaves optimally in regards to the given objectives. Our use case is the calibration of a structural finite element model of a rock mass located nearby the Iron Gate hydropower plant on the Danube river. The optimization task is to find the optimal model parameters to match series obtained from shear and pressure application experiments.

Experiments of shear and pressure application were conducted for each of the 6 groups of rocks, modeled using the Mohr–Coulomb material model [41]. In the shear test, the rock mass was subjected to normal stress in order to consolidate vertical displacements, followed by a gradual application of shear stress until the fracture. In the second test, displacement-pressure relationship was obtained by the application of a pressure to the rock mass in multiple load and release cycles.

Boundary conditions and loads applied to the finite element model corresponded to the real experiments. The Mohr–Coulomb material model had a total of 14 parameters. The values of four of them were taken from literature. The values of the rest of the parameters were estimated so the model fits experimental measurements. The objective functions were to minimize: (1) the RMSE between experimentally obtained and calculated vertical displacement; (2) the RMSE between experimentally obtained and calculated shear stress.

The multi-objective optimization problem was solved using NSGA-II algorithm [42]. The following set of algorithm parameters was adopted: a population size of 100 individuals, a maximum of 100 generations, a simulated binary crossover with a probability of 0.9 and a distribution index of 20, and polynomial mutation with a distribution index of 20 and probability of $1/l$, where l is the chromosome length (in our case 1/10). The average value of t_{eval} was 117 s, giving a total of $T_c = t_{eval} \cdot N \cdot G \approx 325h$, thus qualifying this optimization task as a computationally intensive problem.

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

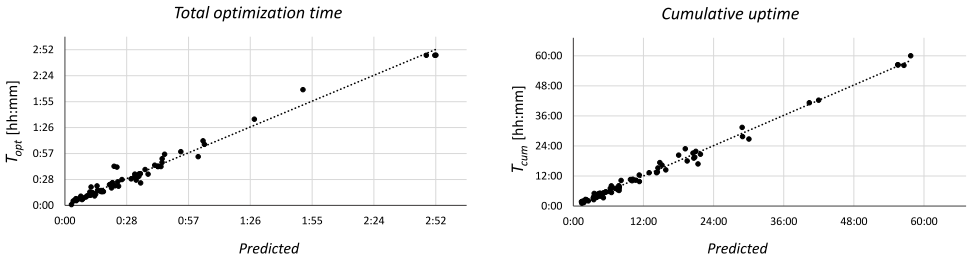

Fig. 8. Training results for total optimization time and cumulative uptime of worker instances.

Table 3
Examples of solutions from Pareto front.

Solution	Lifetime, workers	Total optimization time T_{opt}	Cumulative uptime T_{cum} (for 100 generations)
$S1$	$L = (900, 300)$, $W_{max} = 40$	10 h 52 min	421 h 28 min
$S2$	$L = (1000, 300)$, $W_{max} = 21$	17 h 21 min	378 h 47 min
$S3$	$L = (334, 300)$, $W_{max} = 30$	13 h 31 min	395 h 36 min
D	$L = (150, 300)$, $W_{max} = 20$	18 h 5 min	403 h 53 min

Before actually solving the described optimization problem, the decision support engine determines the optimal combination of instances' lifetime L and the maximum number of workers W_{max} for this particular optimization task. The ANN models of the infrastructure behavior were used as the evaluators in the process of determining the optimal (L, W_{max}) combination. The decision support engine also uses NSGA-II multi-objective GA to solve this bi-criteria optimization problem. The obtained Pareto front (Fig. 9) shows possible tradeoffs between the total optimization time and the infrastructure cost. The results are shown for 100 generations run. For different combinations of (L, W_{max}), either total optimization time drops, but the cost is very high, or the cost is lower, but optimization lasts longer. We can observe that T_{cum} ranged from approximately 379 h to 421 h, which in relative terms means that it varied by only 11%. On the other hand, total optimization time T_{opt} can differ up to 60% depending on the chosen combination of lifetime and maximum number of workers. This makes sense, since for a fixed number of evaluations, there is a fixed amount of work to be performed by the launched computing instances, and their cumulative uptime T_{cum} cannot vary significantly, regardless of the selected elasticity parameters. However, the duration of the optimization process can be largely influenced primarily by the maximum number of workers W_{max} involved in the fitness evaluations.

Table 3 shows the comparison of the solution at the very left end of the Pareto front (denoted as $S1$), and the one at the far right of the front ($S2$). It can be noticed that the difference between them in terms of the T_{opt} is 6 h 30 min, while T_{cum} differs by as much as 42 h 40 min. If a user wants to reduce not only the cost, but also the optimization duration, he can chose a solution $(L = (334, 300), W_{max} = 30)$ at the approximate middle of the Pareto front (denoted as $S3$).

If he did not use the decision support engine and went with random lifetime of $L = (150, 300)$ and W_{max} of 20 workers (point D), then the whole optimization would last more than 18 h, and the cumulative uptime would be 403 h 53 min. It is apparent that by choosing the arbitrary point D over the solutions offered by the prediction engine, one will either delay results' delivery or increase expenses. If we compare point D with solution $S1$, the T_{opt} is increased by nearly 62% with only a 4% decrease in T_{cum}. Comparing further point D with the solution $S2$, we can notice that the cost is significantly higher - nearly 25 h, while the optimization time is almost the same. By choosing D over $S3$, the increase of optimization time would be approximately 35% and uptime approximately 3%.

Selecting any Pareto optimal solution suggested by the decision support engine, one can achieve considerable savings, either in terms of optimization time or optimization cost. For the optimization tasks with more time-consuming fitness evaluations, even greater savings could be achieved, as explained in Sections 4.1 and 4.2.

5. Conclusion

We have presented Cloud extensions to an existing framework [10] for genetic algorithm based large-scale optimizations. Instead of a standard HPC stack, we employed IaaS in order to distribute computationally expensive fitness evaluations. The benchmarks were carried out with the purpose of showing to what extent a higher infrastructure overhead affects the auto-scaling capabilities of the framework, including its frugal approach to engagement of computing resources. We benchmarked the fully static pilot job approach versus WoBinGO's elastic approach in terms of pure performance, total instances' uptime and cost-to-performance ratio. The obtained results are promising, especially in the case of computationally heavy fitness evaluation functions, when the relative cost of the IaaS overhead can be neglected. Additionally, the elastic worker instance launcher gives a significantly lower cost-to-performance ratio than any static counterpart.

Our major innovation lies in optimizing the performance of large scale optimization in the Cloud environment. We propose a decision support engine that recommends optimal framework parameters to achieve minimal total execution time and total cumulative uptime for a specified optimization problem. The engine solves a bicriteria optimization problem and uses the surrogate models of the IaaS behavior under various large-scale optimization loads as a fitness evaluator in MOGA. The decision support engine output is given in the form of a Pareto front showing possible tradeoffs between overall optimization execution time and the cumulative time of engaged instances. According to a given assessment, a user can decide upon faster delivery of results or lower infrastructure cost.

The real-world simulation-based optimization clearly showed that it was possible to make significant savings with the proposed decision support engine. With more data in the training set, a better selection of training method and selection of error measures based on a validation data set, it would be possible to obtain even more robust surrogate models of infrastructure.

It is worth noting that, speaking in terms of technology readiness level (TRL) [43], WoBinGO prototype was demonstrated in

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

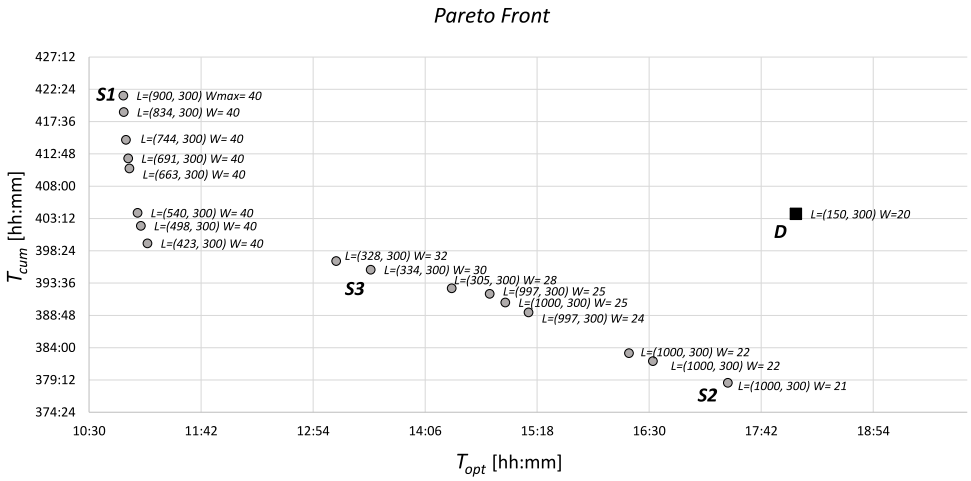

Fig. 9. Pareto front obtained by decision support engine.

an operational environment (TRL level 7), while the decision support engine was validated in a laboratory environment (TRL level 4). In future we will continue the development of decision support engine and will strive to solve the following challenges: (1) Different IaaS provider payment policies have to be taken into account, (2) Introduce and respect QoS requirements by users (3) Provide a decision support engine that helps users to determine a run scenario based on a specific provider, QoS specification, cost limits, etc. (4) Consider integrated security approach on OpenStack similar to [44], and (5) Support Cloud interoperability and federation.

Acknowledgments

Part of this research is supported by the Ministry of Education, Science and Technological Development, Republic of Serbia Grants III41007, OI174028, TR37013, III44010, R35021 and TR14005. This research is supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 777204 - SILICOFCM. This article reflects only the author's view. The Commission is not responsible for any use that may be made of the information it contains. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] K. Keahey, I. Raicu, K. Chard, B. Nicolae, Guest editors introduction: Special issue on scientific cloud computing, IEEE Trans. Cloud Comput. 4 (1) (2016) 4–5.
[2] I. Sadooghi, J.H. Martin, T. Li, K. Brandstatter, K. Maheshwari, T.P.P. de Lacerda Ruivo, G. Garzoglio, S. Timm, Y. Zhao, I. Raicu, Understanding the performance and potential of cloud computing for scientific applications, IEEE Trans. Cloud Comput. 5 (2) (2017) 358–371.
[3] L. Wang, Y. Ma, J. Yan, V. Chang, A.Y. Zomaya, Pipscloud: High performance cloud computing for remote sensing big data management and processing, Future Gener. Comput. Syst. 78 (2018) 353–368.
[4] B. Varghese, R. Buyya, Next generation cloud computing: New trends and research directions, Future Gener. Comput. Syst. 79 (2018) 849–861.
[5] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, first ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
[6] E. Alba, G. Luque, S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, Int. Trans. Oper. Res. 20 (1) (2013) 1–48.
[7] S. Cahon, N. Melab, E.-G. Talbi, Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics, J. Heuristics 10 (3) (2004) 357–380.
[8] N. Cole, T. Desell, D.L. González, F.F. de Vega, M. Magdon-Ismail, H. Newberg, B. Szymanski, C. Varela, Evolutionary algorithms on volunteer computing platforms: The milkyway@ home project, in: Parallel and Distributed Computational Intelligence, Springer, 2010, pp. 63–90.
[9] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, B.-S. Lee, Efficient hierarchical parallel genetic algorithms using grid computing, Future Gener. Comput. Syst. 23 (4) (2007) 658–670.
[10] M. Ivanovic, V. Simic, B. Stojanovic, A. Kaplarevic-Malisic, B. Marovic, Elastic grid resource provisioning with wobingo: A parallel framework for genetic algorithm based optimization, Future Gener. Comput. Syst. 42 (2015) 44–54.
[11] M. Drenovak, V. Ranković, M. Ivanović, B. Urošević, R. Jelic, Market risk management in a post-basel II regulatory environment, European J. Oper. Res. 257 (3) (2017) 1030–1044.
[12] B. Stojanovic, M. Milivojevic, N. Milivojevic, D. Antonijevic, A self-tuning system for dam behavior modeling based on evolving artificial neural networks, Adv. Eng. Softw. 97 (2016) 85–95.
[13] C. Vecchiola, M. Kirley, R. Buyya, Multi-objective problem solving with offspring on enterprise clouds, in: Proceedings of the 10th International Conference on HighPerformance Computing in Asia-Pacific Region (HPC Asia 2009), 2009, pp. 132—139.
[14] C. Vecchiola, X. Chu, R. Buyya, Aneka: a software platform for .net-based cloud computing, High Speed Large Scale Sci. Comput. 18 (2009) 267–295.
[15] R.N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds, Future Gener. Comput. Syst. 28 (6) (2012) 861–870.
[16] K. Meri, M.G. Arenas, A.M. Mora, J. Merelo, P.A. Castillo, P. García-Sánchez, J.L.J. Laredo, Cloud-based evolutionary algorithms: An algorithmic study, Nat. Comput. 12 (2) (2013) 135–147.
[17] M. García-Arenas, J.-J. Merelo, A.M. Mora, P. Castillo, G. Romero, J. Laredo, Assessing speed-ups in commodity cloud storage services for distributed evolutionary algorithms, in: Evolutionary Computation (CEC), 2011 IEEE Congress on, IEEE, 2011, pp. 304–311.
[18] M. García-Valdez, A. Mancilla, L. Trujillo, J.-J. Merelo, F. Fernández-de Vega, Is there a free lunch for cloud-based evolutionary algorithms?, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, 2013, pp. 1255–1262.
[19] M. García-Valdez, L. Trujillo, F.F. de Vega, J.J.M. Guervós, G. Olague, Evospace: a distributed evolutionary platform based on the tuple space model, in: European Conference on the Applications of Evolutionary Computation, Springer, 2013, pp. 499–508.
[20] M. García-Valdez, L. Trujillo, J.-J. Merelo, F.F. De Vega, G. Olague, The evospace model for pool-based evolutionary algorithms, J. Grid Comput. 13 (3) (2015) 329–349.
[21] W. Kurschl, S. Pimminger, S. Wagner, J. Heinzelreiter, Concepts and requirements for a cloud-based optimization service, in: Computer Aided System Engineering (APCASE), 2014 Asia-Pacific Conference on, IEEE, 2014, pp. 9–18.
[22] S. Pimminger, S. Wagner, W. Kurschl, J. Heinzelreiter, Optimization as a service: On the use of cloud computing for metaheuristic optimization, in: International Conference on Computer Aided Systems Theory, Springer, 2013, pp. 348–355.
[23] G. Leclerc, J.E. Auerbach, G. Iacca, D. Floreano, The seamless peer and cloud evolution framework, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, 2016, pp. 821–828.
[24] P. Salza, F. Ferrucci, An Approach for Parallel Genetic Algorithms in the Cloud using Software Containers, arXiv preprint arXiv:1606.06961, 2016.
[25] S. Di Martino, F. Ferrucci, V. Maggio, F. Sarro, Towards migrating genetic algorithms for test data generation to the cloud, in: Software Testing in the Cloud: Perspectives on an Emerging Discipline, IGI Global, 2013, pp. 113–135.

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.

[26] F. Ferrucci, P. Salza, F. Sarro, Using hadoop mapreduce for parallel genetic algorithms: a comparison of the global, grid and island models, Evol. Comput. 26 (4) (2018) 535–567.
[27] P. Salza, F. Ferrucci, F. Sarro, Elephant56: Design and implementation of a parallel genetic algorithms framework on hadoop mapreduce, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, ACM, 2016, pp. 1315–1322.
[28] T. Lorido-Botran, J. Miguel-Alonso, J.A. Lozano, A review of auto-scaling techniques for elastic applications in cloud environments, J. Grid Comput. 12 (4) (2014) 559–592.
[29] F.D. Muñoz-Escoí, J.M. Bernabéu-Aubán, A survey on elasticity management in paas systems, Computing 99 (7) (2017) 617–656.
[30] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H.S.-H. Chung, Y. Li, Cloud computing resource scheduling and a survey of its evolutionary approaches, ACM Comput. Surv. 47 (4) (2015) 63.
[31] E.F. Coutinho, F.R. de Carvalho Sousa, P.A.L. Rego, D.G. Gomes, J.N. de Souza, Elasticity in cloud computing: a survey, Ann. Telecommun.-Ann. Télécommun. 70 (7–8) (2015) 289–309.
[32] G. Galante, L.C.E.d. Bona, A survey on cloud computing elasticity, in: Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing, IEEE Computer Society, 2012, pp. 263–270.
[33] V.F. Rodrigues, R. da Rosa Righi, G. Rostirolla, J.L.V. Barbosa, C.A. da Costa, A.M. Alberti, V. Chang, Towards enabling live thresholding as utility to manage elastic master-slave applications in the cloud, J. Grid Comput. 15 (4) (2017) 535–556.
[34] B. Marović, M. Potočnik, B. Čukanović, Multi-application bag of jobs for interactive and on-demand computing, Scalable Comput.: Pract. Exp. 10 (4) (2009).
[35] Cloud-init Documentation, URL https://cloudinit.readthedocs.io/en/latest/, Accessed: 2018-08-28.
[36] Y. Jin, A comprehensive survey of fitness approximation in evolutionary computation, Soft Comput. 9 (1) (2005) 3–12.
[37] R.C. Team, et al., R: A language and environment for statistical computing, Vienna, Austria, 2015.
[38] M. Herrera, S. Meniconi, S. Alvisi, J. Izquierdo, Advanced Hydroinformatic Techniques for the Simulation and Analysis of Water Supply and Distribution Systems, MDPI, Multidisciplinary Digital Publishing Institute, 2018.
[39] S.K. Sood, R. Sandhu, K. Singla, V. Chang, Iot, big data and hpc based smart flood management framework, Sustain. Comput.: Inf. Syst. 20 (2018) 102–117.
[40] H.R. Maier, Z. Kapelan, J. Kasprzyk, J. Kollat, L.S. Matott, M.C. Cunha, G.C. Dandy, M.S. Gibbs, E. Keedwell, A. Marchi, et al., Evolutionary algorithms and other metaheuristics in water resources: current status, research challenges and future directions, Environ. Model. Softw. 62 (2014) 271–299.
[41] N.S. Ottosen, M. Ristinmaa, The mechanics of constitutive modeling, Elsevier, 2005.
[42] 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.
[43] European Commission. Horizon 2020 - Work Programme 2016-2017. Annex G, URL http://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2016_2017/annexes/h2020-wp1617-annex-g-trl_en.pdf, Accessed: 2019-03-15.
[44] V. Chang, Y.-H. Kuo, M. Ramachandran, Cloud computing adoption framework: A security framework for business clouds, Future Gener. Comput. Syst. 57 (2016) 24–41.

Visnja Simic holds a Ph.D. in Computer Science and a B.Sc. in Mathematics and Informatics. She is an assistant professor at the Department of Mathematics and Informatics, Faculty of Science, University of Kragujevac, Serbia. She is the co-author of several papers published in international peer reviewed journals and conference proceedings and has participated in research projects funded nationally and by the European Union. Main research interests include evolutionary multiobjective optimization with applications in hydroinformatics and high performance computing.

Boban Stojanovic is an associate professor at the Faculty of Science, University of Kragujevac, Serbia. He is expert on computer modeling and simulations, optimization, applied informatics, and software development. Boban Stojanovic has earned a Ph.D. in technical sciences from the University of Kragujevac. His research interests are in the area of numerical simulation and optimization methods, bioengineering and hydroinformatics. He is author and co-author of one monograph and more than ten publications in peer review journals and a number of simulation software. He has participated in several international scientific projects and is prime investigator in bioengineering project with Northeastern University, Boston.

Milos Ivanovic is an associate professor of Parallel Computing at the Department of Mathematics and Informatics, Faculty of Science, University of Kragujevac, Serbia. With basic degree in Physics, he holds Ph.D. in computer science. His main research interests include numerical modeling, the application of shared and distributed memory parallelism, GPU computing, grid and Cloud Computing. He has participated in several national, European and US funded projects. EU funded projects include research grants FP7-224297 ARTreat and H2020-777204 SILICOFCM. He has authored over 20 publications in international journals and numerous conference publications.

Please cite this article as: V. Simic, B. Stojanovic and M. Ivanovic, Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.042.