Simulation Modelling Practice and Theory 83 (2018) 108–123

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

Distributed simulation optimization and parameter exploration framework for the cloud☆

Michele Carillo[a], Gennaro Cordasco[b], Flavio Serrapica[a], Vittorio Scarano[a], Carmine Spagnuolo[a,*], Przemysław Szufel[c]

[a] Università degli studi di Salerno, Italy
[b] Università degli Studi della Campania "Luigi Vanvitelli", Italy
[c] Warsaw School of Economics (WSE - SGH), Poland

ARTICLE INFO

Article history:
Available online 15 December 2017

Keywords:
Agent-based simulation
Parallel computing
Distributed computing
Simulation optimization
Model exploration
Cloud computing

ABSTRACT

Simulation models are becoming an increasingly popular tool for the analysis and optimization of complex real systems in different fields. Finding an optimal system design requires performing a large sweep over the parameter space in an organized way. Hence, the model optimization process is extremely demanding from a computational point of view, as it requires careful, time-consuming, complex orchestration of coordinated executions. In this paper, we present the design of SOF (Simulation Optimization and exploration Framework in the cloud), a framework which exploits the computing power of a cloud computational environment in order to carry out effective and efficient simulation optimization strategies. SOF offers several attractive features. Firstly, SOF requires "zero configuration", as it does not require *any* additional software installed on the remote node; only standard Apache Hadoop and SSH access are sufficient. Secondly, SOF is transparent to the user, since the user is totally unaware that the system operates on a distributed environment. Finally, SOF is highly customizable and programmable, since it enables the running of different simulation optimization scenarios using diverse programming languages – provided that the hosting platform supports them – and different simulation toolkits, as developed by the modeler. The tool has been fully developed and is available on a public repository[1] under the terms of the open source Apache License. It has been tested and validated on several private platforms, such as a dedicated cluster of workstations, as well as on public platforms, including the Hortonworks Data Platform and Amazon Web Services Elastic MapReduce solution.

1. Introduction

Complex system simulation is gaining relevance in business and academic fields as a powerful experimental tool for research and management. Simulations are mainly used to analyze behaviors that are too complex to be studied analytically,

☆ A preliminary version of this paper was presented at the 24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2016), Heraklion, Crete, Greece, February 17–19, 2016.

* Corresponding author.
E-mail addresses: michele.carillo@gmail.com (M. Carillo), gennaro.cordasco@unicampania.it (G. Cordasco), flavio.serrapica@gmail.com (F. Serrapica), vitsca@dia.unisa.it (V. Scarano), cspagnuolo@unisa.it (C. Spagnuolo), pszufe@gmail.com (P. Szufel).

[1] SOF GitHub public repository, https://github.com/isislab-unisa/sof.

https://doi.org/10.1016/j.simpat.2017.12.005
1569-190X/

Table 1
Simulation-optimization algorithms can be divided into four computational categories (a list of example SO algorithms is presented). The goal of this paper is to provide a framework that allows rapid deployment of computational infrastructure supporting all of the proposed categories - with particular focus on the parallel simulation-optimization problem.

Parameter space	Computation type	
	Sequential (single thread)	Parallel (multi-process)
discrete	KG [20], IZ [35], PGS [9] VIP [12]	NHH [37], NSGS [35] OCBA [11] SKG, AKG, AOCBA [27]
continuous	Stochastic Krigging [5] RSM [33]	GP-BUCB [16], $EI^{(\mu,\lambda)}$ [26]

or too costly to be tested experimentally [28]. The representation of such complex systems results in a mathematical model comprising several parameters. Hence, the question arises as to how changes in model parameter values influence model output (simulation meta-modeling, e.g. [6]) and how to find model parameter values that yield minimum (maximum) model output (simulation-optimization e.g. see [19,47]). Considering the multi-dimensionality of the parameter space, exploring parameter values and determining the optimal parameters configuration is by no means an easy undertaking and requires extensive computing power.

In this paper a software framework for Simulations Optimization (SO) is developed. SO is understood as techniques studied for ascertaining the parameters of the model that minimize (or maximize) given criteria (one or many), which can only be computed by performing a simulation run — see [25,47]. There are several approaches to addressing simulation-optimization problems depending on model type. Simulation-optimization model types can be classified either by algorithm type or by computation type.

The following set SO model classes are considered in the literature (see [7,19]): (1) Ranking and Selection, (2) Response Surface Methodology (RSM), (3) Gradient-Based Procedures, (4) Random Search, (5) Sample Path (6) Optimization and Meta-heuristics. Since many methods exist for parallel simulation-optimization there is a need for a general computational approach to manage such a process.

From a computational point of view, the problems considered in the literature can be classified according to decision space (continuous and discrete) and parallelization (single-threaded vs multi-threaded). Most simulation-optimization policies assume sequential data processing and hence do not allow for parallelization. To illustrate the proposed classification approach a list of sample simulation-optimization algorithms and their classes has been presented in Table 1.

Branke et al. [9] note that since several approaches are available for simulation-optimization, a researcher should choose the one which has the best performance for a given problem. However, in the case of parallel, distributed algorithms, implementing the appropriate computational infrastructure is a very sophisticated task. The goal of this paper is to provide a distributed software framework that will streamline this complex process.

There are several frameworks for distributed computing including MPI, task management platforms such as SGE, and Hadoop. Ni et al. [36] compare MapReduce applicability with a message passing interface (MPI). They found out that MapReduce has a much higher overhead – in particular in scenarios where running simulations requires large amount of input data (they give a geographic system simulation as an example). However, they also note that the MapReduce model is much more appropriate for cloud computing environments, like Amazon EC2, where nodes might become unavailable during computations. Indeed, Apache Hadoop [44] offers built-in protection against core failures by having the master core periodically detect the status of worker cores and re-launch failed map or reduce tasks. Moreover, the Hadoop distributed file system (HDFS) [44], which is used by Apache Hadoop to keep MapReduce input and output, also maintains replicates of data blocks to ensure that no data is lost as a result of a single hardware failure. Therefore, the increased overhead of those fault tolerance mechanisms is often offset by the decrease in cost of using cheaper resources.

This raises the need for tools which exploit the computing power of parallel systems to improve the effectiveness and the efficiency of SO strategies. The crucial characteristics of such tools are: *zero configuration, ease of use, programmability* and *efficiency*. Zero configuration and ease of use are required because both the design and the use of SO strategies are performed by domain experts, who are seldom computer scientists and have limited knowledge of managing modern parallel infrastructures. Programmability is mandatory because different models usually require different SO strategies. Finally, the system must be efficient in order to meet the huge demand for computing power.

In the paper the specific problem of introducing or evaluating new ways to explore the parameters space is addressed — a support tool to perform the parameters space exploration and SO on stochastic simulation model is developed. The proposed framework exploits the computing power of a cloud computational environment in order to quickly carry out simulation exploration and/or optimization strategies.

The applicability of the proposed SOF framework will be illustrated by an agent-based simulation model (ABSM). ABSM enables the reproduction of complex and significant aspects of real phenomena by defining a small set of simple rules regulating how agents interact in social structures and how information is spread from agent to agent. Agent-based models have been successfully applied in several fields such as biology, sociology, economics, military and infrastructure – for a

review of ABSM applications see [32]. Since ABSMs are the most complicated simulation model types [10,15,28], there are several libraries and frameworks that speed up and facilitate the tasks of developing and testing simulations have been developed. Some examples are NetLogo [48], AnyLogic [7], MASON [30] and Repast [39,42]. Note that the applicability of SOF the framework developed in this paper is not limited only to ABSM, but rather it is considered as a representative complex simulation use case scenario.

The framework that will be described in Section 4 has been fully developed and is available on a public repository [63]. The framework has been validated on several private platforms, such as a dedicated cluster of workstations, as well as on public platforms, including the Hortonworks Data Platform and Amazon Web Services Elastic MapReduce (EMR) solution. Section 5 reports the results of several experiment performed on the framework in order to assess its effectiveness and efficiency.

2. Background

2.1. Simulations

A Simulation is an attempt to reproduce the behaviour of a real-life system over time. A system is understood as "a set of interrelated elements", where each element is connected to every other element, either directly or indirectly [2]. Law [28] defines a simulation as "numerically exercising the model for the inputs in question to see how they affect the output measures of performance". Hence, the goal of simulation is to experiment with a model of a system – observe, understand, infer and answer "what if" questions about the system. Simulations can either be used to design a novel system or for predicting the effect of changes to an existing system [11]. The main advantage of simulation is that it can be used to explore certain behaviour without causing disruption to the actual system.

2.2. Model parameter space exploration and simulation optimization

The simulation modeler usually needs to execute a large number of simulations in order to find the optimal configuration of input parameters (that is, the configuration which allows them to imitate the desired system). This process is named Parameter Space Exploration (PSE) or parameter sweep. Simulation results are evaluated using an objective (evaluating) function which associates a score with each simulation performed with a given set of parameters. As the number of the parameters of a model increases, the parameter space to be explored expands exponentially and it becomes unfeasible to handle the parameter space exploration process - which comprises parameters selection, simulations run and output evaluations - manually. Moreover, the feedback obtained from the simulation of previous configurations can be used to select future configurations to be simulated and evaluated. This circular process: (a) choice of initial configurations, (b) execution, (c) evaluation and (d) selection of new candidates, is referred to as the Simulation Optimization (SO) process [4,11,18,34,47], which can be formally presented as

$$\min_{x \in D} \Gamma(x),$$

where $D \subseteq \Theta$ is the feasible decision space, Θ is the entire parameters space, $x \in D$ is a 1-dimensional vector having size δ representing a single configuration (δ is the number of parameters of the considered simulation), and $\Gamma(x)$ is a function being estimated using simulation. The feasible decision space D can be either discrete or continuous.

Generally, the problem has a single objective (i.e., $\Gamma(x) \in R$), although multi-objective optimization problems ($\Gamma(x) \in R^n$) can also be considered. For the remainder of the paper single-objective optimization problems are considered; however, the proposed methodology can easily be applied to multi-objective optimization in a similar fashion, as described in [8]. The stochastic nature of simulation means that the output of a simulation run is not deterministic and an expected value for it is calculated as $\mathbb{E}[\Phi(x, \epsilon)]$, where $\Phi(x, \epsilon)$ is the result of a stochastic simulation run on configuration x and a random feed ϵ. Finally, $\Gamma(x) = f(\mathbb{E}[\Phi(x, \epsilon)])$is calculated, where $f(\cdot)$ is a function that evaluates the result of a simulation and calculates a single rank value. For instance, in [11], the value of $\mathbb{E}[\cdot]$ is estimated as a mean result of $r \geq 1$ simulation runs.

2.3. Distributed computing in the cloud

It should be noted that the kind of computation needed for an SO process resembles an instance of the well known bag-of-tasks application [1], i.e., an application made of a collection of independent tasks, to be scheduled on a master–worker platform. Nevertheless, a mechanism for the distribution of the task and the collection of results is required. Hence, we believe that SO processes can readily be deployed by exploiting the MapReduce (MP) programming model. Moreover, an SO process potentially requires several optimization loops in which a large amount of data is generated. One aspect that is of particular significance is the amount of inputs and outputs generated in the SO process that must be managed in a distributed storage environment. In the following we briefly introduce the MapReduce paradigm and Apache Hadoop. A natural cost-effective choice for distributed computations is the cloud that we discuss in the last subsection.

2.3.1. Mapreduce paradigm overview

MapReduce (MP) [13] is a programming model, proposed by Google, for processing large data sets by exploiting parallel/distributed computations on a set of loosely coupled machines. MP is based on two principal functions named *Map* and *Reduce*, commonly used in functional programming languages such as Lisp. Each function takes a collection of inputs pairs, expressed as a key/value combination, to compute some transformation on it. The Map function produces a collection of intermediate results while the Reduce function merges the intermediate results into a new collection of key/value pairs. Historically, MP has been used for indexing and calculating PageRank, but since its creation the research community has adopted the programming model for several purposes, in particular, when the amount of computation is large and the whole computation can be easily decomposed into smaller independent tasks.

2.3.2. Apache Hadoop

Apache Hadoop is an open-source alternative to the Google technologies: *Google File System* [22] and *MapReduce* [13]. Hadoop is the top-level of many subprojects comprising *Hadoop Distributed File System* (HDFS) and *MapReduce*. HDFS is a distributed filesystem that enables storage of a huge dataset across a distributed system. HDFS is designed to accommodate the following requirements [44]: *Large Data Sets, Simple Coherency Model, Moving Computation is Cheaper than Moving Data* (it attempts to assign a computation to a node that maintains the data instead of move the data around the nodes), *Portability Across Heterogeneous Hardware and Software Platforms* and *Hardware Failure*. Hadoop defines a specification for the Map and Reduce functions, the developers must provide the input/output specific and the implementations of Map and Reduce functions, often referred as mappers and reducers. Then the framework manages all the functionality needed to run an MP application, such as job execution, parallelization, and coordination. A typical MP program, as implemented on Hadoop, starts on a single node that launches and manages the execution of the entire distributed program on the distributed system. Subsequently, several components operate at different stages:

- *Splitter,* handles the single data source providing input pairs (key/value) to mappers.
- *Mapper,* processes a key/value pair to generate a set of intermediate key/value pairs.
- *Combiner,* also called “Local Reducer” (optional). It can help in cutting down the amount of data exchanged between Mappers and Reducers.
- *Partitioner,* also called the “Shuffle Operation”. It ensures that records with the same key will be assigned to the same Reducer.
- *Reducer,* gathers the results of the computation and concludes the job, giving output as the new collection of key/value pairs, typically stored in the HDFS.

2.3.3. Cloud computing

Cloud computing is developing very quickly as a tool to orchestrate high volume and high scale computational jobs. The cloud services offered by major companies (Amazon AWS, Microsoft Azure, Google Cloud Platform) make it possible to provision several thousands of computing cores within just minutes. The pay-as-you go model, combined with hourly billing for the provisioned computing power, is a perfect fit for any SO computational problem. The computational cluster can be scaled to match the complexity of the SO problem and hence provide the answer within a short time.

3. Related work

Since the proposed framework involves different topics, we will present the related works in three sections: Distributed computing frameworks, Parameter Space Exploration on existing tools and Parameter Space Exploration libraries and frameworks.

3.1. Distributed computing frameworks

The evolution of computer science in the last two decades has been characterized by the architectural shift that has brought the centralized computation paradigm toward distributed architectures, where data processing and data storing are cooperatively performed on several nodes, interconnected by a network.

Performing distributed computation is a significant challenge which involves job decomposition, task assignment and machine synchronization. In the following, we will briefly revise some distributed computing frameworks. See also [46] for a detailed discussion. MPI (Message Passing Interface) [21] provides a language independent standard allowing process communication. MPI provides a real implementation of the message passing protocol in any distributed computing system. This approach, requires a detailed knowledge of the considered systems while the robustness of the system has to be guaranteed by dedicated procedures. Apache Spark [53] is an in-memory distributed data analysis platform, primarily targeted at speeding up batch analysis jobs, iterative machine learning jobs, interactive query and graph processing. Apache Storm [54] is a free and open source distributed real-time computation system, focused on stream processing. Hadoop [44] is a framework or ecosystem of components, intended for carrying out batch analytics or a simple distributed task over a massive amount of distributed data. We decided to adopt Hadoop because: (1) the SO process can easily be translated into MapReduce model, which guarantees inherent scalability of the framework; (2) it provides a robust environment to ensure that no data is lost

as a result of a bounded number of failures; (3) it provides a distributed filesystem that enables storage of algorithms and input/output data.

3.2. Parameter space exploration on existing tools

Several ABM platforms enable modelers to automatically perform simulations in order to explore a pre-defined parameter space. The parameter space is defined using specific tools that enable selection of the parameters, their range and increment/decrement values, or selection by a simple list of values. Further details about the PSE on existing ABM toolkits are listed below:

- *NetLogo* [48] provides both GUI (BehaviorSpace [59]) and command line based batch run capabilities. NetLogo has a Behavior Search tool [45], which enables the use of a predefined set of model exploration heuristics (e.g. simulated annealing, genetic algorithms) for running a PSE on NetLogo models.
- *AnyLogic* [7] is a proprietary multi-method simulation toolkit. It supports agent-based, discrete event, and system dynamics simulation methodologies. The AnyLogic engine Java API enables the user to perform "Experiments", including optimization, calibration, and user-defined custom experiments. (It is worth mentioning that many of the advanced capabilities are available only for AnyLogic Professional and University Researcher editions.)
- *MASON* simulation library [30] offers a set of capabilities for creating ABSMs. Its modularity allows MASON models to be called either via the command line or as libraries from Java-based programs for model exploration purposes.
- *Repast Simphony*, [38] is the Java based toolkit of the Repast Suite. Repast Simphony has a batch functionality which divides a given parameter space into discrete sets of parameter values and executes simulations over those discrete sets in parallel. The simulations can be run on a local machine, on remote machines accessible through secure shell (ssh), in the cloud (e.g., Amazon EC2) or on some combination of the three. Using an InstanceRunner interface, Repast Simphony models can be launched by other control applications such as a bash, Portable Batch System (PBS), or Swift scripts.

None of the ABSM toolkits on their own offer the capabilities or scope, in terms of flexible, simple integration of external model exploration tools and performance on massively parallel computing resources, that the **SOF** framework aims to provide.

3.3. Parameter space exploration libraries and frameworks

In the following, the existing Parameter Space Exploration (PSE) or, in general, Simulation Optimization (SO) libraries and frameworks are briefly discussed. While most can be used as standalone tools, some of these can be used as libraries to introduce PSE/SO modules within a given ABSM framework. Most of the following software falls under the metaheuristics umbrella. For an overview of metaheuristics see [31], for reviews of more metaheuristics frameworks see [41] and for parallel metaheuristics frameworks see [3].

- OptQuest [61] is proprietary simulation optimization engine developed by OptTek Systems for metaheuristic optimization. OptQuest is also directly integrated into a number of ABSM and simulation tools (e.g., AnyLogic).
- Industrial Strength COMPASS (ISC) [51] is a library for discrete event simulation which combines metaheuristics with stochastic search algorithms.
- ECJ [56] is an open source (AFL v3) research system for evolutionary computation. ECJ can be used for developing evolutionary algorithms and general integration of simulation code on massively parallel systems. ECJ can be integrated with Java based simulation code (e.g., written with MASON or Repast Simphony) and is designed to be highly flexible: all structures in the system are arranged to be easily modifiable.
- ParadisEO [62], published under the CeCILL license, and MALLBA [58], published under a non-commercial license, are libraries of frameworks for combinatorial optimization that can deal with parallel metaheuristics methods.
- Dakota [55] combines a number of optimization, design of experiment and uncertainty quantification libraries developed by Sandia National Laboratories (e.g., DDACE, HOPSPACK), in addition to other external libraries. Dakota can be used on machines from desktops to massively parallel computers.

Closely related to this work are Extreme-scale Model Exploration With Swift/T (EMEWS) [40], Open MOdeL Experiment (OpenMOLE) [43] and Model Exploration ModulE (MEME) [23]. EMEWS [40] uses the general-purpose parallel scripting language Swift [49] to generate highly concurrent simulation workflows. These workflows enable the integration of external PSE algorithms to coordinate the running and evaluation of large numbers of simulations, in addition to making the SO process easy to implement. OpenMOLE [43] provides an execution platform that distributes simulation experiments on high performance computing environments using a domain-specific language (DSL) that is an extension of the Scala programming language. MEME [23] is based on virtual hosts specifically prepared for simulation experiments, deployed on EC2 (the Amazon Elastic Cloud).

Table 2 summarizes the characteristics of the competitor frameworks. The table shows that only EMEWS enable the user to perform several loops in order to run a Simulation Optimization algorithm. On the other hand, EMEWS uses Swift, which needs a dedicated installation and configuration. Unlike such systems, our tool is specifically designed for the optimization

Table 2
Frameworks' comparison table.

	Type	Multi-lang.	Zero-conf.	Easy to use	Program.	Tutorial
EMEWS	SO	✓	–	–	✓	✓[a]
OpenMOLE	PSE	✓	–	✓	✓	✓[b]
MEME	PSE	–	–	✓	–	na

[a] EMEWS tutorials are available at http://www.mcs.anl.gov/~emews/tutorial/.
[b] OpenMOLE tutorials are available at https://www.openmole.org/Tutorials.html.

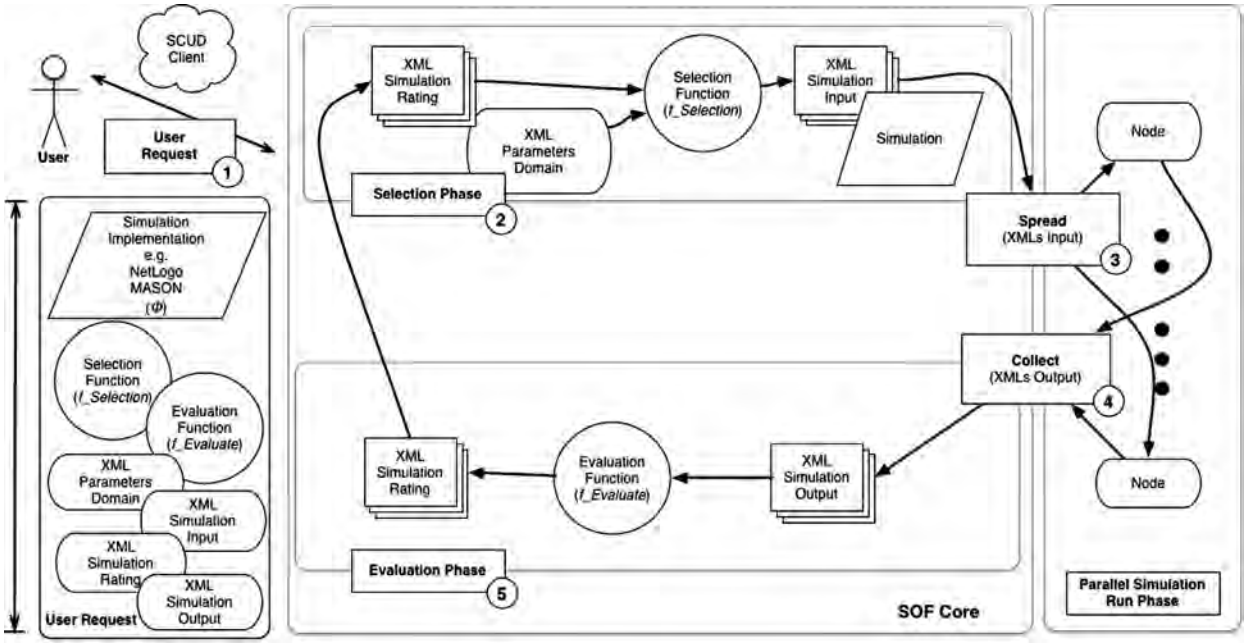

Fig. 1. SOF Work Cycle.

of simulations using an evolutionary computing strategy. Furthermore, we have designed SOF to be deployed on any Hadoop installation; only SSH access to the hosting platform is required. Moreover, our tool provides an easy-to-use environment devoted to domain experts who are seldom computer scientists and have limited knowledge of managing a modern parallel infrastructure or of developing parallel code.

4. Simulation exploration and Optimization Framework for the cloud (SOF)

We present the *Simulation exploration and Optimization Framework for the cloud* (SOF), a framework that allows us to run and collect results for two kinds of optimization scenarios: parameter space exploration (PSE) and simulation optimization (SO).

Fig. 1 depicts the SOF work cycle, which comprises three phases: selection, parallel simulation and evaluation. SOF provides a set of functionality that allows developers to construct their own simulation optimization strategy. We designed the framework based on the following objectives:

- *Zero configuration:* the framework neither requires the installation nor the configuration of any additional software, only Hadoop and SSH access to the hosting platform are required.
- *Ease of use:* the tool is transparent to the user, since the user is totally unaware that the system operates on a distributed environment.
- *Programmability:* both the simulation implementation and the simulation optimization functionalities can be implemented using different simulation toolkits (MASON, NetLogo, etc.) and/or by exploiting different programming languages, provided that the hosting platform supports them.
- *Efficiency:* by executing several independent tasks (simulations) concurrently, the framework adequately exploits the resources available on the hosting platforms.

4.1. Scenarios

In the following we will presents two algorithms – inspired by evolutionary algorithms [14] – which define the simulation optimization scenarios. The following symbols are used in the algorithms description:

- Θ, parameters space;
- $D \subseteq \Theta$, feasible decision space;
- $\mathbb{X} \subseteq D$ is a set of configurations from the feasible decision space D, $\mathbb{X} = \{x_1, x_2, \ldots, : x_i \in D\}$;
- r denotes the number of simulation run;
- $\Phi(x, \epsilon)$ denotes the results of a stochastic simulation run on configuration x and a random feed ϵ;
- $\mathbb{E}[\cdots]$ denotes the expected results of a set of stochastic simulation run;
- $\mathbb{Y}$ is the set of expected simulation results corresponding to the configuration in $\mathbb{X}$.
- t is the current optimization loop;
- $\mathbb{\Gamma}$ contains the ranking values associated with the configurations in $\mathbb{X}$;

PSE algorithm (PSE)

The PSE scenario describes a generic process of simulation optimization where a fixed set of configuration $\mathbb{X}$ is executed and all the corresponding results are collected. The Algorithm 4.1 performs this task and also handles stochastic simulations, which require several execution runs (with a different random seed) and the estimate of the expected values. The Algorithm 4.1 performs r simulations for each point $x_i \in \mathbb{X}$ and collects the simulation results in a set $\mathbb{Y}$.

Algorithm 4.1: PSE()

INPUT: $\mathbb{X}, \Phi(\cdot,\cdot)$

OUTPUT: $\mathbb{Y}$

parallel
- **for each** $x_i \in \mathbb{X}$
  - **do**
    - **for** $j \leftarrow 1$ **to** r
      - **do** $Z_j \leftarrow \Phi(x_i, \epsilon_j)$
    - $Y_i \leftarrow \mathbb{E}[Z_1, Z_2, \ldots, Z_r]$
- $\mathbb{Y} = \{Y_1, Y_2, \ldots\}$

SO algorithm (SO). The simulation optimization is a general case of PSE and corresponds to the execution of several loops of the PSE algorithm. For each optimization loop t, the set of configurations to be executed and evaluated, $\mathbb{X}_t$, depends on the results obtained from the previous loops. Each optimization loop uses the function $f\_Selection(\cdot,\cdot,\cdot)$ to generate a novel set of configurations. At the end of each loop, the function $f\_Evaluate(\cdot)$ computes the ranking values associated with each configuration in $\mathbb{X}_t$. The SO algorithm ends when the selection function returns an empty set.

Algorithm 4.2: SO()

INPUT: $D, \Phi(\cdot,\cdot), f\_Selection(\cdot,\cdot,\cdot), f\_Evaluate(\cdot)$

OUTPUT: $\{\mathbb{Y}_1, \mathbb{Y}_2, \ldots\}, \{\mathbb{\Gamma}_1, \mathbb{\Gamma}_1, \ldots\}$

$t = 1$

while $((\mathbb{X}_t = f\_Selection(D, \{\mathbb{X}_1, ..., \mathbb{X}_{t-1}\}, \{\mathbb{\Gamma}_1, ..., \mathbb{\Gamma}_{t-1}\})! = \emptyset)$

parallel
- **for each** $x_i \in \mathbb{X}_t$
  - **do**
    - **for** $j \leftarrow 1$ **to** r
      - **do** $Z_j \leftarrow \Phi(x_i, \epsilon_j)$
    - $Y_i \leftarrow \mathbb{E}[Z_1, Z_2, \ldots, Z_r]$
- $\mathbb{Y}_t = \{Y_1, Y_2, \ldots\}$
- **for each** $Y_i \in \mathbb{Y}_t$
  - **do** $\Gamma_i \leftarrow f\_Evaluate(Y_i)$
- $\mathbb{\Gamma}_t = \{\Gamma_1, \Gamma_2, \ldots\}$

$t = t + 1$

A *contributor* who implements an SO package needs to provide a functional mechanism for the definition of the feasible decision space D and the implementation of both the *f_Selection* and *f_Evaluate* functions.

Then the modeler wishing to use an implementation of SO, developed for SOF, must provide:

- the definition of D according to the mechanism provided by the SO package;
- a stochastic simulation model $\Phi(x, \epsilon)$;
- all the parameters required by the $f\_Selection(\ldots)$ and $f\_Evaluate(\ldots)$ functions.

4.2. SOF work cycle

The SOF architecture has been designed according to the *Work Cycle* shown in Fig. 1 and the algorithms shown in Section 4.1. The framework is divided into three functional blocks: *the User Front-end* (Fig. 1, left); *the SOF core*, which acts as a controller (Fig. 1, middle); *the computational resources* (Fig. 1, right).

The User front-end is implemented as a web or a standalone application through which the user provides the inputs to the system: *Simulation Implementation, Selection Function, Evaluation Function*. In order to ensure flexibility, we also include in the request to the system an XML schema for the description of Domain (*Parameters Domain*), Input (*Simulation Input*), Output (*Simulation Output*) and Rating (*Simulation Rating*). The application level of SOF provides a tool to easily generate the XML files needed. The execution of the system is described by the loop shown in Fig. 1. We summarize it in the following key phases:

1. ***User request***. The user submits the *Simulation Implementation*, the *Selection Function* and the *Evaluation Function* written using any language supported by the cloud environment. Then s/he defines the Parameters Domain, the Simulation Input, Output and Rating format in XML using the SOF XML schema.
2. ***Selection***. The system processes the request using the *Selection Function* and generates a set of parameters according to the XML schema defined by the user.
3. ***Spread***. The generated XML inputs are dynamically assigned to the computational resources. We note that our system delegates to the distributed computing environment (Hadoop in our case) both scheduling and load balancing of tasks (simulations).
4. ***Collect***. When all the simulation runs complete, the computation state is synchronized and the outputs are collected at the SOF core system according to the XML schema defined by the user, through a set of messages exchanged between the computational resources and the core system.
5. ***Evaluation phase***. The system applies the evaluation function to the collected outputs and generates the rating (again in the desired XML format).

After the evaluation phase, the system returns to the selection phase, which, also using the evaluation results obtained during the preceding steps, generates a new set of XML inputs. Obviously, the selection function also includes a stopping rule (for instance an empty set of parameters) which allows termination of the SO process.

During the spread phases, the framework executes a large number of simulations in order to achieve the results of a PSE or an SO scenario. The challenge is “How to elaborate a large number of inputs, on a distributed system, in order to ensure fault tolerance and good performance, even for different SO processes running concurrently?”. We believe that a good solution to this question is to leverage the Apache Hadoop framework. Apache Hadoop, briefly described in Section 2.3, provides some tools for managing MapReduce applications and the HDFS File System. It also provides a set of Java libraries for writing MapReduce applications. According to the language used by the Simulation Implementation, it will be possible to run the MapReduce application in several ways. For instance, when the implementation is written in Java (eg, MASON) is it possible to write a MapReduce application that initializes the simulation at code level by using mechanisms like Java Reflection. Other frameworks, like Netlogo, provide a Java library for executing simulations from a Java application. Eventually, in the case of generic implementations, the setting of simulation parameters is performed using the Java Runtime to set the input as command line arguments of the executable.

4.3. SOF architecture

As shown in Fig. 2, the system workflow presents two main entities: the SOF client and the remote host (on which Apache Hadoop is installed). The SOF architecture is divided into three main software components:

- the SOF front-end (client side), which is the SOF application for running and managing the simulation on the Hadoop infrastructure;
- the Hadoop layer (remote side), which comprises software and libraries provided by the Hadoop infrastructure;
- the SOF core composed of five functional blocks, which are used on both the client and the remote side.

SOF core. The main objective of the SOF core is to ensure flexibility in terms of the ability to use any Hadoop installation on-the-fly without requiring a specific configuration of Hadoop infrastructure or a particular software installation on the remote host. The SOF core uses the Secure SHell (SSH) protocol for the communication between the client and remote host so as to ensure the highest level of flexibility and a secure consolidated communication mechanism. Further details about what the SOF core comprises are given below:

- *Parameters Manager:* defines the XML schema of the *Parameters Domain Definition, Simulation Input Definition, Simulation Output Definition* and *Simulation Rating Definition*. It also provides routines for creating, managing and verifying the XML files.

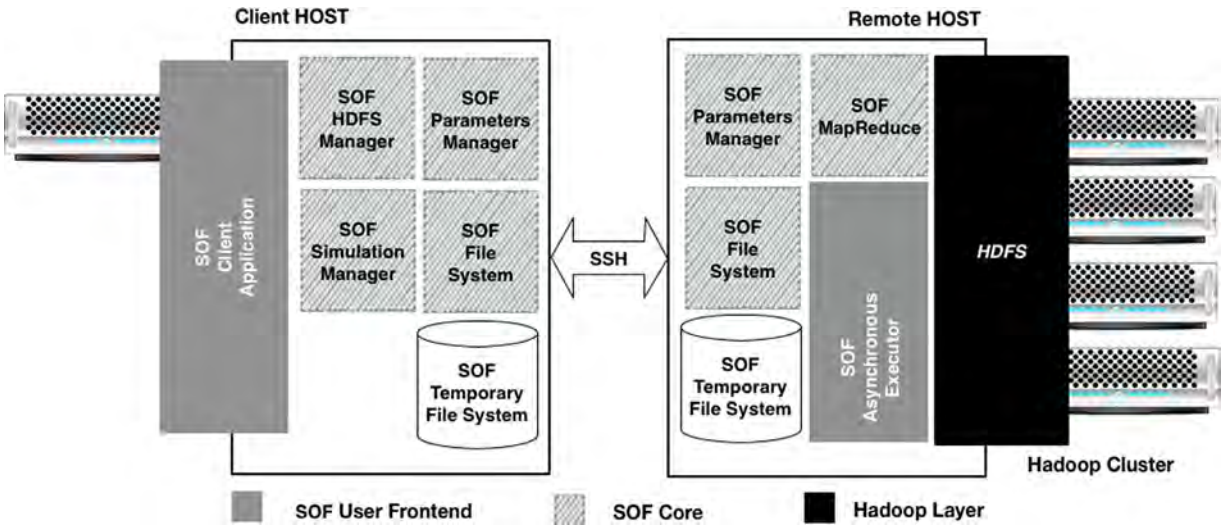

Fig. 2. SOF - Hadoop architecture.

- *File System:* defines the structure of the SOF Environment, which comprises the directories hierarchy on the HDFS, Remote and Client hosts. This block exposes routines to obtain the paths of a simulation, a simulation loop or for temporary files and folders on both the remote and client hosts.
- *HDFS Manager:* is responsible for monitoring and creating files on the HDFS.
- *MapReduce (SOF process) and Asynchronous Executor (SOF-RUNNER):* allow execution of the SO algorithm on a Hadoop environment.
- *Simulation manager:* is the fundamental block in the SOF architecture and provides the routines for executing and monitoring simulations. This block uses SSH to invoke an asynchronous execution of the SOF-RUNNER. When an SO process is started, the remote process ID is stored in the XML simulation descriptor file on the HDFS. In this way it is always possible to monitor the SO process on the remote machine and it is also possible to stop/restart or abort the SO process.

SOF–Hadoop interaction. SOF has been designed under the assumption that the remote host is a Unix machine. Therefore, the interactions between the client, remote host and Hadoop system are made using SSH and Unix commands. An important contribution of this work is that we present a novel approach to managing SO processes by embedding them in the MapReduce paradigm.

For SOF use case we use ABSM, hence we consider three types of simulation frameworks: MASON, NetLogo and simulations implemented without a programmatic framework (generic). The first two are the most relevant ABSM frameworks in the ABSM community; the latter refers to any application executable on the computation host.

In the following we describe in detail the interaction between the SOF core and Hadoop. The main events in the system are:

- *User Login:* After the user login on the Remote machine, the system automatically builds a new *SOF Environment* on the Remote machine and the HDFS and copies two programs onto the remote machine: SOF and SOF-RUNNER. SOF is the MapReduce application specialized for execution, on Hadoop, of MASON, NetLogo or a generic simulation framework. SOF-RUNNER is the SOF process manager, responsible for executing the PSE or SO algorithm exploiting the SOF MapReduce application. The Simulation Environment allows the storage of all request and output files for the simulation process. The structure of the Simulation Environment is defined by the SOF *File System;*
- *Simulation Creation:* The user prepares the Simulation Environment exploiting the features provided by the SOF frontend. Subsequently, all simulation files — *Simulation Implementation, Selection Function, Evaluation Function, Parameters Domain Definition, Simulation Input Definition, Simulation Output Definition* and *Simulation Rating Definition* are copied onto the HDFS using the structure defined by the SOF File System;
- *Simulation Submission:* The SOF core provides a routine to run a new process that launches the SOF-RUNNER via SSH on a particular simulation. The SOF-RUNNER executes the PSE or SO algorithm, exploiting the *Selection Function* and the SOF MapReduce application on the Hadoop infrastructure, for parallel execution of the *Simulation Implementation* and the Evaluation Phase.

Due to the asynchronous nature of the system and decoupling from the Hadoop infrastructure, all states of the processes are visible only by reading the state of the *SOF Environment*, which comprises the Simulation Environment of all the SO processes in the system. On the HDFS, the SOF Environment contains the state of the simulation and the state of the

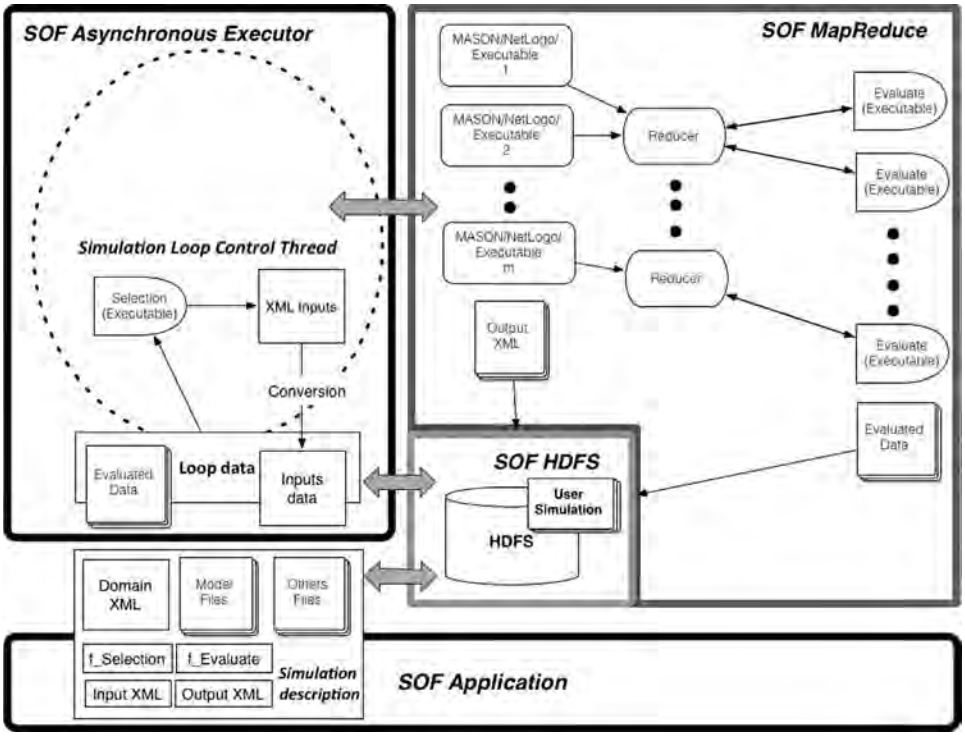

Fig. 3. Parameter Space embedding in MP and Simulation Optimization embedding in MP.

optimization loop for any SO process. On the Remote machine the SOF Environment stores the state of the SOF-RUNNER process: in this way it is also possible to stop/restart or abort any SO process. SOF has been designed for the concurrent optimization of different simulations performed by one or many users. In order to avoid the concurrency issues, SOF separates Simulation Environments and assigns unique identifiers for each SO process.

MapReduce paradigm embedding. The computation schema of an SO process, as mentioned above, resembles the well known paradigm MapReduce. In particular, we consider the parameter space Θ as a dataset of Key/Value, where the Keys are the input IDs and the Values are a feasible set of values for the simulation parameters. Two main transformations of the inputs are considered: transform the input $x \in \Theta$ into the simulation output $\Phi(x)$ and evaluate the simulation output $f(\Phi(x))$. The first transformation requires the execution of the simulation on the desired set of parameters, while the second transformation evaluates the output using an evaluating function ($f\_Evaluate(\cdot)$) according to the SO adopted scenario (see Section 4.1).

The SOF-RUNNER, depicted in Fig. 3, performs the following steps:

1. It executes the selection function using the Java Runtime. The selection function takes three arguments: the path to the input sets already executed, the path to the rating corresponding to the input sets executed, and the path where the function creates the novel input set.
2. When the selection function ends, the SOF-RUNNER transforms the input set from XML format to a standard format for the Hadoop MapReduce application and copies it on the HDFS (this is not strictly necessary because the latest version of Hadoop also supports XML input, yet to ensure compatibility with a larger number of Hadoop clusters we favor the use of a standard format).
3. It launches the SOF MapReduce application. The MapReduce application (SOF) consists of two main routines, *map* and *reduce*, as described in Section 2.3:
   (a) the *map* routine corresponds to executing the simulation and generating an output XML file, which represents the final state of the executed simulation;
   (b) the *reduce* routine executes the evaluation function, using the Java Runtime. The evaluation function takes two arguments: the path of the output XML files and the path where the function creates the rating XML file.
4. When the evaluation function ends the reducer saves the rating XML file on the HDFS.

The mapper routine should be specialized according to the specific simulation framework used. For MASON and NetLogo, the system can automatically read the final state of the simulation and generate the output XML. In order to ensure this kind of functionality, the system requires the definition of the XML format for the input and the output files. Such files do not contain any values for the parameters but are needed just to inform the system about the names of the parameters to be

Table 3
Completion time (s) with different test settings where n is the number of simulations performed per loop and p is the number of cluster nodes.

p / n	2000	4000	8000	16,000	32,000
1	1817	3109	6615	12,516	25656
2	1330	2517	3620	6562	13,420
4	1058	1440	2471	4093	7854

initialized at the beginning of the simulation and to be returned at the end. In the case of a generic simulation framework, the modeler of the simulation is responsible for generating the output files using a specific XML format in a predefined output folder.

It is worth mentioning that although the system is designed for Hadoop, by changing the SOF application it is still possible to use the system on other environments for distributed computing [1].

5. Evaluation

In the following section, we describe the benchmarks used to evaluate SOF scalability on a Hadoop cluster machine and present a genuine use case on a cloud computing provider: Amazon EMR.

5.1. Scalability evaluation on a Hadoop cluster machine

We have tested a simple SO process using the NetLogo Fire model [50]. This model simulates the spread of a fire through a forest. It shows that the fire-s chance of reaching a particular point in a forest (e.g. right border) depends critically on the density of trees. This is an example of a common feature of complex systems, the presence of a non-linear threshold or critical parameter. In particular, at 59% density, the fire has a 50/50 chance of reaching the right edge. The Fire model has also been used to validate the OpenMOLE platform [43].

Since we are evaluating the performance of the framework, the SO process is based on an empty $f\_Evaluate(\cdot)$ function, while the $f\_Selection(\cdot,\cdot,\cdot)$ function generates a set of n configurations for the first 10 loops and an empty set at the end of the 10th loop, so that the SO process terminates. Each configuration consists of the density parameter and a seed for the random number generator. All the simulations perform 1000 simulation steps.

The Simulation Environment. Simulations have been performed on a Hadoop cluster of four nodes, each equipped as follows:

- Hardware:
  - CPUs: 2 x Intel(R) Xeon(R) CPU E5-2680 @ 2.70 GHz (#core 16, #threads 32)
  - RAM: 256GB
  - Network: adapters Intel Corporation I350Gigabit
- Software:
  - Ubuntu 12.04.4 LTS (GNU/Linux 3.11.0-15-generic x86_64)
  - Java JDK 1.6.25
  - Apache Hadoop 2.4.0

Experimental results. We executed 12 test setting experiments, varying both the number of cluster nodes ($p \in \{1, 2, 4\}$) and the number of configurations generated per loop ($n \in \{2000, 4000, 8000, 16, 000, 32, 000\}$). Such values have been selected in order to analyze both the strong and weak scalability of the framework. Table 3 depicts the completion time in seconds required for the execution of each test. Results show that the system scales fairly well, especially when the number of configurations is large. This result was unsurprising given that the tasks are all independent and do not generate any communication overhead.

Strong and Weak Scalability. In order to better evaluate the scalability efficiency of the framework, we computed both the weak and strong scaling efficiency.

The strong scaling efficiency measures the capability of the framework to complete a set of simulations in a reasonable amount of time. In order to compute the strong scalability efficiency, the problem size stays fixed but the number of processing elements are increased. The strong scalability efficiency (as a percentage of the optimum) is given by

$$\frac{t_{1,n}}{p \times t_{p,n}} \times 100\%$$

where $t_{i,\,n}$ denotes the completion time to perform the overall set of n simulations using i cluster nodes. In our cases the strong scaling efficiency ranges from 42.93% ($p = 4, n = 2000$) and 95.59% ($p = 2, n = 32{,}000$).

The weak scaling efficiency measures the capability of the framework to solve larger problems as the number of processing elements increases. In order to compute the strong scalability efficiency, the problem size assigned to each processing

element stays constant and additional elements are used to solve a larger problem. The weak scalability efficiency (as a percentage of the optimum) is given by

$$\frac{t_{1,n}}{t_{p,n\times p}} \times 100\%$$

and in our cases is equal to 100% (for $p = 2$ and $n = 8000$) and 73.53% (for $p = 4$ and $n = 2000$). Finally, on the larger problem ($n = 32{,}000$) the speedup measured as

$$S_p = \frac{t_{1,n}}{t_{p,n}}$$

is $S_2 = 1.91$ and $S_4 = 3.27$.

5.2. SOF genuine use case

Optimal Computing Budget Allocation. For the real case evaluation we have chosen a Ranking and Selection SO problem class. We show how the proposed SOF framework can be integrated with the classic Optimal Computing Budget Allocation (OCBA) mechanism [11]. Since in our solution each worker takes a single additional simulation to evaluate and the standard OCBA is not numerically stable for computational budgets equal to 1, we used the *purified OCBA* approach proposed in [27]. Please note that the SOF framework provides a general abstract mechanism for running SO computations - hence the OCBA model is here just a *proof-of-concept* example.

OCBA allocates every solution a certain number of replications in the initial loop to obtain a general idea about these solutions. Then, over successive loops the best candidate solutions are analyzed with greater detail (increasing the number of replications). The classic OCBA mechanism is unsuitable when the solution space is large, continuous, or even unbounded. To tackle this problem several OCBA integrations with search algorithms have been developed (see [29] and references quoted therein). In the OCBA mechanism n points are considered. In the first step of the procedure, for each point k, $k \geq 2$, simulations are run. Next, for each design point i, $i = 1 \ldots n$, the mean value μ_i and the standard deviation σ_i, are calculated. Finally, for a given computational budget T, μ_i and σ_i, new computations allocations m_i are calculated where $m_1 + \ldots + m_n = T$. Further details can be found in [24] and implementation details for small computational budget increments can be found in [27]. Full implementation source code is available in the SOF GitHub repository [63].

Simulation model. We have tested a simple SO process using the Wealth Distribution model [60].

The model simulates the "rich get richer" effect in the distribution of wealth. It is adapted from Epstein & Axtell's "Sugarscape" model [17] but uses grain instead of sugar. The field is a bi-dimensional space of 50×50 patches. Each patch has an amount of grain and a grain capacity (the amount of grain it can grow). People collect grain from the patches, and eat the grain to survive. How much grain each person accumulates constitutes his or her wealth. The model begins with a roughly equal wealth distribution. The people then wander around the landscape gathering as much grain as they can. Each person attempts to move in the direction where the most grain lies.

The model is characterized by several parameters. Among them, in our experiment we look for an optimization of the following parameters:

- PERCENT-BEST-LAND, which determines the initial density of patches that are seeded with the maximum amount of grain. It ranges from 5 to 25%;
- NUM-GRAIN-GROWN, which determines how much grain is grown after a certain number of ticks. It ranges from 1 to 10;
- LIFE-EXPECTANCY-MAX is the longest number of ticks that a person can possibly live. It ranges from 1 to 100;

We assume that the modeler aims at maximizing the inequality of wealth. We measured the inequality of the system using the Gini-Index. The Gini-Index is a measure of statistical dispersion intended to represent the income or wealth distribution, and is the most commonly used measure of inequality. The Gini-Index ranges from 0 to 1. A Gini-Index of 0 expresses perfect equality, where all values are the same. A Gini-Index of 1 expresses maximal inequality among values (e.g., where only one person has all the income or consumption, and all others have none).

The Simulation Environment. Simulations have been performed on Amazon Elastic MapReduce (Amazon EMR) which exploits Amazon Web Services (AWS).

Amazon Web Services (AWS) is a scalable and highly reliable cloud computing infrastructure that offers on demand cloud computing infrastructure. AWS provides different services on the cloud. In this work, we are interested in the web services that enable either the modeler or the developer to run their simulation on the cloud. The Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the cloud. In terms of abstraction layers, Amazon EC2 is an instance of the Infrastructure as a Service (IaaS) model, where the Amazon infrastructure is seen as a complete virtual environment which allows execution of different instances of virtual machines.

Amazon Elastic MapReduce (Amazon EMR) [52] is a cluster of virtual servers running on the Amazon cloud. It is managed using the open-source framework Apache Hadoop. Amazon EMR enables Hadoop applications to work seamlessly with AWS. The EMR architecture assumes that a Hadoop clusters running on Amazon EMR uses EC2 instances as virtual Linux servers. One of these virtual servers acts as the master node of the Hadoop cluster and the others as slave nodes. AWS recommends

Table 4
Amazon EC2 general purpose instances comparison.

Name	Memory	Processor	vCPUs	Instance Storage	Network Speed	Linux on Demand cost
m1.medium	3.75GB	Xeon E5-2650	1 vCPUs	410GB HDD	Moderate	$ 0.087 hourly
m1.large	7.5GB	Xeon E5-2650	2 vCPUs	840GB (2 × 420GB HDD)	Moderate	$ 0.175 hourly
m1.xlarge	15.0GB	Xeon E5-2650	4 vCPUs	1680GB (4 × 420GB HDD)	High	$ 0.350 hourly
m3.xlarge	15.0GB	Xeon E5-2670 v2	4 vCPUs	80GB (2 × 40GB SSD)	High	$ 0.266 hourly
m3.2xlarge	30.0GB	Xeon E5-2670 v2	8 vCPUs	160GB (2 × 80GB SSD)	High	$ 0.532 hourly
c4.2xlarge	15.0GB	Xeon E5-2666 v3	8 vCPUs	EBS only	High	$ 0.398 hourly
m4.2xlarge	32.0GB	Xeon E5-2686 v4	8 vCPUs	EBS only	High	$ 0.431 hourly
r4.2xlarge	61.0GB	Xeon E5-2686 v4	8 vCPUs	EBS only	Up to 10 Gigabit	$ 0.532 hourly

Table 5
Time in seconds and cost ($) required by the first loop of the SO process evaluating 2000 simulation configurations.

Name	vCPUs	Number of Slave instances													
		2		4		8		16		32		64		128	
		time (s)	cost ($)	time (s)	cost ($)	time (s)	cost ($)	time (s)	cost ($)	time (s)	cost ($)	time (s)	cost ($)	time (s)	cost ($)
m1.medium	1	3572	0.26	2070	0.25	1099	0.24	678	0.28	553	0.44	420	0.66	377	1.18
m1.large	2	2043	0.30	1193	0.29	728	0.32	494	0.41	345	0.55	306	0.96	246	1.54
m1.xlarge	4	1288	0.38	806	0.39	550	0.48	520	0.86	444	1.42	465	2.94	427	5.35
m3.xlarge	4	1100	0.24	591	**0.22**	400	0.27	**305**	0.38	207	0.50	173	0.83	121	1.15
m3.2xlarge	8	626	0.28	383	0.28	294	0.39	315	0.79	303	1.48	270	2.59	301	5.74
c4.2xlarge	8	628	0.21	278	**0.15**	208	0.21	159	0.30	103	0.37	**92**	1.31	94	1.34
m4.2xlarge	8	554	0.18	356	0.20	261	0.26	275	0.52	277	1.01	283	2.04	292	4.18
r4.2xlarge	8	591	0.26	319	0.24	217	0.29	172	0.43	113	0.55	99	0.95	113	2.15

to use the Amazon S3 storage service in place of HDFS service for persistent data storage in Hadoop cluster — this approach enables a significant cost reduction. EMR cluster performance can be continuously monitored with the CloudWatch service. In turn, the CloudWatch service can be integrated with autoscaling services — EMR Hadoop cluster slave nodes can be automatically added on-the-fly once the cluster performance is insufficient.

Simulations have been performed using $p \in \{2, 4, 8, 16, 32, 64, 128\}$ slave nodes plus 1 master node, using different Amazon EC2 *instance types*. In particular, we chose all the instance types from the *general purpose instance class*: {*m*1.*medium, m*1.*large, m*1.*xlarge, m*3.*xlarge, m*3.2*xlarge*} plus one instance type for each of the remaining instance classes: *c*4.2*xlarge*, which belong to the class *Compute optimized, m*4.2*xlarge*, which belong to the class *Memory optimized* and *r*4.2*xlarge*, which belong the class *Storage optimized*. Overall, $I =$ {*m*1.*medium*, *m*1.*large*, *m*1.*xlarge*, *m*3.*xlarge*, *m*3.2*xlarge*, *c*4.2*xlarge*, *m*4.2*xlarge*, *r*4.2*xlarge*}.

The features of each instance type are reported in Table 4:

Experimental results. We executed 56 experiments evaluating 8 Amazon EC2 instance types with 2,4,8,16,32,64,128 slave nodes — see Table 5.

All the experiments were conducted on a population of 1000 agents. The SO process is composed of 10 loops, evaluating at most 200 stochastic simulation configurations per loop with at most 10 simulation replicas (varying the random seed). Overall, at most 2000 simulation runs are executed for each loop.

For each experiment we collected the timing and the maximum obtained Gini-Index. All the experiments were able to find an optimal (or nearly optimal) configuration, which in our case corresponds to a Gini-Index equal to 0.506. In terms of speed, the following table gives the timing recorded during the first loop (which in each experiment comprises 2000 simulation runs).

The table also shows the per loop cost of each configuration setting. Results show that the costs increase gradually (almost constant up to 16 slave nodes) for low powerful instances while they tend to increase with the number of instances on powerful instances.

Comparing the different instances, it is worth noting that the best choice seems to be the *c4.2xlarge* instance type which provides the best performance both in terms of efficiency (only 92s to perform the first loop, using 64 slave instances) and in terms of cost (only $ 0.15 to perform the first loop, using 4 slave instances).

Fig. 4 depicts the trend of both the price and the time required by experiments on the *c4.2xlarge* instance type, according to the number of vCPUs used. Specifically, the *X* axis represents the number of vCPUs used, the price trend is shown by histograms whose scale is presented on the left hand side of the figure, and finally the time required by experiments shown by the curve whose scale is presented on the right hand side of the figure.

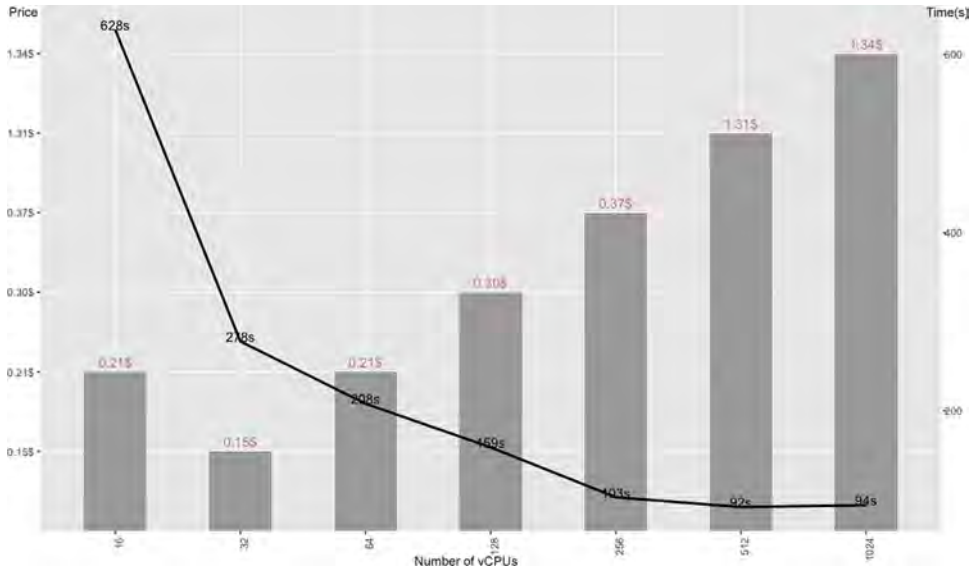

Fig. 4. Price vs Time trends on the *c4.2xlarge* instance using up to 1024 vCPUs.

6. Conclusion

This paper presents the design and implementation of SOF, a framework which exploits the computing power of a cloud computational environment in order to accomplish effective and efficient simulation optimization strategies. The framework has been specifically designed for ABM simulation, in particular for the MASON and NetLogo Simulation Environments, but supports any simulation executable, provided that the hosting platform supports them. Two simulation scenarios, Parameter Space Exploration and Simulation Optimization, have been deployed. We have showed how to embed these kinds of scenarios in a distributed computation using a consolidated infrastructure for distributed computing. SOF offers the following interesting features: (i) *"zero configuration"*: SOF does not require any additional installation on the remote host – only Apache Hadoop and SSH access are needed to set-up and run the system; (ii) *ease of use*: the user is totally unaware that the system operates on a distributed environment, so SOF does not require specific skills on the part of users; (iii) *programmability*: SOF allows the user to run different simulation toolkits and/or exploit diverse programming languages. Preliminary evaluations show that the system is scalable, especially when the SO process involves a large number of concurrent independent simulations. We have validated the proposed framework with an agent-based simulation model, but the proposed approach can be use to run and find optimal parameters for simulation models of any type. The tool has been fully developed and is available on a public GitHub repository [63] under the Apache public license. It has been tested and validated on top of several private platforms, such as a dedicated cluster of workstations, as well as public platforms, including the Hortonworks Data Platform [57] and Amazon Web Services EMR solution. SOF also includes both a command line interface and a GUI client, which can be used either by a contributor – designing an SO package – or by a modeler – to set-up and run an SO process.

Acknowledgments

We are grateful to the reviewers for their careful reading of the paper and for their helpful comments.

This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 645860.

References

[1] M. Adler, Y. Gong, A.L. Rosenberg, Optimal sharing of bags of tasks in heterogeneous clusters, in: Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures, 2003.
[2] R. Ackoff, Towards a System of Systems Concepts, 17(11), Institute of Management Sciences, 1971, pp. 661–671.
[3] E. Alba, G. Luque, S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, International Transactions in Operational Research, 2013.
[4] A. Ammeri, W. Hachicha, H. Chabchoub, F. Masmoudi, A comprehensive literature review of mono-objective simulation optimization methods, Adv. Prod. Eng. Manag. 6 (4) (2011).

[5] B. Ankenman, B.L. Nelson, J. Staum, Stochastic kriging for simulation metamodeling, Oper. Res. 58 (2) (2010) 371–382.
[6] R.R. Barton, Simulation metamodels, in: Proceedings of Winter Simulation Conference, 1998.
[7] A. Borshchev, Y. Karpov, V. Kharitonov, Distributed simulation of hybrid systems with anylogic and HLA, Futur. Gener. Comput. Syst. 18 (6) (2002) 829–839.
[8] J.P. Brans, B. Mareschal, Multiple criteria decision analysis: State of the art surveys, in: G. Salvatore (Ed.), International Series in Operations Research & Management Science, Springer Science, 2005.
[9] J. Branke, S.E. Chick, C. Schmidt, Selecting a selection procedure, Manag. Sci. 53 (12) (2007) 1916–1932.
[10] B. Calvez, G. Hutzler, Parameter space exploration of agent-based models, in: Proceedings of Knowledge-Based Intelligent Information and Engineering Systems, 2005, pp. 633–639.
[11] C. Chen, L. Lee, Stochastic Simulation Optimization: An Optimal Computing Budget Allocation, World Scientific Publishing Co. Inc., 2010.
[12] S.E. Chick, K. Inoue, New two-stage and sequential procedures for selecting the best simulated system, Oper. Res. 49 (5) (2001) 732–743.
[13] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[14] L.N. De Castro, Fundamentals of natural computing: basic concepts, algorithms, and applications, Computer and Information Science Series, Chapman & Hall/CRC, 2006.
[15] A. Deckert, R. Klein, Simulation-based optimization of an agent-based simulation, NETNOMICS: Econ. Res. Electron.Networking 15 (1) (2014) 33–56.
[16] T. Desautels, A. Krause, J.W. Burdick, Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization, J. Mach. Learn. Res. 15 (1) (2014) 3873–3923.
[17] J.M. Epstein, R. Axtell, Growing Artificial Societies: Social Science from the Bottom Up, The Brookings Institution, Washington, DC, USA, 1996.
[18] M.C. Fu, F.W. Glover, J. April, Simulation optimization: a review, new developments, and applications, in: Proceedings of Winter Simulation Conference, 2005.
[19] M.C. Fu, Handbook of Simulation Optimization, 216, Springer, 2015.
[20] P.I. Frazier, W.B. Powell, S. Dayanik, A knowledge-gradient policy for sequential information collection, SIAM J. Control Optim. 47 (5) (2008) 2410–2439.
[21] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, T.S. Woodall, Open MPI: goals, concept, and design of a next generation MPI implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, 2004, pp. 97–104.
[22] S. Ghemawat, H. Gobioff, S. Leung, The Google file system, ACM SIGOPS Oper.Syst.Rev. 37 (5) (2003).
[23] L. Gulyás, A. Szabó, R. Legéndi, T. Máhr, R. Bocsi, G. Kampis, Tools for large scale (distributed) agent-based computational experiments, in: Proceedings of Computational Social Science Society of the Americas CSSSA, 2011.
[24] D. He, S.E. Chick, C.-H. Chen, Opportunity cost and OCBA selection procedures in ordinal optimization for a fixed number of alternative systems, IEEE Trans. Syst. Man Cybern.Part C 37 (5) (2007) 951–961.
[25] D. He, L.H. Lee, C. Chen, M. Fu, S. Wasserkrug, Simulation optimization using the cross-entropy method with optimal computing budget allocation, ACM Transactions on Modeling and Computer Simulation (TOMACS), 20, 2010.
[26] J. Janusevskis, R.L. Riche, D. Ginsbourger, R. Girdziusas, Expected improvements for the asynchronous parallel global optimization of expensive functions. potentials and challenges, in: Learning and Intelligent Optimization, Springer, Berlin, Heidelberg, 2012, pp. 413–418.
[27] B. Kamiński, P. Szufel, On parallel policies for ranking and selection problems, J. Appl. Stat. (2017), doi:10.1080/02664763.2017.1390555. Taylor & Francis
[28] A.M. Law, Simulation Modeling and Analysis, McGraw-Hill, 2007.
[29] L.H. Lee, C.H. Chen, E.P. Chew, J. Li, N.A. Pujowidianto, S. Zhang, A review of optimal computing budget allocation algorithms for simulation optimization problem, Int. J. Oper. Res. 7 (2) (2010) 19–31.
[30] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan, MASON: a multiagent simulation environment, Simulation 81 (7) (2005).
[31] S. Luke, Essentials of Metaheuristics (Second Edition), 2013.
[32] C.M. Macal, M.J. North, Tutorial on agent-based modeling and simulation, in: Proceedings of Winter Simulation Conference, 2005.
[33] R.H. Myers, A.I. Khuri, W.H. Carter, Response surface methodology, Technometrics 31 (2) (1989) 137–157.
[34] B.L. Nelson, Optimization via simulation over discrete decision variables, in: Risk and Optimization in an Uncertain World, Informs, 2010, pp. 193–207.
[35] B.L. Nelson, J. Swann, D. Goldsman, W. Song, Simple procedures for selecting the best simulated system when the number of alternatives is large, Oper. Res. 49 (6) (2001) 950–963.
[36] E.C. Ni, D.F. Ciocan, S.G. Henderson, S.R. Hunter, Comparing message passing interface and mapreduce for large-scale parallel ranking and selection, in: Proceedings of the 2015 Winter Simulation Conference, 2015, pp. 3858–3867.
[37] E.C. Ni, S.R. Hunter, S.G. Henderson, A comparison of two parallel ranking and selection procedures, in: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 3761–3772.
[38] M.J. North, J. Ozik, E.R. Tatara, C. Macal, M. Bragen, P. Sydelko, Complex adaptive systems modeling with repast simphony, Complex Adapt. Syst. Model. 1 (3) (2013).
[39] M. North, T. Howe, N. Collier, J. Vos, A declarative model assembly infrastructure for verification and validation, Advancing Social Simulation: The First World Congress, 2007.
[40] J. Ozik, N. Collier, J. Wozniak, C. Spagnuolo, From desktop to large-scale model exploration with swift/t, in: Proceedings of Winter Simulation Conference (WSC), 2016, pp. 206–220.
[41] J.A. Parejo, A. Ruiz-Cortsa, S. Lozano, P. Fernandez, Metaheuristic optimization frameworks: a survey and benchmarking, Soft Computing, 2011.
[42] S. Railsback, L. Railsback, S. Lytinen, S. Jackson, Agent-based simulation platforms: review and development recommendations, Simulation 82 (9) (2006).
[43] R. Reuillon, M. Leclaire, S. Rey-Coyrehourcq, OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models, in: Future Generation Computer Systems, volume 29, 2013, pp. 1981–1990.
[44] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop distributed file system, in: Proceedings of the 2010 IEEE Symposium on Mass Storage Systems and Technologies (MSST), 2010.
[45] F. Stonedahl, Genetic algorithms for the exploration of parameter spaces in agent-based models, 2011. P.h.D. Thesis.
[46] M.T.d. Silva, Survey on frameworks for distributed computing: Hadoop spark and storm, in: Proceedings of the 10th Doctoral Symposium in Informatics Engineering, 2015, pp. 29–30.
[47] E. Tekin, I. Sabuncuoglu, Simulation optimization: a comprehensive review on theory and applications, IIE Trans. 36 (11) (2004).
[48] S. Tisue, NetLogo: design and implementation of a multi-agent modeling environment, in: Proceedings of Agent, 2004.
[49] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, I. Raicu, Parallel scripting for applications at the petascale and beyond. computer, Computer 42 (2009).
[50] U. Wilensky, Netlogo fire model, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1997. http://ccl.northwestern.edu/netlogo/models/Fire.
[51] J. Xu, B.L. Nelson, J.L. Hong, Industrial strength COMPASS: a comprehensive algorithm and software for optimization via simulation, ACM Trans. Model. Comput. Simul. 20 (2010) 1–29.
[52] Amazon elastic MapReduce developer guide. http://docs.aws.amazon.com/emr/latest/DeveloperGuide/emr-dg.pdf, last accessed 2017-09-25.
[53] Apache spark. https://spark.apache.org/, last accessed 2017-09-25.
[54] Apache storm. http://storm.apache.org/, last accessed 2017-09-25.
[55] Dakota. https://dakota.sandia.gov, last accessed 2017-10-05.
[56] ECJ. https://cs.gmu.edu/~eclab/projects/ecj/, last accessed 2017-11-07.

[57] Hortonworks data platform, the completely open source Apache Hadoop data platform, architected for the enterprise. http://hortonworks.com/hdp/, last accessed 2017-09-27.
[58] Mallba.http://neo.lcc.uma.es/mallba/easy-mallba/, last accessed 2017-09-25.
[59] Netlogo behaviorspace. http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html, last accessed 2017-10-07.
[60] Netlogo: Wealth distribution. http://ccl.northwestern.edu/netlogo/models/WealthDistribution, last accessed 2017-10-07.
[61] Opttek systems. http://www.opttek.com, last accessed 2017-10-03.
[62] ParadisEO. http://paradiseo.gforge.inria.fr, last accessed 2017-10-02.
[63] Simulation optimization and exploration framework on the cloud: SOF — public github repository, isislab-unisa/sof GitHub repository. https://github.com/isislab-unisa/sof.