J Grid Computing (2019) 17:749–770
https://doi.org/10.1007/s10723-019-09476-0

Performability Evaluation and Optimization of Workflow Applications in Cloud Environments

Danilo Oliveira · André Brinkmann · Nelson Rosa · Paulo Maciel

Received: 13 April 2018 / Accepted: 6 January 2019 / Published online: 17 January 2019

Abstract Given the characteristics of dynamic provisioning and illusion of unlimited resources, clouds are becoming a popular alternative for running scientific workflows. In a cloud system for processing workflow applications, the system's performance is heavily influenced by two factors: the scheduling strategy and failure of components. Failures in a cloud system can simultaneously affect several users and depreciate the number of available computing resources. A bad scheduling strategy can increase the expected makespan and the idle time of physical machines. In this paper, we propose an optimization method for the scheduling of scientific workflows on cloud systems. The method comprises the use of a meta-heuristic algorithm coupled to a performability model that provides the fitnesses of explored solutions. For being able to represent the combined effect of scheduling and component failures, we adopted discrete event simulation for the performability model. Experimental results show the effectiveness of the hybrid simulation-optimization approach for optimizing the number of allocated virtual machines and the scheduling of tasks regarding performability.

Keywords Scientific workflows · Performability · Stochastic petri nets · Optimization

D. Oliveira (✉) · N. Rosa · P. Maciel
Federal University of Pernambuco, Informatics Center, Recife, Brazil
e-mail: dmo4@cin.ufpe.br

Nelson Rosa
e-mail: nsr@cin.ufpe.br

Paulo Maciel
e-mail: prmm@cin.ufpe.br

A. Brinkmann
Data Processing Center (ZDV), Johannes Gutenber University, Mainz, Germany
e-mail: brinkman@uni-mainz.de

1 Introduction

In the past decades, computers become an invaluable asset for scientists in many fields of human knowledge. Simulation models are useful when experiments in the real world are too difficult or costly to execute, or when the phenomenon of interest is impossible to reproduce (for instance, in studies about the origin of the universe). Such models are often computationally intensive and require an execution environment composed of many processing units. Many computational scientific applications can be expressed as workflows, i.e., a set of subtasks with data and control-flow dependencies between them. In such applications, the scheduling of tasks in the processing units plays a vital role in the system's performance, but finding an optimal schedule is an NP-Hard problem [8].

Nowadays, cloud computing has been attracting attention as a platform for running scientific applications [25, 27, 55]. The pay-per-use model eliminates the need for upfront investment on a cluster/supercomputer. Moreover, cloud users do not need

to worry about managing underlying hardware infrastructure. While this model makes things more convenient for the user, this task becomes a severe issue for cloud providers, who need to guarantee reliability and performance levels specified by a Service Level Agreement (SLA).

The complexity of cloud infrastructures (i.e., the large number of hardware and software components and their interdependence relationships) raises the need for performance evaluation methods that consider the failure of components. A failure in a single physical server can bring down several virtual machines of different users. Likewise, the unavailability of the cloud manager (the head node of a cloud infrastructure) can provoke the unavailability of the whole system. A classic performance study (that disregards reliability aspects) may not give accurate results since failures of physical and virtual machines can increase waiting times and decrease throughput [43]. To assess the performance degradation caused by failures in a cloud environment via measurement-based evaluation is often prohibitive in practice. Even using fault injectors to provoke failures on the system's components, the associated costs and time constraints for such experiments are high, especially when testing multiple configurations in a sensitivity analysis (i.e., a systematic study of the impact of system parameters on system's performance/reliability [24]). For that reason, state-space based models (e.g., Markov chains [7], Stochastic Petri Nets (SPN) [38], Stochastic Automata Networks [42]) are the most employed technique for performability evaluation of cloud and grid systems [19, 43, 45, 57]. Besides the occurrence of failures, inefficient scheduling strategies can harm the overall system performance by increasing the job makespan and reducing the utilization of processing units.

Evaluating the effects of both scheduling and hardware/virtual machine failures on performability of workflow applications in cloud environments is a challenging task for space-state based models due to the high number of states to be considered. Moreover, the exponential distribution may not be a good fit for computation times in workflows. Given this intrinsic limitation of space-based models, many existing research efforts towards the modeling of cloud applications employ discrete event simulators. CloudSim [13] is the most widely adopted simulation framework in the cloud computing literature. Thanks to its adaptable architecture, many extensions were proposed to address aspects not originally implemented by CloudSim. Some of the extensions feature autoscaling [11], federated clouds [30], fault tolerance mechanisms [1, 16, 65], and workflow applications [10, 14, 35]. Nevertheless, covering multiple real-world characteristics of workflow applications in the same cohesive model is still a gap in the literature. Such characteristics include non-determinism, multitenancy, and the combined effect of hardware and virtual machine failures.

In this work, we propose an optimization method for the scheduling of scientific workflows running in a cloud environment. The throughput of workflow jobs is the problem's objective function. Each request processed by the cloud system is defined by a graph of subtasks with precedence constraints between them. A single job demands the provisioning of a certain number of virtual machines for running the subtasks in parallel. The model used to compute the objective function can measure the impact of inefficient scheduling and failures of components on the system's throughput. Given the presence of stochastic components on the proposed model, our method applies discrete-event simulation for evaluating the objective function. We developed a Stochastic Petri net generator algorithm for creating performance and reliability models of cloud workflow applications. Our experimental results demonstrate the effectiveness of the hybrid Simulation/Optimization approach for optimizing the considered objective function. We also present a sensitivity analysis study of the effects of hardware and virtual machine failures on system throughput.

This work is structured as follows. Section 2 covers the theoretical background and Section 3 presents a list of related works. Section 4 describes the proposed optimization method and Section 5 presents the performability model used for the objective function. In Section 6, we show the evaluation results for the proposed method. Finally, Section 7 makes some final considerations and points further research directions.

2 Background

This section presents basic concepts about combinatorial optimization with simulation and workflow scheduling, aiming to facilitate the understanding of this work.

2.1 Simulation/Optimization Hybrid Heuristics

Combinatorial optimization problems (COP) arise in many areas of human activity and computer science as well. A single-objective COP can be described regarding [6]:

- a set of variables $X = \{x_1, x_2, \ldots, x_n\}$, where x_i belongs to a domain D_i;
- a set of constraints $c_1, c_2, \ldots, c_m$;
- an objective function $f : D_1 \times D_2 \times \ldots D_n \mapsto \mathbb{R}$.

The optimization procedure should give a solution s from the space state $S = D_1 \times D_2 \times \ldots D_n$ that either minimizes or maximizes the objective function $f(s)$, and satisfies all constraints.

The characteristics of the objective function and constraints define the approach used to solve the problem. If the objective function and constraints have linear relationships, the underlying problem can be solved by the efficient Simplex algorithm [39]. However, since many essential COP problems are NP-Hard, a conventional approach to solving them is using approximate algorithms [4]. They can find good enough - but not optimal - solutions from a potentially large solution space. *Metaheuristics* are approximate algorithms not tied to any particular problem/domain and can be adapted for solving many different combinatorial problems [53].

COP problems with stochastic components in either the objective function or constraints can adopt simulation models to represent the random behavior. Hybrid *Simulation-Optimization* (Sim-Opt) methods deal with the issues involved in using simulation models in conjunction with optimization algorithms. Swisher et al. [51] define Sim-Opt methods as "*a structured approach to determine optimal settings for input parameters (i.e., the best system design), where optimality is measured by a (steady-state or transient) function of output variables associated with a simulation model*".

A simulation-optimization problem can be viewed as the selection of the best design among a (potentially large) set of possible designs concerning some output response variable is given by a simulator model. A random distribution will define the response variable for each element of this set. The optimization procedure selects the design that corresponds to the highest or lowest expected value of this distribution by using a sample for each design. Swisher et al. [51] provide a survey of different Sim-Opt approaches and present guidelines to help the selection of the most suited approach given the particular characteristics of the problem (e.g., all designs have the same variance, the variance is known/unknown, and so on).

Using simulation instead deterministic models becomes possible to avoid simplifications needed when using deterministic models. The expressiveness of deterministic models, however, has a drawback: the time to solve a simulation model can be significantly prolonged. This fact is even more problematic in simulations used in conjunction with meta-heuristic algorithms that need to compute the objective function of a large number of solutions. One alternative is to use a *surrogate model* [44], a simplification of a more complex simulation model, that can be evaluated more quickly. A typical approach is to use an Artificial Neural Network (ANN) as a surrogate model [21] along with a simulation model to train the network. The training phase is computationally intensive, but once completed, it generates results very quickly.

2.2 Scheduling of Scientific Workflows on Cloud Systems

A scientific workflow consists of a set of computing and IO intensive tasks with precedence constraints between them. Scientific workflows can be represented by a directed acyclic graph (DAG). A DAG G is defined by a tuple $\{T, E\}$, where $T = \{T_1, T_2, \ldots, T_n\}$ is the set of tasks and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of precedence constraints. Each $e_i = (T_a, T_b)$ tuple denotes that task T_b starts to execute after T_a finishes and sent some input data to T_b. The node weights are the computing times, and the edge weights are the communication times, i.e., the time for sending the results needed by a dependent task running on a different processing node.

A scheduler should map tasks/jobs to a set of processors according to some predefined goals such as utilization of resources, makespan (the total length of a schedule), throughput, meeting of deadlines, etc. A particular scheduling of tasks for some DAG can be defined by a mapping of ordered task lists to the processing units. A scheduling algorithm can either require as input the number of processing units or try to find an optimal/near optimal number of processing units in conjunction with the mapping of tasks. The latter case is harder since it leads to an increased

search space. Additionally, increasing the number of processors can shorten the makespan of an individual job, but it may increase the idle time of processors due to the precedence constraints between tasks [48].

Since many practical scheduling problems are either NP-Hard or NP-Complete, a lot of effort is necessary to apply and adapt meta-heuristic algorithms to schedule tasks in cloud computing environments. The following list summarizes the major contributions made by the literature concerning different aspects of cloud workflow scheduling:

- Devising heuristics algorithms able to provide near-optimal solutions under certain constraints [32];
- Adapting nature inspired and evolutionary algorithms for this problem, such as Genetic algorithm [22, 60, 64], Honey bee colony [5, 31], ant colony [15, 52], and Fish Swarm [61];
- Dealing with a heterogeneous system (processors with different computing power) [41];
- Dealing with conflicting aspects of a multi-cost objective function (e.g., energy versus makespan) [37].

3 Related Work

3.1 Performability Modeling of Cloud and Grid Environments

Performability is the study of systems performance when subjected to the effect of failures on its subcomponents [36]. The performance of a system is said to be *degradable* if failure events may affect it negatively. For instance, a mesh network of routers can tolerate a certain number of failures, but the overall performance will be affected as some routers may be subject to overheads. Similarly, failures on worker nodes in cloud and grid environments can diminish the number of available processing resources, and therefore increasing queueing times and decreasing throughput of jobs.

Due to a large number of components of cloud and grid environments, we can expect a significant failure rate even if the mean time to failure of individual components is high. Thus, neglecting the impact of failures in performance studies of such systems can lead to misleading results. Space stated based models (Stochastic Petri Nets and Markov Chains) is the most adopted method for joint performance/availability evaluation of clouds and grids systems. Dealing with space state explosion is a recurrent problem handled by every work in this category.

Ramakrishnan and Reed [46] propose a qualitative framework for the performability evaluation of scientific workflows running on grid systems. The framework encompasses a Markov Reward Chain model [7] and simulations calibrated with data from real grid applications. Xia et al. [58] describe a queuing network model for evaluating estimated service time and request rejection probability of an Infrastructure-as-a-Service cloud. This model represents features such as request handling, job creation, job execution, job rejection due to insufficient queue capacity, and failure and repair events of physical machines configured in hot/warm standby mode. Ever et al. [18] propose a set of equations obtained from queuing theory for evaluating the performability of clouds with large numbers of servers. Since the underlying space-state model does not need to be generated, this model can represent a large number of servers and simultaneous requests.

A strategy to avoid space state explosion is to adopt small models rather than use a big monolithic one. For combining the results of the submodels, iteration methods can be used [34]. Ghosh [19] uses interacting homogeneous time Markov chains to perform end-to-end performability analysis of cloud services. The proposed model is used to evaluate two essential metrics: service availability and response time. Raei et al. [45] developed Stochastic Reward Net models for representing a public cloud and a cloudlet providing virtual machines for mobile applications. For avoiding space state explosion when modeling both performance and availability aspects of the considered system, the authors divided the public cloud and cloudlet parts into two separated models and used the fixed point iteration method to obtain a joint result.

The performability models cited in this subsection are based on Markov Chain [7, 46], Stochastic Petri nets (with Markov chain generation) [19, 34, 45], and Queuing Theory [18, 58]. By contrast, our work adopts a discrete simulation approach based on Stochastic Petri nets components and automatic generation of models. The advantage of our model over the works mentioned above is the ability to model DAGs as job requests and the relationship between VM and hardware failures. Incorporating these features into

space-state based models would lead to a space-state explosion problem. Also, using the exponential distribution for representing job times can introduce distortions when modeling the makespan of a stochastic DAG (as we demonstrate in Section 6.2.2).

3.2 Simulation of Workflow Execution on Cloud Environments

Simulation is a commonly used approach for evaluating the performance of load-balancing algorithms, allocation policies, and scheduling strategies in cloud systems, considering dynamic workload patterns. CloudSim [13] is the most adopted cloud simulation software in the literature. The CloudSim simulator allows the representation of data-center infrastructures, VM allocation policies, user level workloads, and coordination between multiple cloud environments through a cloud broker service. After being released as open source software, CloudSim was extended in many different ways by the research community. Fault tolerance capabilities were introduced in [65]. FederatedCloudSim [30] extended CloudSim to represent SLA policies in federated clouds. FailureSim [16] introduced failure prediction of cloud nodes based on ANNs. Performance and usage levels (bandwidth, number of tasks running, the quantity of available million of instructions per second per node) are used as predictors for training the network. Alwabel et al. proposed DesktopCloudSim [1], a CloudSim extension with a layer of failure injection for the physical nodes. Like our work, DesktopCloudSim allows the investigation of failure events on system throughput.

CloudSim does not offer, by default, classes for representing workflows modeled as DAGs. Given the importance of this application category, some extensions to CloudSim were proposed for representing workflows. WorkflowSim [14] is a CloudSim extension that includes support for workflow representation and management. It also provides task aggregation capabilities and a fault generator at job/task level. It can generate recoverable transient failures that can be handled by task re-execution and permanent job failures that cannot be recovered. DynamicCloudSim [10] is a CloudSim based simulator that includes workflow execution considering VM inhomogeneity and failure of tasks at runtime. Malawskia et al. [35] developed a cloud workflow simulator for evaluating task scheduling and resource provisioning algorithms for optimizing the execution of workflow ensembles under deadline constraints in IaaS clouds. ElasticCloudSim [11] is a CloudSim extension for evaluating workflow applications which supports auto-scaling capabilities and considers non-deterministic (stochastic) workflows.

Our work differs from WorkflowSim and DynamicCloudSim by modeling hardware and virtual machine failures instead of representing transient/permanent failure on tasks. FailureSim is able to model hardware failures, and DesktopCloudSim can represent both hardware and VM failures. However, DesktopCloudSim and FailureSim do not target workflow applications. We opted not to create another CloudSim extension as the employed SPN based simulator presents some advantages. The proposed SPN models can be used separately for obtaining other metrics than performability (e.g., availability, reliability, and expected makespan). Using this simulation environment also allows us to use existing SPN models in the literature for representing reliability and performance aspects of our system.

3.3 Cloud Workflow Optimization

The execution of workflow applications on clouds brings the need of new modeling strategies, scheduling algorithms, and optimization metrics. The reason for this need is the particular aspects of cloud systems when contrasted to traditional grid/cloud environments. Kliazovich et al. [29] demonstrated how existent workflow models fail to address the communication patterns typically found in cloud workflow applications. They proposed CA-DAG (Communication-Aware DAG), a workflow model which represents communication processes as vertices instead of edges. Arabnejad and Barbosa [3] developed a Heterogeneous Budget Constrained Scheduling (HBCS) for minimizing makespan and rental cost of cloud workflow applications. The HBCS algorithm is able to reduce up to 30% of the execution time while maintaining the same budget level.

Many works in the scheduling literature consider deterministic computation and communication times. However, using a deterministic objective function does not match the non-deterministic nature of real-world applications [2]. In this sense, Zheng et al. [62] proposed a Monte Carlo based scheduling

method for cloud/grid workflows which consider non-deterministic computing and communication time. The method is not dependent on a particular heuristic algorithm, and the HEFT is adopted in the evaluation. In [63], a randomized version of HEFT was proposed. The algorithm consists in running a deterministic HEFT for random predictions of the stochastic DAG, generating a list of potential candidates for best scheduling. The scheduling from the list with the smaller expected makespan is selected. Cai et al. [12] presents a dynamic algorithm for minimizing the rental cost (of VMs in a cloud) of bag-of-tasks workflows with non-deterministic times. The Cloud Workflow Scheduling Algorithm (CWSA) [47] aims to optimize the scheduling of workflows in a multi-tenant cloud environment. This algorithm considers non-deterministic times for task computation times.

The presence of failures in data centers can pose a threat to workflow applications with strict deadlines. In [56], an original fault-tolerant scheduling algorithm name FESTAL was proposed. It employs a primary/backup redundancy model and VM migration to achieve high-availability and load balancing into a cloud workflow application. Vinay et al. [54] present a new heuristic for cloud scheduling named CHEFT (Cluster-based Heterogeneous Earliest Finish Time). It uses the idle time of the processors for resubmitting the failed tasks as a mean to achieve fault tolerance. FASTER [66] is another algorithm that employs the primary/backup redundancy model for providing a fault tolerant scheduling mechanism for cloud applications. Performability was first considered an objective function in [17]. The authors developed a performability model of a grid resource based on a stochastic reward net model and the universal generating function. The proposed model is connected to a genetic algorithm which aims to optimize the scheduling of a DAG into a set of grid resources.

Our work aims to contribute to the research line opened by Entezari et al. [17] - workflow scheduling optimization from a performability viewpoint. To the best of our knowledge, no existing method covers simultaneously performability as the objective function, multi-tenancy, non-deterministic computing/communication times, and failures of hosts and VMs. Table 1 shows a comparison of our work to the state of the art.

4 Problem Definition and Proposed Optimization Method

This work aims to solve the multiprocessor scheduling problem of scientific workflows running in a cloud environment, using a performability metric as the objective function. Given a workflow described by a DAG $G = \{T, E\}$, our objective is to find a scheduling S of tasks in T on m virtual machines that maximizes the throughput of jobs. The number m is not fixed and must be determined by the optimization method.

The flow diagram of Fig. 1 presents a high-level overview of the proposed optimization method. The workflow DAG has a list of tasks and their dependencies, processing time of each task and communication time between dependent tasks running on

Table 1 Comparison of the state of art for cloud workflow scheduling

Work	Metric	Multi-tenancy	Non-deterministic times	Failure model
Kliazovich et al. [29]	Makespan	No	No	Not considered
Arabnejad and Barbosa [3]	Makespan and rental cost	No	No	Not considered
Zheng and Sakellariou [62]	Makespan	No	Yes	Not considered
Cai et al. [12]	Rental cost	Yes	Yes	Not considered
Rimal and Maier [47]	Makespan and resource usage	Yes	Yes	Not considered
Zhou et al. [56]	Job reliability and resource usage	Yes	No	Host failures
Zhu et al. [66]	Job reliability and resource usage	Yes	No	Host failures
Vinay and Kumar [54]	Makespan and cost	No	No	Task failures
Entezari-Maleki et al. [17]	Performability	Yes	Yes (exponential times)	Processor failures
Our work	Performability	Yes	Yes	Host, VM, and cloud manager failures

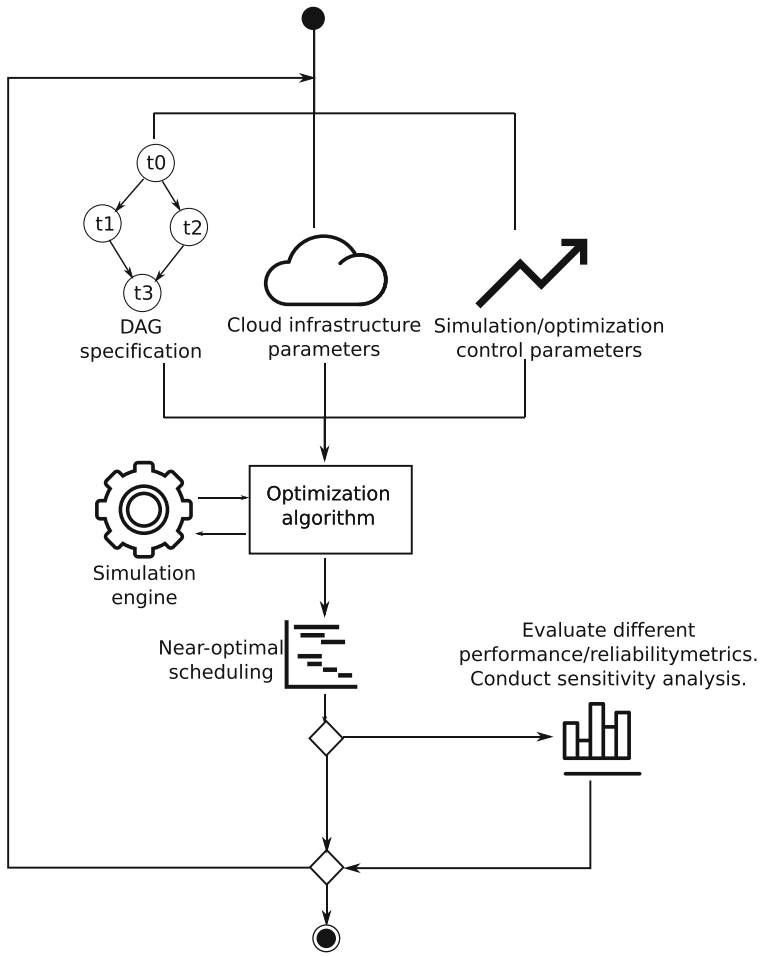


Fig. 1 Overview of the proposed method

different processors. The computing and communication times of a DAG can either be deterministic or follow a specific random distribution (normal, exponential, Erlang, and so on). The cloud infrastructure parameters define the number of physical servers, the maximum number of virtual machines that each host can provide, and the failure/repair/switchover rates of the physical/virtual machines. The simulation/optimization parameters configure the simulation engine (e.g., number of replications for an individual simulation) and the optimization algorithm (e.g., population size, number of elite chromosomes, number of generations and so on).

The optimization algorithm explores the solution space for the input DAG until the stopping condition is reached, which is defined by the control parameters. Then, a near-optimal scheduling solution is provided by the algorithm. The user can perform further analysis of the obtained solution and evaluate additional performance/reliability metrics, such as average waiting/response time, discard rate of tasks/jobs, the probability of completing a job. The method can be used interactively, i.e., the user can modify the cloud/control parameters and repeat the process, obtaining new scheduling and performability metrics.

The remainder of this section will explain further each part that composes the proposed method.

4.1 Genetic Algorithm with Stochastic Fitness Function

The activity diagram of Fig. 2 describes the optimization algorithm adopted in this paper. It is a genetic optimization procedure that uses a simulation model for computing the fitness value of explored chromosomes. The chromosome representation consists of a pair of vectors representing the ordering of tasks and

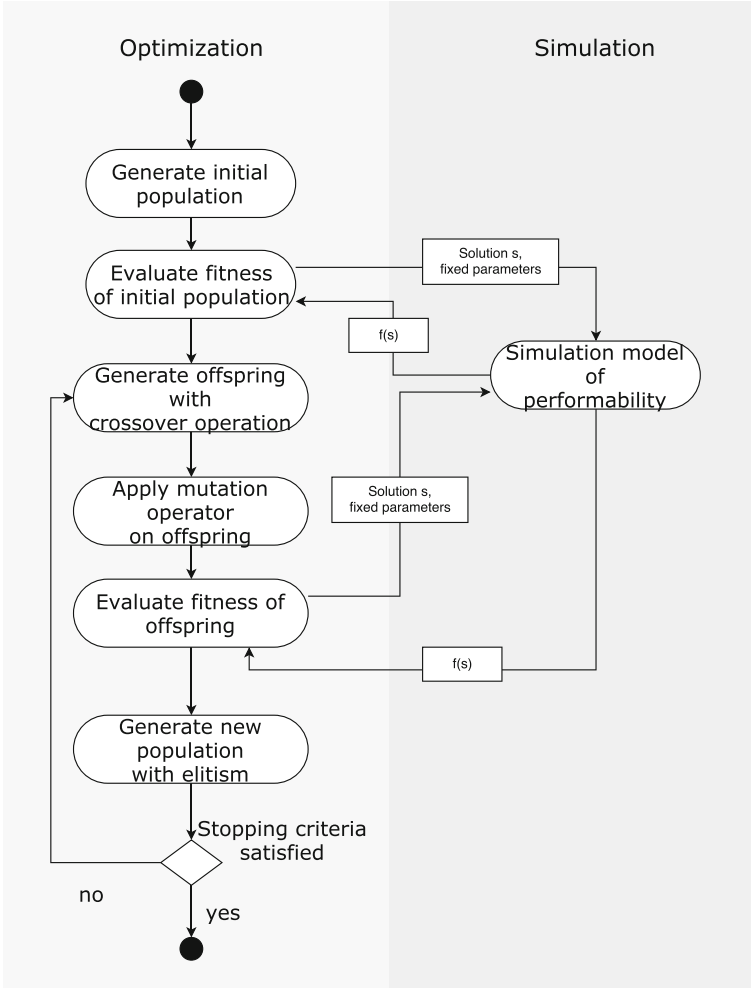


Fig. 2 Genetic algorithm with a stochastic fitness function

the mapping of tasks to the processors, as illustrated in Fig. 3. The partially-mapped crossover (PMX) operator [20] was adopted to generate the offspring of a population. Two random chromosomes are randomly selected for creating a pair of children (parents with higher fitness value are more likely to be selected). The process to generate the children is defined as follow. First, it creates a copy of the parents. A paired subinterval is randomly selected and switched among the children. Then, a mapping function is applied to convert the repeated alleles (i.e.: the units of information that compose the chromosome) outside the random subinterval. Figure 4 shows an example of crossover operator. The mutation operator modifies a chromosome with a random operation by swapping

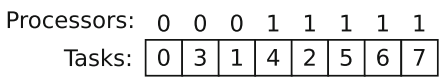


Fig. 3 Chromosome representation

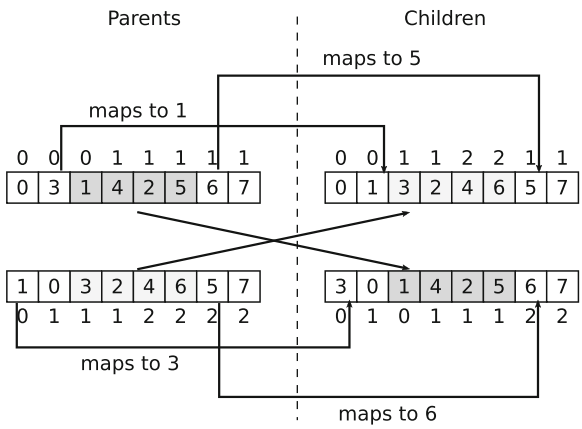


Fig. 4 PMX crossover operator

the order of two tasks or changing a task to a different processor. The mutation operator is illustrated in Fig. 5.

Figure 6 shows the activity diagram of the method to obtain a fitness value of a solution s. The method takes as input a solution s from the search space of available schedulings, a set of fixed parameters, a template model, and a set of auxiliary models. The solution s is parsed to obtain a set of solution parameters. The fixed and solution parameters are combined into a single set. The joint solution-fixed parameters set is divided into two categories: structural and nonstructural. The structural parameters define the fixed structures of the model and the arcs/edges that connect them. Nonstructural parameters define the delay/rate parameters and information such as probabilities, and buffer capacity.

The combined set of structural/non structural parameters and the base models are used as input for generating the final simulation model for a solution. The next section explains the performability model and the conversion algorithm.

5 Performability Model for the Fitness Function

In this section, we describe the simulation model used to compute the objective function of the scheduling engine available in the Mercury modeling tool [33, 40]. This simulation engine is based on the Stochastic Petri nets formalism and allows working with models created on other formalisms such as Continuous Time Markov Chains and Reliability Block Diagrams.

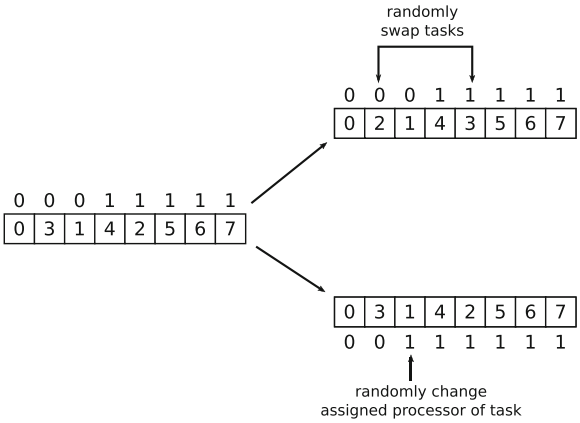


Fig. 5 Mutation operator

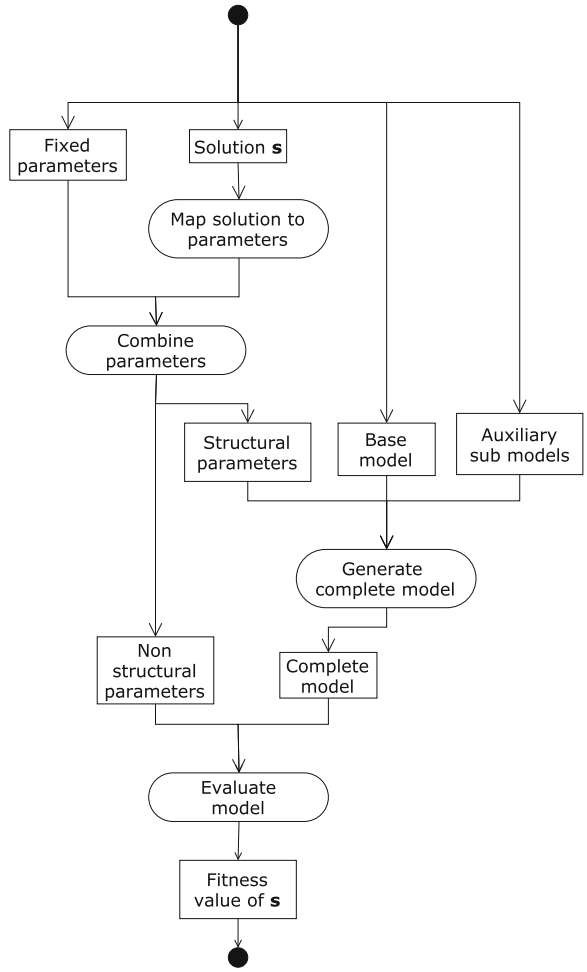


Fig. 6 Method to obtain the fitness value for a solution s

It enables us to use existing reliability models for private clouds [49] due to its hierarchical modeling capabilities.

Figure 7 provides an overview of the modeled system. The cloud infrastructure consists of a collection of hosts that acts either as workers or part of a cloud manager. A work node is a server that runs the users' virtual machines. The cloud manager is a subsystem composed of the software components for handling users' requests and orchestrating the cloud infrastructure. Employing more than one physical machine to run the necessary services is recommended to avoid a single point of failure.

Job requests arrive according to a Poisson process, and the rate is a controlled parameter. Each job is composed of a series of subtasks with precedence constraints defined as a DAG. A predefined strategy

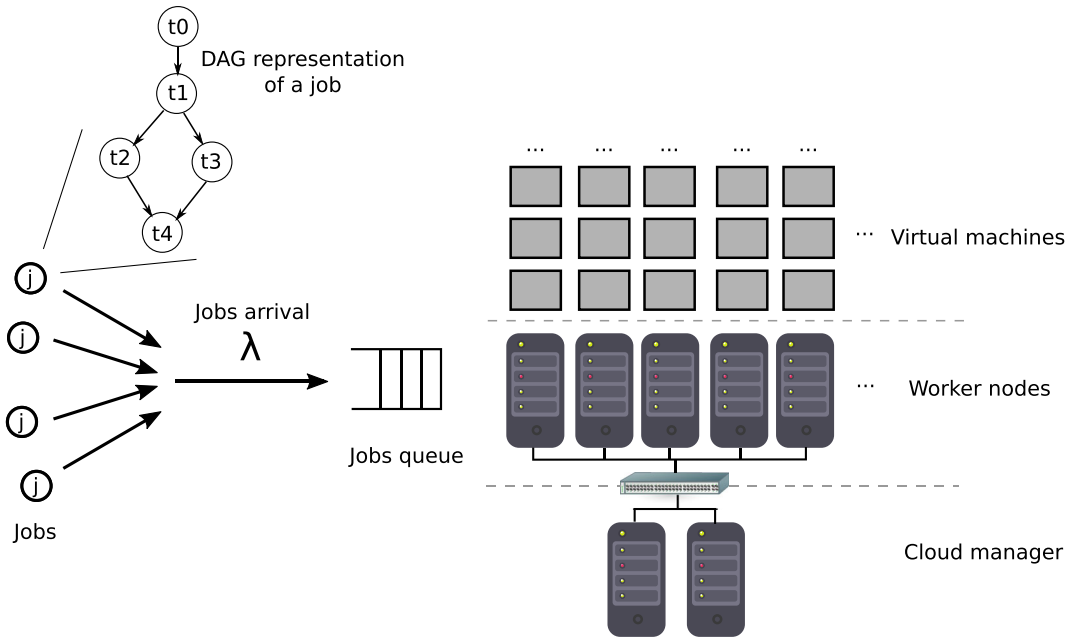


Fig. 7 Modeled system

establishes the scheduling of the subtasks. This strategy determines the number of virtual machines allocated from the cloud provider and the tasks executed on each of them. The scheduling can be generated by some heuristic (First Come-First Served, Shortest Job First, Heterogeneous Earliest Finish Time, etc.) or meta-heuristic algorithm (taboo search, genetic algorithm, ant colony, etc.).

Figure 8 depicts the discrete event simulation model as an open queue accepting a job submission influx with rate equals to λ. Each job will be promptly executed in case there are available resources in the cloud. Otherwise, it will be enqueued or discarded (if the queue is full). A job will also be discarded if the cloud manager is unavailable since in this case, it is not possible to allocate the cloud resources for the job execution. Another possibility for the failure of a job submission is when some virtual machine failure prevents the execution of one or more subtasks. A virtual machine can fail due software (operating system or hypervisor) or hardware faults.

The number of completed and failed jobs are recorded by the simulation model. The annual throughput is obtained by dividing the simulation time in years by the total number of completed jobs. The job failure ratio is defined by (1). The annual throughput and the job failure ratio are the metrics adopted in the analysis of Section 6.

$$\text{Job failure ratio} = \frac{\text{Discarded jobs}}{\text{Completed jobs} + \text{Discarded jobs}} \quad (1)$$

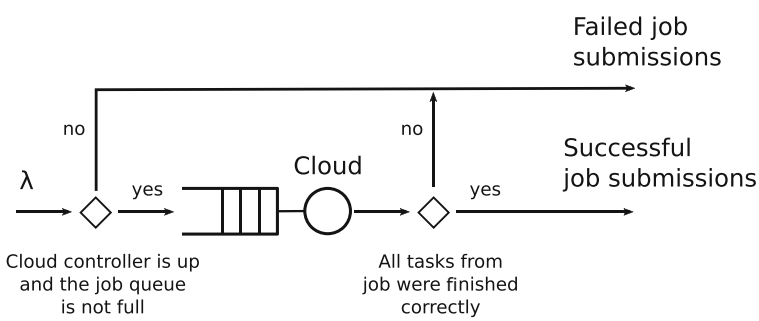


Fig. 8 Discrete event simulation model

In the simulation model, the service time of each job execution is computed by converting it to a Stochastic Petri net and measuring the time to reach a deadlock marking. The transitions from the generated SPN represent the following events: i) the processing of a task; ii) communication between tasks scheduled to different VMs; iii) and failure of VMs. The SPN will reach a deadlock marking after all tasks are executed or if a VM failure impedes any task to finish properly. In the next subsection, we explain the conversion algorithm.

5.1 Performance Modeling via DAG to SPN Conversion

Stochastic Petri nets are well suited for modeling parallel/concurrent activities and the logical ordering of events, such as resource contention, forking, joining, etc. In this work, we propose a conversion algorithm that takes a DAG workflow specification and scheduling as input and produces an SPN model as output. The resulting SPN model is used in the simulation model for representing the executions of the tasks that compose a job request. Besides being a fundamental part of our simulation-based performability model, the SPN representation enables us to evaluate the reliability, the estimated makespan, and the mean time to interruption of the scheduling of a DAG with non-deterministic times, via numerical analysis. For obtaining those metrics without employing simulation it is necessary that the processing and communication times can be depicted as exponential or phase-type random variables.

The Algorithm 1 is a pseudo-code representation of the proposed method. This algorithm takes a DAG and scheduling as input, and produces an SPN performance model as output. It means that a different SPN model will be generated for each particular scheduling. In the resulting model, each task is represented by a place-transition pair. The place represents the inputs for the task and the transition represents the task processing event. A processing transition for a task t must forward a token for each dependent task's inputs place. For a pair of tasks related by a precedence relationship, there are two possibilities:

– **The tasks are scheduled to the same VM.** In this case, the processing transition from the source task has an output arc connected to the inputs place from the destination task.

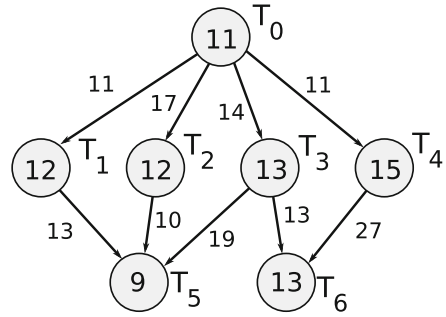


Fig. 9 Example of Directed Acyclic Graph (adapted from [59])

– **The tasks are scheduled to different VMs.** In this case it is necessary to include an additional place-transition pair for representing the communication. This transition is connected to the inputs place from the destination task via an output arc.

Besides the representation of data relationships from the DAG structure, the SPN model should express the temporal constraints imposed by the scheduling. If a set of tasks $t_1, t_2, \ldots, t_n$ are scheduled to the same VM (in this order), the processing of a task t_i should be enabled only after the execution of the previous task t_{i-1}, in addition to the DAG's data constraints. In our algorithm, this is ensured by connecting control places to the processing transitions via input arcs.

To illustrate our method, we used the DAG from Fig. 9 (adapted from [59]) and the scheduling displayed in Fig. 10 as the input for our algorithm. The resulting SPN model is displayed in Fig. 11. The control places and connected arcs are depicted in gray.

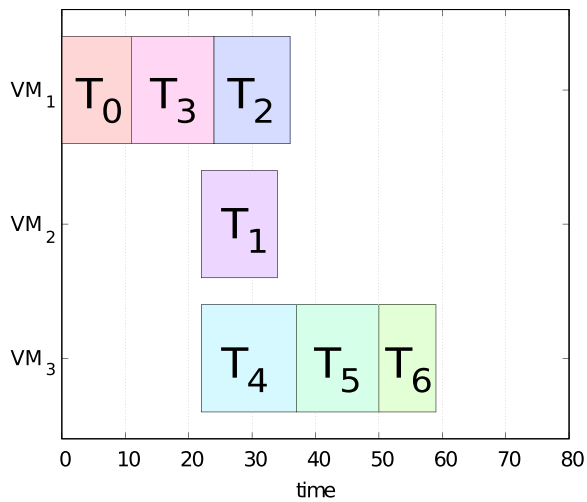


Fig. 10 A scheduling for the DAG shown in Fig. 9

Algorithm 1 DAG+scheduling to SPN conversion algorithm.

```
Data: DAG and scheduling specifications
Result: A Stochastic Petri net
for Each task t in the DAG do
    create a place pl for the dependencies of task t;
    create a transition trans for the processing of task t;
    connect pl and trans with an output arc;
    if t has no parent tasks then
        add one token to pl;
    end
end
for Each task t in the DAG do
    let pt be the processing transition for task t ;
    for Each task ct connected to t by an output arc do
        if t and ct are scheduled to the same VM then
            let ip be the inputs place for ct;
            let pct be the processing transition for task ct;
            connect ip and pct by an output arc;
        end
        else
            create a new place outp for the output from t to ct;
            let pct be the processing transition for task ct;
            create a new transition sendt for representing the communication between t and ct;
            connect t and outp by an output arc;
            connect outp and sendt by an input arc;
            connect sendt and pct by an outpyt arc;
        end
    end
end
for Each processing transition pt in the SPN do
    set the weight of the output arc of pt to be the number of input arcs;
end
for Each VM v do
    for Each task t scheduled to v do
        if t has predecessor then
            let t_pred be the predecessor task to t on the scheduling;
            connect the processing transition of t_pred and the input place of t by an input arc;
        end
    end
end
```

5.2 Reliability Modeling

Failure of virtual machines and physical servers have a substantial influence on system's performance. Job requests are enqueued if the cloud cannot provide the specified number of virtual machines. The decrease of the number of virtual machines due the effect of failures causes an increase in the average waiting time and a reduction on the system's throughput. When a job request completes, the allocated virtual machines are released and become available for processing further requests. If a failure occurs during the processing of a job, the failed virtual machines/worker nodes are subject to repair, and the processing is canceled. In case of a failure on the cloud manager, arriving job requests are discarded, since this is the component that acts as an interface between the users and the cloud infrastructure.

The reliability submodel describes the operational state (up/active or down/failed) of the cloud manager and the worker nodes and running virtual machines. It is represented as an SPN model integrated to the performability model. The cloud manager is defined as a fixed structure and the worker nodes and virtual machines are dynamically generated according to two parameters: the number of worker nodes and the maximum number of running virtual machines per server. Figure 12 illustrates the model generated for two worker nodes with two virtual machines per node. Each component is defined as a pair of places for defining the component's operational state, and a pair of transitions for causing failure/repair events. An immediate transition controlled by an inhibitor arc is used for updating the operational state of every virtual machine associated to a physical server.

We adopted the SPN availability model from [50] for representing the operational state of the cloud manager (shown in Fig. 13). This model represents two servers configured to work in a warm-standby redundancy scheme. The system availability is obtained by the following expression:

$$P\{\#primary\_up = 1 \text{ or } \#spare\_active\_up = 1\},$$

which represents the steady state probability of having a token in the $\#primary\_up$ or $\#spare\_active\_up$ places. In the warm-standby scheme, the spare server remains turned on but it does not accept workload if the main server is active. The failure rate of a

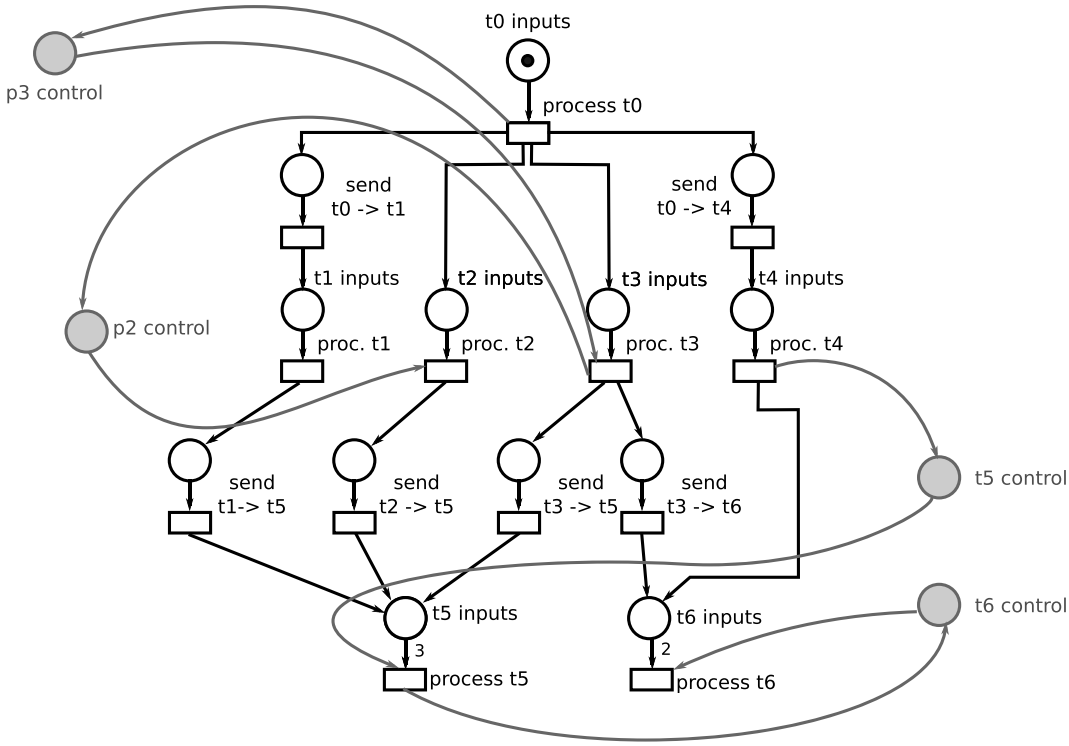


Fig. 11 DAG and scheduling converted to an SPN model

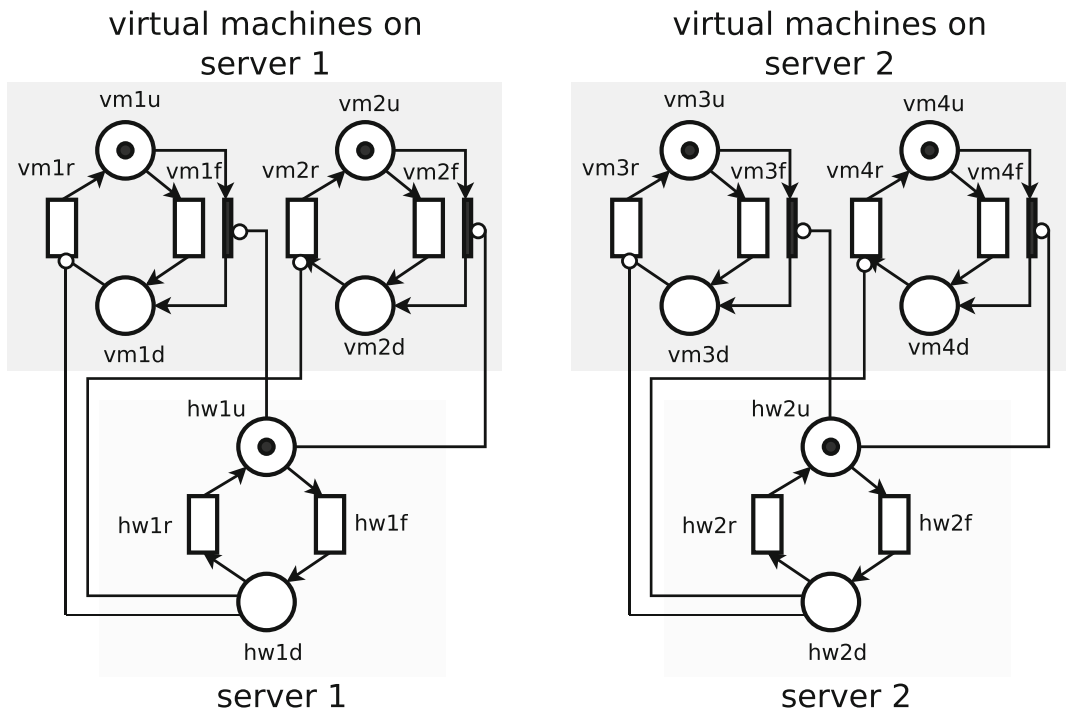


Fig. 12 Model generated with structural parameters set to $number\_of\_servers = 2$, $vms\_per\_server = 2$

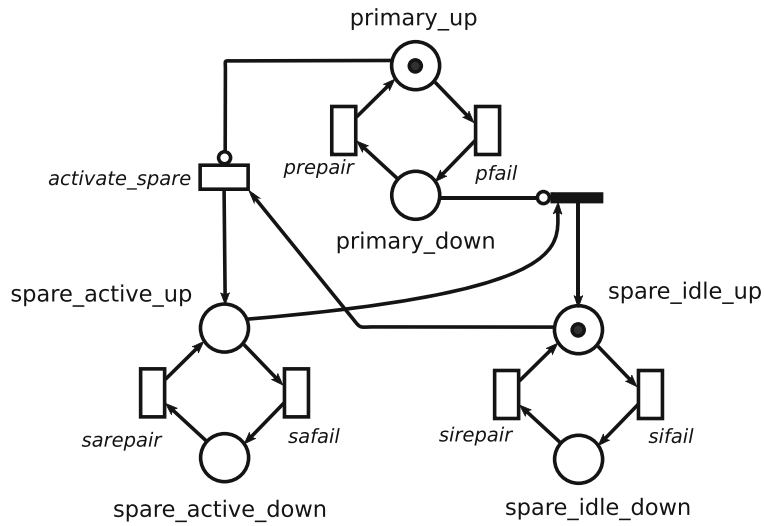


Fig. 13 Availability model for the cloud manager

idle component is assumed to be lower than the failure rate of a component serving requests [23]. The *activate_spare* transition represents the switchover event, i.e., the process of configuring the standby server to handle the incoming workload. This transition is enabled only when a failure occurs on the primary server due to the inhibitor arc. After the primary server is repaired, the standby server is sent back to the idle state.

5.3 Simulation Environment

In this subsection, we go into details about the top level simulation model and the simulation engine. Figure 14 displays an overview of top-level simulation model. As a discrete event simulator, it has a global clock for the simulation time and an ordered list of events which is processed and updated as the simulator runs. The simulation engine also maintains a list of running Petri nets. Petri nets can be configured at the beginning of the simulation, and new Petri nets can be created or destroyed during simulation run time. The simulation routines are software modules (implemented in the Java programming language) that are invoked according to specific simulation events. These routines can modify the simulation state, schedule/cancel simulation events, and start/destroy Petri nets. The Petri nets generate firing events and timed

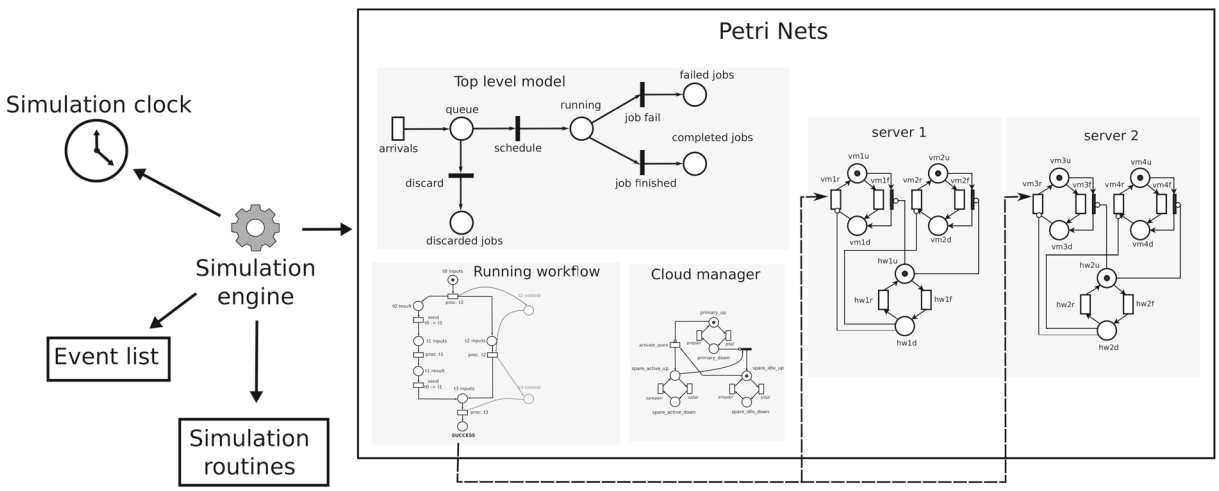


Fig. 14 Simulation model - high level view

transitions can be configured to fire according to a specific probability distribution.

The performability submodel keeps track of the workload (the workflows processed by the cloud resources). The transition *arrivals* create tokens corresponding to individual job requests. A job request being executed is associated to a *running workflow* Petri net (created by the Algorithm 1). An incoming workflow job starts its execution when resources are provisioned. Virtual machines are provisioned to workflow jobs via the VM scheduling algorithm (in this work, we used a round-robin algorithm). In Fig. 14, we display the association of the running job to the scheduled virtual machines with a dashed line arrow.

In this work, we employ transient simulations, i.e., we set a finish time and let the simulator run until this time is reached. For being able to capture failure events, a large simulation time should be employed. We adopted a runtime of three years, since using a higher simulation runtime was too computationally demanding. Metrics are obtained by reading the simulation state after a transient run. For obtaining point estimators and confidence intervals, we perform several transient runs with different random seeds.

6 Experimental Results

In this section, we present two case studies to evaluate the proposed method. The first case study shows the analysis of a small-sized workflow application/cloud infrastructure. In the second case study, we examine a real workflow application and a greater cloud infrastructure. In both case studies, the metric used for the fitness function was the average number of completed jobs in one year. We use a Fujitsu Primergy RX200 S7 server to conduct the experiments. The server has the configuration shown in Table 2.

Table 2 Server used for experiments - specifications

CPU 1, CPU 2	Intel Xeon E5-2650 (8 cores and 16 threads)
Memory	(10 banks of) DIMM DDR3 1600 MHz, 4GB
Operating system	Debian 7, Linux kernel version 4.9
JVM	Oracle JDK version 1.7

Table 3 Model parameters

Parameter	Value
Mean time to failure - physical machine	8760 h
Mean time to failure - idle physical machine	13140 h
Mean time to failure - virtual machine	2880 h
Mean time to repair - physical machine	1 h
Mean time to repair - virtual machine	1 h
Arrival rate of jobs	1/2.5 (1/h)
Number of workers	20
VMs per worker	3
Activation time of standby server	0.004 (h)
Number of replications for the simulation	30

6.1 Optimization and Performability Analysis of a Small Sized DAG Application

In this case study, we choose a small-sized DAG application for being able to apply a brute force analysis and compare the results found by the meta-heuristic algorithm with the global maximum. Table 3 shows the input parameters for the performability model. We adopted the mean time to failure, repair, and activation times for physical and virtual servers as defined in [28]. The DAG from Fig. 9 was adopted in this case study. We consider the computing and communication times to follow a normal distribution [12]. The mean values are obtained from the graph's nodes and edges and the standard deviation is assumed to be 10% of the mean value.

The scatter plot from Fig. 15 shows the fitnesses values (the number of completed jobs per year) for all possible schedulings of Fig. 9's DAG, evaluated by brute force. The solution space has 7840 different schedulings. Each solution is assigned to a unique integer identifier which is displayed on the plot's x-axis. It can be observed that the scheduling and the number of provisioned virtual machines play a critical role in the system's throughput. The difference between the worst and the best solutions is approximately 2430 jobs per year.

Due to the solution space not being too big, we adopted a small population of ten chromosomes for testing the genetic algorithm in this case study. We configured the algorithm to keep two elite chromosomes from the previous population in each iteration and employed a mutation probability of 0.05. Figure 16 shows the population's average and highest

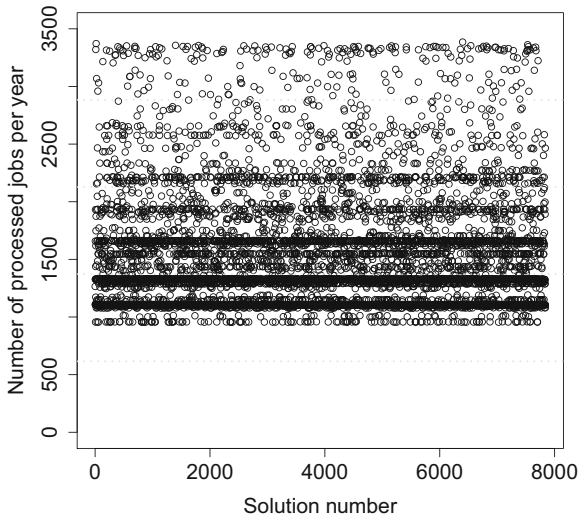


Fig. 15 Scatter plot of all solutions evaluated by brute force

fitness values for each generation. A horizontal line represents the fitness value for the global optimum at the top of the plot. It is possible to observe that the increase in the average fitness value after each generation leads to the discovery of a new elite chromosome after the fifth generation. In the tenth generation, the average fitness gets closer to the best fitness.

To analyze the impact of failures on the number of completed jobs per year, we made a sensitivity analysis on the mean time to failure for physical and virtual machines. Each parameter ranges from 20% to 200% of the base value shown in Table 3. Figure 17

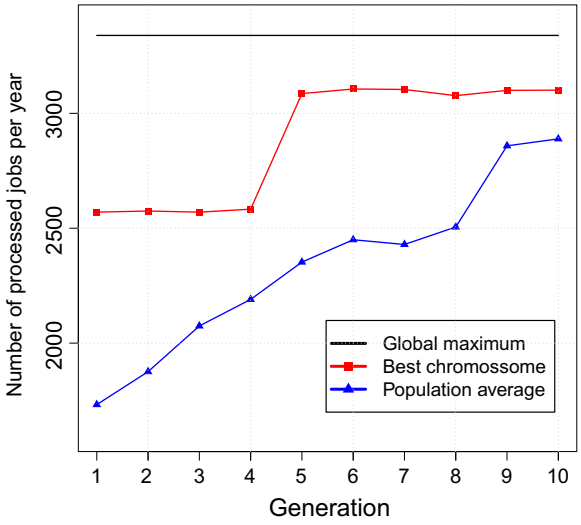
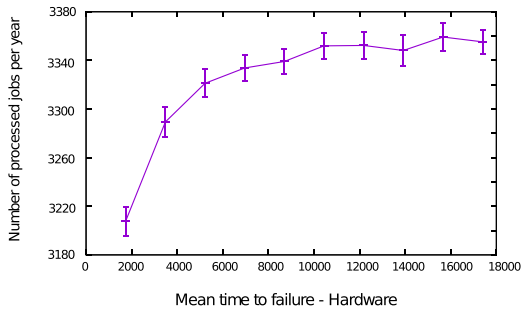
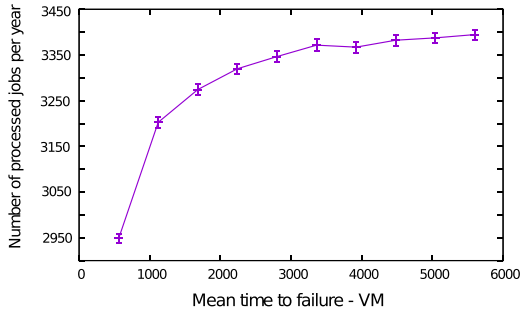


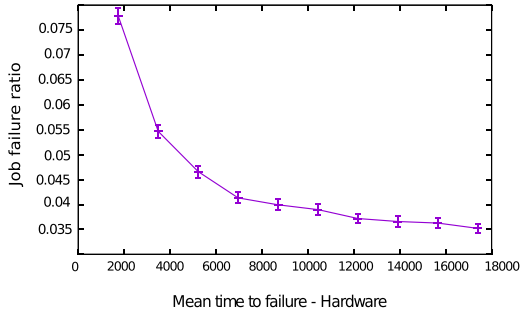
Fig. 16 Average and max fitness value (number of processed jobs per year) of each generation



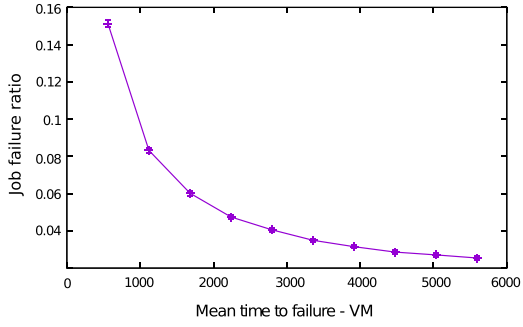
(a) MTTF Hardware × Number of processed jobs



(b) MTTF VM × Number of processed jobs



(c) MTTF Hardware × Job failure ratio



(d) MTTF VM × Job failure ratio

Fig. 17 Sensitivity analysis - one factor at time (95% confidence interval)

presents the results of the sensitivity analysis, considering the best scheduling found by brute force. For the collected metrics, we indicate the 95% confidence intervals alongside the average values. For each point on the plots, we obtained 500 samples from the simulation model. Figure 17a and b show the impact of the hardware and virtual machine MTTFs on the number of processed jobs per year and Fig. 17c and d show the sensitivity analysis of the failure ratio of jobs. The analysis reveals that the system throughput and job failure ratio are sensitive to VM failures. It also can be noticed that as the hardware/virtual machine MTTFs increase, the differences between adjacent points in the plots become less pronounced and the confidence intervals overlap.

6.2 Optimization of a LIGO Workflow Application

In the second case study, we used two scientific workflows: the LIGO Inspiral Analysis workflow [9] (Fig. 18a) and a randomly generated DAG (Fig. 18b). The LIGO workflow was created with the Pegasus Workflow Generator available in [26]. This workflow generator creates synthetic workflows based on traces collected from real-world scientific workflows. The random DAG was created by an ad-hoc algorithm. Computing and communication times for the random DAG were generated using a Uniform distribution with the interval from [1 min, 20 min] and [1min, 10 min], respectively.

Table 4 shows the updated parameters for the second case study. Unfortunately, increasing too much the cloud scale leads to a huge computational effort to solve the simulation model. The reason for this limitation is the presence of stiffness on performability models. For capturing failure events in the performance model, it is necessary to employ a long simulation run (larger than a year). The high number of events to be processed in a single simulation run leads to a great simulation runtime. SRIP (Single Replication in Parallel) techniques can be used to simulate a larger cloud infrastructure, i.e., a cloud having hundreds or thousands of physical servers.

Figure 19a and b summarize the results of the genetic algorithm. They are displayed as boxplots for each generation produced by the optimization algorithm. Since we are using elitism, the best solution (represented by the top horizontal bar in each box plot) is kept until better solutions are obtained. In contrast to the previous case study, we noticed a more accentuated non-monotonic growth in the average fitness value (i.e., the average fitness value for the ith generation being smaller than the value for the $(i-1)$th generation). However, the algorithm can increase both the average and maximum fitness value in the long term.

Table 4 Model/configuration parameters - second case study

Parameter	Value
Number of workers	50
VMs per worker	4
Number of replications for the simulation	10
Generations	25
Population size	40
Number of elite chromosomes	3

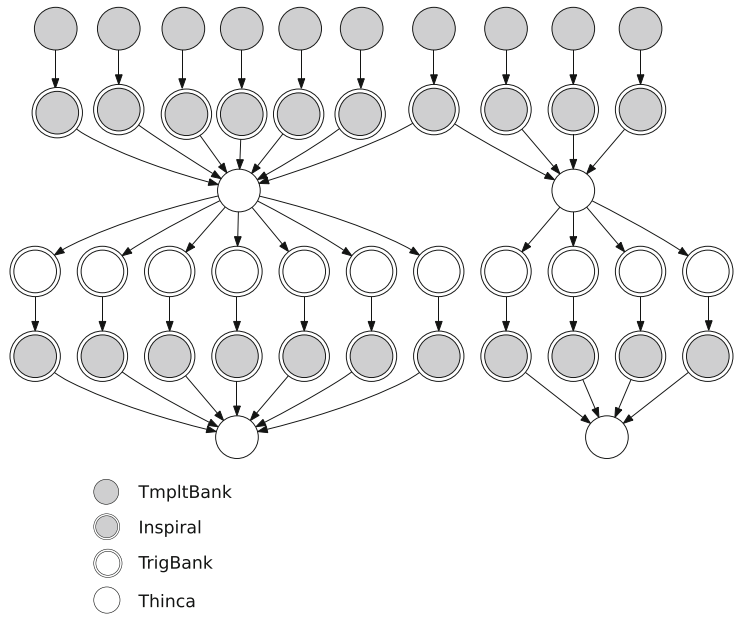
The presented case studies confirm the ability of the proposed simulation-based optimization method in solving the workflow scheduling problem from a performability viewpoint. Our method enables the optimization process to treat aspects that would be impossible to capture with a deterministic function, namely:

- Modeling non-deterministic and non-exponential computation/communication times;
- Capturing the failure relationships between servers and virtual machines;
- Modeling the provisioning of cloud resources to multiple users concurrently;
- Representing the influence of the cloud controller on the overall system's performance.

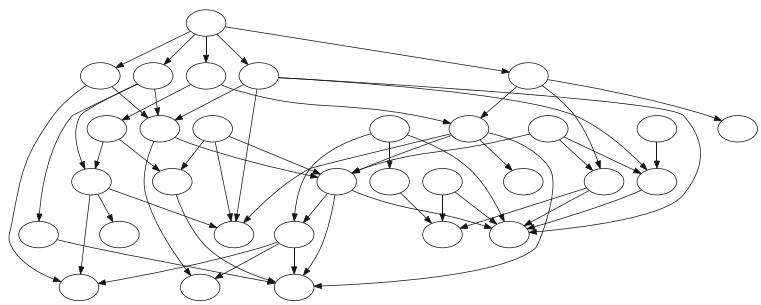
Using such complex non-deterministic objective function (a discrete simulation model) did not cause the optimization algorithm to misbehave. The results for the LIGO and random workflows show the effectiveness of the generic operators (mutation and crossover) in avoiding getting stuck at a local maximum. New elite chromosomes were found multiple times in both scenarios.

6.2.1 Performance and Reliability Analysis

For evaluating the impact of failures on the second case study, we performed a sensitivity analysis on the effect of hardware and VM failures in the adopted



(a) LIGO workflow



(b) Randomly generated workflow

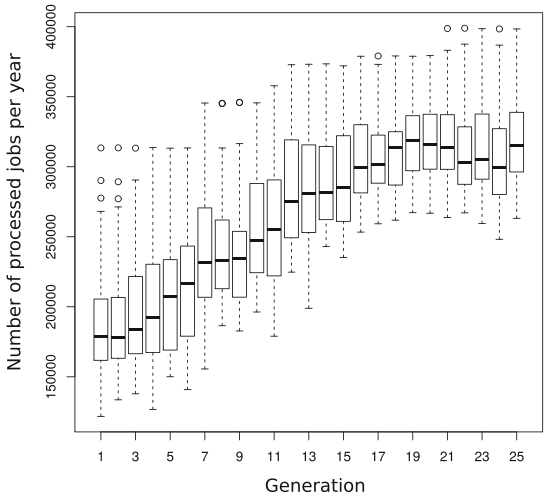
Fig. 18 DAGs for second case study

workflows. In this study, we consider the best scheduling obtained with the optimization method. The results are shown in Fig. 20. We confirm the same pattern visualized in the previous section for the small DAG: the impact of hardware and VM failures diminishes as the reliability of these components reach a certain level.

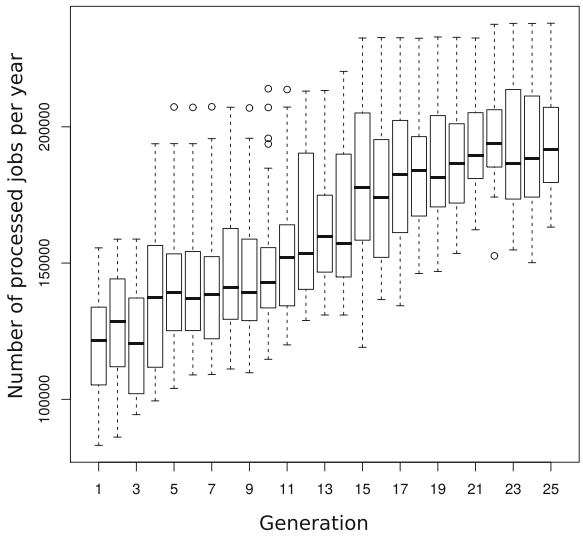
Figure 21 shows the impact of failures on the cloud manager in the system throughput. It also allows us to evaluate the effectiveness of the warm-standby redundancy mechanism when contrasted with a single node cloud manager (without redundancy). Figure 21 indicates that for a small MTTF for a server node, there is a substantial increase in the number of processed jobs per year when using a redundant cloud manager. For a large MTTF, however, the difference between the mean number of processed jobs is minimal, and the confidence intervals overlap. These results indicate a negligible impact of the cloud-manager on system throughput when this subsystem is highly-available.

6.2.2 Random Makespan Kernel Density Estimation (LIGO workflow)

Considering non-deterministic communication and computation times for a workflow scheduling algorithm means that the makespan will be defined by a random variable instead of being a fixed value. We performed a kernel density estimation for the random makespan of the best scheduling found for the LIGO



(a) LIGO workflow

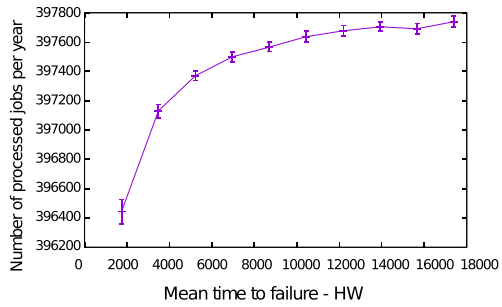


(b) Randomly generated DAG workflow

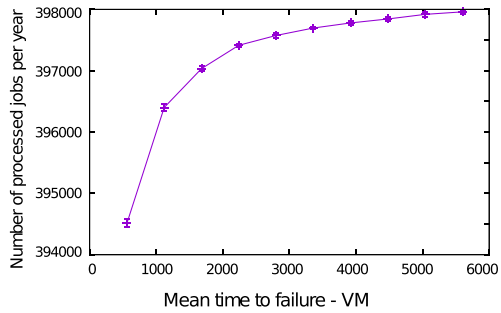
Fig. 19 Fitness values for each generation

workflow. Figure 22 shows kernel density plot for the following scenarios: i) normally distributed times with standard deviation being equals to 10, 20, and 50% of the mean, respectively; ii) exponentially distributed times; and iii) deterministic times. The expected value for all distributions is equal to the value considered in the deterministic scenario.

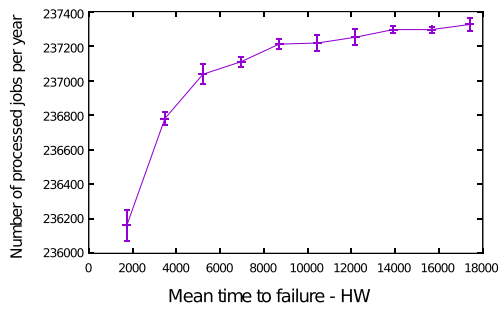
It can be observed that increasing the variance of individual communication/computation times in the DAG increases the expected makespan. The exponential distribution presents a high dispersion



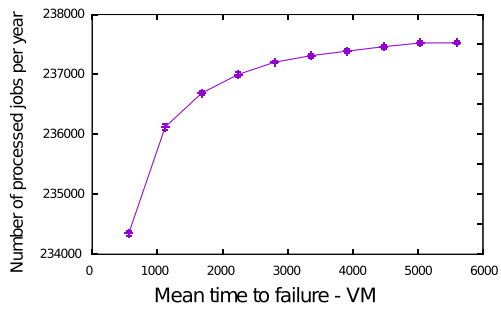
(a) MTTF Hardware × Number of processed jobs - LIGO workflow



(b) MTTF VM × Number of processed jobs - LIGO workflow

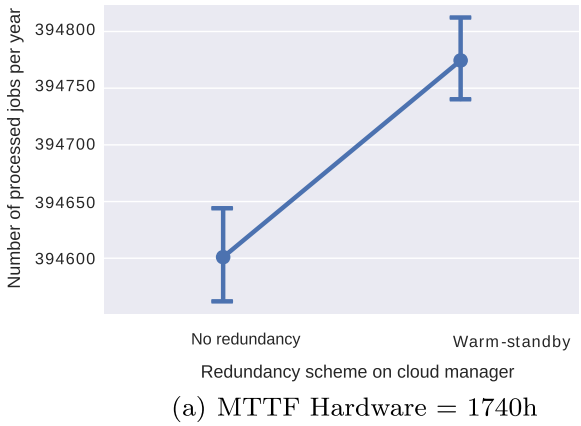


(c) MTTF Hardware × Number of processed jobs - random DAG workflow

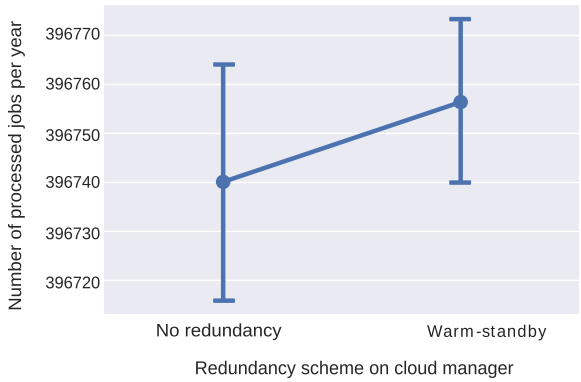


(d) MTTF VM × Number of processed jobs - random DAG workflow

Fig. 20 Sensitivity analysis - one factor at time (95% confidence interval)



(a) MTTF Hardware = 1740h



(b) MTTF Hardware = 8700h

Fig. 21 Influence of cloud manager failures on system throughput (95% confidence interval)

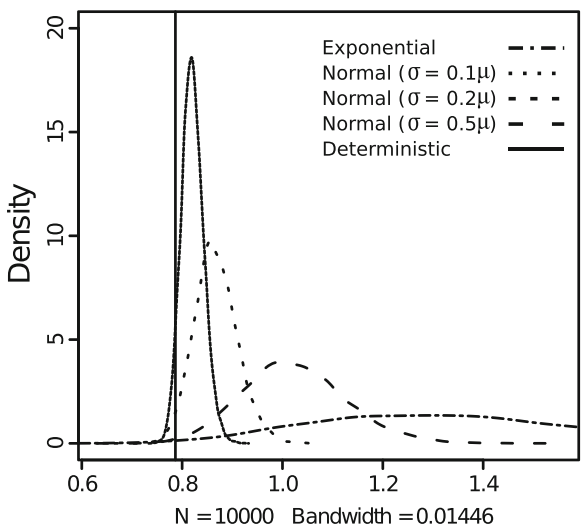


Fig. 22 Kernel density plot for Makespan of LIGO workflow (hours)

and an expected value distant from the deterministic makespan. We, therefore, conclude that using Markov-chain based methods for analyzing workflow applications may not be a good choice since they assume exponentially distributed times. Phase-type distributions can be used to approximate non-exponential times, but using them can substantially increase the number of modeled states and causing the space-station explosion problem.

7 Conclusions

This paper presented a simulation-based optimization method for scheduling of scientific workflows in cloud systems that considers system performability as the objective function. Our method employs the automatic generation of performability models for a cloud application that takes as input the workflow description (as a DAG) and the infrastructure configuration. Our evaluation shows that the genetic algorithm is efficient in optimizing both the number of virtual machines and the scheduling of the tasks concerning the system's throughput.

In future works, we intend to deal with heterogeneity by considering virtual machines with different computational capabilities. Furthermore, we are interested in implementing fault tolerance schemes such as checkpointing and replication in our simulation model. For evaluating more massive infrastructures/applications, we plan to implement SRIP parallelism in our simulation engine.

References

1. Alwabel, A., Walters, R., Wills, G.: Desktopcloudsim: Simulation of node failures in the cloud. In: International Conference on Cloud Computing, GRIDs, and Virtualization, p. 29 (2015)
2. Ando, E., Nakata, T., Yamashita, M.: Approximating the longest path length of a stochastic dag by a normal distribution in linear time. J. Discrete Algoritms **7**(4), 420–438 (2009)
3. Arabnejad, H., Barbosa, J.G.: A budget constrained scheduling algorithm for workflow applications. J. Grid Comput. **12**(4), 665–679 (2014)

4. Bianchi, L., Dorigo, M., Gambardella, L.M., Gutjahr, W.J.: A survey on metaheuristics for stochastic combinatorial optimization. Nat. Comput. **8**(2), 239–287 (2009)
5. Bitam, S.: Bees life algorithm for job scheduling in cloud computing. In: Proceedings of the Third International Conference on Communications and Information Technology, pp. 186–191 (2012)
6. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. (CSUR) **35**(3), 268–308 (2003)
7. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. Wiley, Hoboken (2006)
8. Book, R.V. et al.: Michael r. garey and david s. johnson, computers and intractability: a guide to the theory of *np*-completeness. Bulletin (New Series) of the American Mathematical Society **3**(2), 898–904 (1980)
9. Brown, D.A., Brady, P.R., Dietz, A., Cao, J., Johnson, B., McNabb, J.: A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis. In: Workflows for E-Science, pp. 39–59. Springer (2007)
10. Bux, M., Leser, U.: Dynamiccloudsim: Simulating heterogeneity in computational clouds. Futur. Gener. Comput. Syst. **46**, 85–99 (2015)
11. Cai, Z., Li, Q., Li, X.: Elasticsim: a toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times. J. Grid Comput. **15**(2), 257–272 (2017)
12. Cai, Z., Li, X., Ruiz, R., Li, Q.: A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. Futur. Gener. Comput. Syst. **71**, 57–72 (2017)
13. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
14. Chen, W., Deelman, E.: Workflowsim: a toolkit for simulating scientific workflows in distributed environments. In: 2012 IEEE 8th International Conference on E-Science (E-Science), pp. 1–8. IEEE (2012)
15. Chen, W.N., Zhang, J.: Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE Trans. Softw. Eng. **39**(1), 1–17 (2013)
16. Davis, N.A., Rezgui, A., Soliman, H., Manzanares, S., Coates, M.: Failuresim: a system for predicting hardware failures in cloud data centers using neural networks. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 544–551. IEEE (2017)
17. Entezari-Maleki, R., Trivedi, K.S., Sousa, L., Movaghar, A.: Performability-based workflow scheduling in grids. The Computer Journal (2018)
18. Ever, E.: Performability analysis of cloud computing centers with large numbers of servers. J. Supercomput. **73**(5), 2130–2156 (2017)
19. Ghosh, R., Trivedi, K.S., Naik, V.K., Kim, D.S.: End-To-End performability analysis for infrastructure-as-a-service cloud: an interacting stochastic models approach. In: 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 125–132. IEEE (2010)
20. Goldberg, D.E., Lingle, R., et al.: Alleles, loci, and the traveling salesman problem. In: Proceedings of an International Conference on Genetic Algorithms and their Applications, vol. 154, pp. 154–159. Lawrence Erlbaum, Hillsdale (1985)
21. Gorissen, D., Couckuyt, I., Demeester, P., Dhaene, T., Crombecq, K.: A surrogate modeling and adaptive sampling toolbox for computer based design. J. Mach. Learn. Res. **11**, 2051–2055 (2010)
22. Gu, J., Hu, J., Zhao, T., Sun, G.: A new resource scheduling strategy based on genetic algorithm in cloud computing environment. J. Comput. **7**(1), 42–52 (2012)
23. Guimarães, A.P., Maciel, P.R., Matias, R.: An analytical modeling framework to evaluate converged networks through business-oriented metrics. Reliab. Eng. Syst. Saf. **118**, 81–92 (2013)
24. Hamby, D.: A review of techniques for parameter sensitivity analysis of environmental models. Environ. Monit. Assess. **32**(2), 135–154 (1994)
25. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the use of cloud computing for scientific workflows. In: 2008. Escience'08. IEEE Fourth International Conference on Escience, pp. 640–645. IEEE (2008)
26. Juve, G., Bharathi, S.: Pegasus synthetic workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator (2014)
27. Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B.P., Maechling, P.: Scientific workflow applications on amazon Ec2. In: 2009 5th IEEE International Conference on E-Science Workshops, pp. 59–66. IEEE (2009)
28. Kim, D.S., Machida, F., Trivedi, K.S.: Availability modeling and analysis of a virtualized system. In: 2009. PRDC'09. 15th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 365–371. IEEE (2009)
29. Kliazovich, D., Pecero, J.E., Tchernykh, A., Bouvry, P., Khan, S.U., Zomaya, A.Y.: Ca-dag: Modeling communication-aware applications for scheduling in cloud computing. J. Grid Comput. **14**(1), 23–39 (2016)
30. Kohne, A., Spohr, M., Nagel, L., Spinczyk, O.: Federatedcloudsim: a sla-aware federated cloud simulation framework. In: Proceedings of the 2nd International Workshop on CrossCloud Systems, pp. 3. ACM (2014)
31. LD, D.B., Krishna, P.V.: Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl. Soft Comput. **13**(5), 2292–2303 (2013)
32. Lin, W., Wu, W., Wang, J.Z.: A heuristic task scheduling algorithm for heterogeneous virtual clusters. Sci. Program. 2016, Article ID 7040276 (2016)
33. Maciel, P., Matos, R., Silva, B., Figueiredo, J., Oliveira, D., Fé, I., Maciel, R., Dantas, J.: Mercury: performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In: 2017 IEEE 22Nd Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 50–57. IEEE (2017)
34. Mainkar, V., Trivedi, K.S.: Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. IEEE Trans. Softw. Eng. **22**(9), 640–653 (1996)

35. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. Futur. Gener. Comput. Syst. **48**, 1–18 (2015)
36. Meyer, J.F.: On evaluating the performability of degradable computing systems. IEEE Trans. Comput. **C-29**(8), 720–731 (1980)
37. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y.C., Talbi, E.G., Zomaya, A.Y., Tuyttens, D.: A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J. Parallel Distrib. Comput. **71**(11), 1497–1508 (2011)
38. Molloy, M.K.: Performance analysis using stochastic petri nets. IEEE Trans. Comput. **31**(9), 913–917 (1982)
39. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965)
40. Oliveira, D., Matos, R., Dantas, J., Ferreira, J., Silva, B., Callou, G., Maciel, P., Brinkmann, A.: Advanced stochastic petri net modeling with the mercury scripting language. In: ValueTools 2017, 11th EAI International Conference on Performance Evaluation Methodologies and Tools. Venice, Italy. Elsevier (2017)
41. Panda, S.K., Jana, P.K.: Efficient task scheduling algorithms for heterogeneous multi-cloud environment. J. Supercomput. **71**(4), 1505–1533 (2015)
42. Plateau, B., Atif, K.: Stochastic automata network of modeling parallel systems. IEEE Trans. Softw. Eng. **17**(10), 1093–1108 (1991)
43. Qiu, X., Sun, P., Guo, X., Xiang, Y.: Performability analysis of a cloud system. In: 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), pp. 1–6. IEEE (2015)
44. Queipo, N.V., Haftka, R.T., Shyy, W., Goel, T., Vaidyanathan, R., Tucker, P.K.: Surrogate-based analysis and optimization. Prog. Aerosp. Sci. **41**(1), 1–28 (2005)
45. Raei, H., Yazdani, N.: Performability analysis of cloudlet in mobile cloud computing. Inform. Sci. **388**, 99–117 (2017)
46. Ramakrishnan, L., Reed, D.A.: Performability modeling for scheduling and fault tolerance strategies for scientific workflows. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing, pp. 23–34. ACM (2008)
47. Rimal, B.P., Maier, M.: Workflow scheduling in multitenant cloud computing environments. IEEE Trans. Parallel Distrib. Syst. **28**(1), 290–304 (2017)
48. Rodriguez, M.A., Buyya, R.: A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. Concurr. Comput. Pract. Exp. **29**(8), e4041 (2017)
49. Sousa, E., Lins, F., Tavares, E., Cunha, P., Maciel, P.: A modeling approach for cloud infrastructure planning considering dependability and cost requirements. IEEE Trans. Syst. Man Cybern. Syst. Hum. **45**(4), 549–558 (2015)
50. Sousa, E., Lins, F., Tavares, E., Maciel, P.: Cloud infrastructure planning considering different redundancy mechanisms. Computing **99**(9), 841–864 (2017)
51. Swisher, J.R., Hyden, P.D., Jacobson, S.H., Schruben, L.W.: A Survey of simulation optimization techniques and procedures. In: Simulation Conference, 2000. Proceedings. Winter, vol. 1, pp. 119–128. IEEE (2000)
52. Tawfeek, M.A., El-Sisi, A., Keshk, A.E., Torkey, F.A.: Cloud task scheduling based on ant colony optimization. In: 2013 8th International Conference on Computer Engineering & Systems (ICCES), pp. 64–69. IEEE (2013)
53. Tsai, C.W., Rodrigues, J.J.: Metaheuristic scheduling for cloud: a survey. IEEE Syst. J. **8**(1), 279–291 (2014)
54. Vinay, K., Kumar, S.D.: Fault-tolerant scheduling for scientific workflows in cloud environments. In: 2017 IEEE 7th International Advance Computing Conference (IACC), pp. 150–155. IEEE (2017)
55. Vöckler, J.S., Juve, G., Deelman, E., Rynge, M., Berriman, B.: Experiences using cloud computing for a scientific workflow application, In: Proceedings of the 2nd International Workshop on Scientific Cloud Computing, pp. 15–24. ACM (2011)
56. Wang, J., Bao, W., Zhu, X., Yang, L.T., Xiang, Y.: Festal: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds. IEEE Trans. Comput. **64**(9), 2545–2558 (2015)
57. Wang, T., Chang, X., Liu, B.: Performability analysis for iaas cloud data center. In: 2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 91–94. IEEE (2016)
58. Xia, Y., Zhou, M., Luo, X., Zhu, Q., Li, J., Huang, Y.: Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds. IEEE Trans. Autom. Sci. Eng. **12**(1), 162–170 (2015)
59. Xu, Y., Li, K., He, L., Zhang, L., Li, K.: A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **26**(12), 3208–3222 (2015)
60. Zhao, C., Zhang, S., Liu, Q., Xie, J., Hu, J.: Independent tasks scheduling based on genetic algorithm in cloud computing. In: 2009. Wicom'09. 5th International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4. IEEE (2009)
61. Zhao, H.W., Tian, L.W.: Resource schedule algorithm based on artificial fish swarm in cloud computing environment. In: Applied Mechanics and Materials, vol. 635, pp. 1614–1617. Trans Tech Publ (2014)
62. Zheng, W., Sakellariou, R.: Stochastic dag scheduling using a monte carlo approach. J. Parallel Distrib. Comput. **73**(12), 1673–1689 (2013)
63. Zheng, W., Wang, C., Zhang, D.: A randomization approach for stochastic workflow scheduling in clouds. Sci. Program. 2016, Article ID 9136107 (2016)
64. Zheng, Z., Wang, R., Zhong, H., Zhang, X.: An approach for cloud resource scheduling based on parallel genetic algorithm. In: 2011 3rd International Conference on Computer Research and Development (ICCRD), vol. 2, pp. 444–447. IEEE (2011)
65. Zhou, A., Wang, S., Sun, Q., Zou, H., Yang, F.: Ftcloudsim: a simulation tool for cloud service reliability enhancement mechanisms. In: Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference, p. 2. ACM (2013)
66. Zhu, X., Wang, J., Guo, H., Zhu, D., Yang, L.T., Liu, L.: Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE Trans. Parallel Distrib. Syst. **27**(12), 3501–3517 (2016)