# Improving Cloud Simulation Using the Monte-Carlo Method

Luke Bertot(✉), Stéphane Genaud, and Julien Gossa

Icube-ICPS — UMR 7357, Université de Strasbourg, CNRS Pôle API,
300 Blvd S. Brant, 67400 Illkirch-Graffenstaden, France
{lbertot,genaud,gossa}@unistra.fr

**Abstract.** In the cloud computing model, cloud providers invoice clients for resource consumption. Hence, tools helping the client to budget the cost of running his application are of pre-eminent importance. However, the opaque and multi-tenant nature of clouds make task runtimes variable and hard to predict, and hamper the creation of reliable simulation tools. In this paper, we propose an improved simulation framework that takes into account this variability using the Monte-Carlo method.

We consider the execution of batch jobs on an actual platform, scheduled using typical heuristics based on the user estimates of task runtimes. We model the observed variability through simple random variables to use as inputs to the Monte-Carlo simulation. Based on this stochastic process, predictions are expressed as interval-based makespan and cost. We show that, our method can capture over 90% of the empirical observations of makespan while keeping the capture interval size below 5% of the average makespan.

## 1 Introduction

Over the last decade, the advancement of virtualization techniques has led to the emergence of new economic and exploitation approaches of computer resources in Infrastructure as a Service (IaaS), one form of cloud computing. In this model, all computing resources are made available on demand by third-party operators and paid based on usage. The ability to provision resources on demand is mainly used in two ways. First, it can serve for scaling purposes where new machines are brought online to face higher workloads and allows for a lower baseline cost. Second, it is useful for parallelizing tasks to achieve a shorter makespan (*i.e.* the time between the submission of the first task and the completion of the last task) at equal cost, this approach being often used for scientific and industrial workloads when runtime is heavily dependent on computing power. This approach is made possible by the pricing model of cloud infrastructures, as popularized by Amazon Web Services, in which payment for computing power provided as Virtual Machines (VMs), happens in increments of arbitrary lengths of time, billing time unit (BTU), usually of one hour. This model offers the client an almost complete freedom to start or stop new servers as long as it can be afforded. However, for distributed applications, it quickly becomes difficult to manually provision

© Springer International Publishing AG, part of Springer Nature 2018
M. Aldinucci et al. (Eds.): Euro-Par 2018, LNCS 11014, pp. 404–416, 2018.
https://doi.org/10.1007/978-3-319-96983-1_29

the resources in an efficient way. The use of a scheduler becomes unavoidable for such workloads. In this paper, we are interested in predicting the execution time and cost of such workloads, in which the scheduling plays an important role.

Independently of scheduling decisions, the accurate prediction of complex workload execution is hampered by the inherent variability of clouds, explained by multiple factors. Firstly IaaS operates in an opaque fashion: the exact nature of the underlying platforms is unknown, and their hardware are subject to evolution. Secondly cloud systems are multi-tenant by nature. This adds uncertainty due to contention on network and memory accesses. This variability, reported by a number of practitioners who evaluate parallel application performance on clouds (e.g. [13], who report an average 5%–6% variability on AWS cluster compute instances), has also been measured by one of the most comprehensive and recent surveys by Leitner and Cito [9]. We will see in this paper that our observations fit with the figures presented in this survey. This variability increases the difficulty of modeling task execution times. In this regard, the prediction is highly dependent on the underlying simulator of the system and on the phenomena it can capture. In our work, we rely on the SimGrid [4] simulation toolkit, enabling us to build discrete event simulators of distributed systems such as Grids, Clouds, or HPC systems. SimGrid has been chosen for its well-studied accuracy against reality (e.g. [18,20]). In particular, given a precise description of the hardware platform, its network model takes into account network contention in presence of multiple communication flows.

However, we may not be able to provide a fully accurate platform description, or be unable to estimate the network cross-traffic, yielding a distortion between simulation and reality. To deal with this problem, the standard approach is to consider task runtimes to be stochastic. Every task can be modeled by a random variable (RV) that models the whole spectrum of possible runtimes. These RVs are the basis required for a stochastic simulation. Such simulations output RVs of the observed phenomenon (*makespan* or $BTU$) which in turn can be used to create intervals of possible results with their assorted confidence. In this paper, we propose a stochastic method to enrich the classical prediction based on the discrete-event simulator SimGrid, and we study the conditions needed for this approach to be relevant. This study is carried out in a real setting, described in Sect. 3, where the applications, and the scheduler are presented. The stochastic framework we propose is then presented in Sect. 4 and is evaluated in Sect. 5. We discuss the limits of this approach in Sect. 6.

## 2 Related Work

*Simulation.* Most cloud simulators are based on discrete event simulation (DES). In discrete event simulation the simulation is a serie of events changing the state of the simulated system. For instance, events can be the start (or end) of computations or communications. The simulator will jump from one event to the next, updating the times of upcoming events to reflect the state change in the simulation. Such DES-based simulators require at least a platform specification and an

application description. The available cloud DESs can be divided in two categories. In the first category are the simulators dedicated to study the clouds from the provider point-of-view, whose purpose is to help evaluating the design decisions of the datacenter. Examples of such simulators are MDCSim [11], which offers specific and precise models for low-level components including network (e.g. InfiniBand or Gigabit ethernet), operating system kernel and disks. It also offers a model for energy consumption. However, the cloud client activity that can be modeled is restricted to web-servers, application-servers, or data-base applications. GreenCloud [8] follows the same purpose with a strong focus on energy consumption of cloud's network apparatus using a packet-level simulation for network communications (NS2). In the second category (which we focus on) are the simulators targeting the whole cloud ecosystem including client activity. In this category, CloudSim [2] is the most broadly used simulator in academic research. It offers simplified models regarding network communications, CPU, or disks. However, it is easily extensible and serves as the underlying simulation engine in a number of projects. Simgrid [4] is the other long-standing project, which when used in conjunction with the SchIaaS cloud interface provides similar functionalities as CloudSim. Among the other related projects is iCanCloud [15] proposed to address scalability issues encountered with CloudSim (written in Java) for the simulation of large use-cases. Most recently, PICS [7] has been proposed to evaluate specifically the simulation of public clouds. The configuration of the simulator uses only parameters that can be measured by the cloud client, namely inbound and outbound network bandwidths, average CPU power, VM boot times, and scale-in/scale-out policies. The data center is therefore seen as a black box, for which no detailed description of the hardware setting is required. The validation study of PICS under a variety of use-cases has nonetheless shown accurate predictions.

However, when the simulated system is subject to variability, it is difficult to establish the validity of simulation results formally. Indeed, given some defined inputs, a DES outputs a single deterministic result, while a real system will output slightly different results at each repeated execution. Hence, in practice the simulation is informally regarded as valid if its results are "close" to one or some of the real observations.

*Stochastic Simulation and Monte-Carlo Method.* For more comprehensive predictions in such variable environments, the simulation must be *stochastic*. In stochastic simulations inputs become random variables (RVs) representing the distribution of possible values for the parameters. The result of one such simulation is itself an RV representing the distribution of the results.

Extensive work has been done on numerical methods for solving stochastic simulations of directed acyclic graph (DAG) [10,12]. In a DAG model the vertices represent the tasks comprising the application, and the edges represent the dependencies between those tasks. The numerical approach presented in [10,12] shows that, when tasks' runtimes are independent, the makespan distribution of two successive tasks is the convolution product of the tasks' probability density function, while the makespan of two parallel tasks joining is the product of

the tasks' cumulative distribution function. This makes the numerical approach computationally intensive and its core constraint, the tasks RVs independence, can not be guaranteed in all cases. Moreover this DAG-based approach implies fixed scheduling, since the scheduling creates implicit dependencies between tasks scheduled one after another. In a cloud context where resources can be provisioned on the fly, dynamic scheduling is much more common.

Instead of numerically computing the resulting RV, a Monte-Carlo simulation (MCS) samples the possible results by testing multiple *realizations* in a deterministic fashion. A realization is obtained by drawing a runtime that follows their task's respective RV for every task in the application. This allows one to simulate each realization using traditional methods like DES. Eventually, given enough realizations, the distribution of the simulation results will tend towards the distribution of the equivalent stochastic simulation. Statistical fitting techniques can then be used to characterize this makespan RV. MCS's permits non-independent RV and dynamic scheduling. This approach was first suggested in [17] for stochastic PERT graphs. Later, in the context of grids, where the number of resources is fixed during one execution, Tang et al. [19] proposed, a modification of the well-known scheduling heuristic HEFT to compute a schedule yielding the shortest makespan given randomly variable task durations. Canon and Jeannot [3] have used MCS to evaluate the robustness of DAG schedules when task durations vary, and similarly, Zheng and Sakellariou [21] evaluated the impact of this variability on the makespan. More recently, ElasticSim [1] has been proposed as a simulator extending Cloudsim to integrate resource auto-scaling and stochastic task durations. Similarly to our work, ElasticSim computes a schedule whose objective is to minimize rental cost while meeting deadline constraints. For several generated workflows, the study compares the simulation results regarding rental cost and makespan, when varying the variability of task duration and deadline with arbitrary values. By contrast, our work focuses on how the MCS method, under some given variability assumptions, captures actual observations.

## 3 Work Context

The study conducted in this paper is built upon a real comparison between experiments run in actual environments and experimental results obtained by simulation. To strengthen the validity of the comparison, the experimental conditions for the real setup and the simulation should share as many commonalities as possible, as advocated in [16]. Our experimental setup described hereafter consists of two test applications which, on one hand, are run on a real platform with our scheduler Schlouder, and on the other hand are simulated with our simulator SimSchlouder based on SimGrid.

*Test Applications.* We carried out multiple executions of two broadly used scientific applications to evaluate Schlouder performance. The execution traces for those runs were collected in an archive. This backlog of real executions is the

benchmark against which our simulation performance will be evaluated. Those applications are:

- Montage [6], the Montage Astronomical Image Mosaic Engine, is designed to splice astronomical images. This application is a data intensive fork-join type workflow with a *communication-to-computation* ratio greater than 90%.
- OMSSA [5], the Open Mass-Spectrometry Search Algorithm, is used to analyze mass-spectrometer results. The application is a computation intensive set of independent parallel tasks with a *communication-to-computation* ratio lower than 20%.

*Real Execution Setup.* Schlouder [14] is a client-side cloud broker for IaaS capable of executing the user's batch jobs, sets of independent tasks and workflows alike. The broker's main role is to schedule the tasks onto a set of cloud resources, which the broker can scale up or down. Technically, the broker connects to the cloud management system (for instance, OpenStack) to instruct how the infrastructure should be provisioned. It then assigns the tasks to the resources using the Slurm job management system. As in most batch scheduler systems, the task description includes its runtime estimation by the user called *user estimate*. In case of a workflow, the task dependencies are also provided. Schlouder uses just-in-time scheduling where tasks are assigned to VMs as soon as all their dependencies are satisfied. A task's real runtime, called *effective runtime*, usually differs from estimated runtimes, but this does not change previous scheduling decisions. The scheduling problem in IaaS clouds is a bi-objective optimization problem, taking into account the rental cost of resources and the execution makespan. Schlouder's requests users to choose a strategy that favors one objective or the other. The scheduling and provisioning decisions are then controlled accordingly by specific *heuristics*. In this paper, we used the two following heuristics:

- ASAP (*as soon as possible*) schedules each task onto an idle VM if one is available, or provisions a new VM otherwise. This heuristic minimizes the makespan.
- AFAP (*as full as possible*) schedules each task onto one VM if it does not increase the rental cost (*i.e.* the number of BTU), or provisions a new VM otherwise. This heuristic minimizes cost by minimizing the BTU count.

*Simulated Execution Setup.* As a follow-up to our work on Schlouder we developed SimSchlouder, a simulator mimicking the behaviour of Schlouder. It has the same interfaces and implements the same scheduling heuristics as Schlouder. It uses SimGrid as its core simulation engine. In practice, SimSchlouder is included as a plugin into Schlouder to allow the user to request an estimate of the makespan and cost before choosing an heuristic for a real run. SimSchlouder shares with Schlouder a common subset of inputs, including the same tasks description and heuristic. Whereas Schlouder operates on a real cloud controller, SimSchlouder provisions simulated VMs through SimGrid's cloud interface called SchIaaS. Additionally SimSchlouder requires a platform specification,

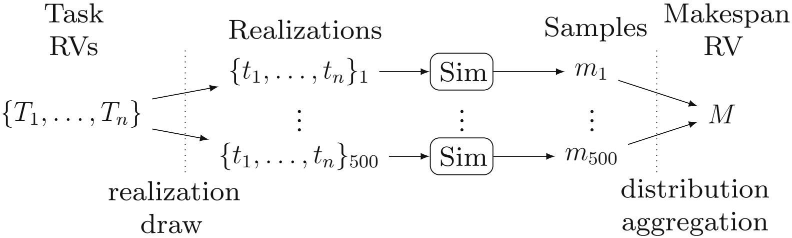


**Fig. 1.** Overview of a 500-iteration Monte-Carlo simulation.

which describe the physical nature of the cloud as well as the management rules, and the effective runtime of each task, that are used by the simulator to compute the tasks' end dates. Together, they allow the simulation to be accurately representative of reality.

## 4 Proposal: An Enriched Simulation Framework

To address the limited trustworthiness of DES in variable environments such as clouds, we propose a framework implementing the Monte-Carlo method using SimSchlouder as simulation engine. This solution combines the extensive results provided by stochastic simulations with correctness of scheduling and provisioning provided by SimSchlouder.

### 4.1 Simulation Process

The whole extended simulation process is referred to as MCS. For an application composed of $n$ tasks (as depicted Fig. 1), MCS consists in applying successive MCS-iterations. Assuming we can provide a runtime distribution $T_j$ for every task $j$, a MCS-iteration $k$ consists in:

- drawing a runtime value, $t_j$, for each task from the associated RV, $T_j$;
- proceed to a simulation using all runtimes $t_j$ to obtain a makespan $m_k$.

With enough makespans $m_k$, we can compute a statistical distribution of the makespan as a final RV noted $M$. We extend our simulation to two output variables: we will not only observe the makespan computed at every iteration but also the cost for each execution in number of BTU.

### 4.2 Real Observations

Using Schlouder (cf. Sect. 3), we performed numerous executions of the application of OMSSA and Montage. These executions were performed on a 96 cores Openstack cloud system set up on 4 identical dual 2.67 GHz Intel Xeon X5650 servers. We used the KVM hypervisor and Openstack version 2012.1 and 2014.4.

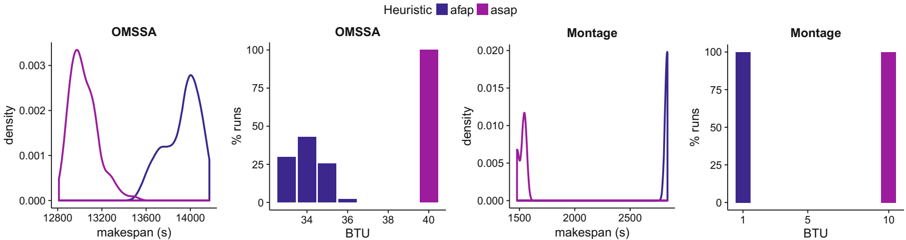


**Fig. 2.** Empirical observations for makespan distributions and #BTU.

The traces obtained from these experiments contain several useful metrics including, but not limited to, the VM start dates, boot time, shutdown times, and assigned tasks, as well as the task start date and effective runtimes. They were initially used all along the development of Schlouder and then to properly tune SimSchlouder in order to make the simulation as accurate as possible. As a result, for the execution used in this paper, simulations done with SimSchlouder are precise to the second on the makespan and systematically exact on the BTU count. Regarding variability, we find our platform variability to stand between 3% and 6% using the metrics described in the study [9] based on relative standard deviation of tasks runtimes. This variability is within the range reported in the study for platforms like Amazon's EC2 or Google Cloud Engine, with the exception of shared CPU instances.

In this paper these execution traces are used to generate our MCS input RVs using the method we will describe in Sect. 4.3 and we compare the makespan and BTU distributions of the MCS to the distribution observed in the corresponding traces. For this purpose, traces from comparable runs are grouped by application and heuristic. Figure 2 presents the distribution of resulting makespans and BTU counts. For OMSSA, ASAP yields a makespan variation in the range [12811s;13488s] (variability ≈5%) with a constant BTU count of 40, and AFAP yields [13564s;14172s] (4%) with a BTU count ranging [33;36]. For Montage, the makespans are in the range [1478s;1554s] (≈ 4%) with 10 BTUs for ASAP and [2833s;2837s] (0.1%) with 1 BTU for AFAP.

### 4.3 Input Modeling

Using a MCS we can account for this variability and provide the user with a range of possible makespans. The MCS requires a runtime RV for every task in the application. These RVs form the input model. Although precise models will yield more exact results, creating such models would not be possible in more common use-cases where a backlog of real observations is not available. In this section we propose a simple model to represent the variability of the whole system using a single factor parameter to create a small range around every estimated runtime. We test this model against our backlog of real runs. The key finding detailed hereafter is that this simple model can be precise enough for the MCS to predict over 90% of real runs.

This model for the runtimes RV uses a single expected runtime per task and a global perturbation level for all tasks. This model uses uniform distributions ($\mathcal{U}$). These RVs are centered on the expected runtime of the task they represent. The relative spread of these distributions is defined by the *perturbation level* $P$, which is the same for every task. If we assume $P$ can summarize the variability of the whole system, a central question is how should $P$ and the expected runtimes be chosen to assess the validity of the MCS. To this end, we assume a good guess for an expected runtime is the average of all effective runtimes, $\bar{r_j}$ for a given task $j$. As such the runtime distribution's RV $T_j$ for task $j$ is:

$$T_j = \mathcal{U}[\bar{r_j} \times (1 - P), \bar{r_j} \times (1 + P)] \tag{1}$$

Since the global perturbation level $P$ establishes the limit for the worst deviations from the estimated runtimes, the relative standard deviation metric used in [9] is not well suited. Instead we choose to build $P$ using the average of the worst observed deviation for every task in the application. With $r_j^n$ the $n$th runtime observation for task $j$, $P$ is set to :

$$P = \underset{j}{\text{mean}} \left( \max_n \left( \frac{|r_j^n - \bar{r_j}|}{\bar{r_j}} \right) \right) \tag{2}$$

For OMSSA, the perturbation level given by this model is $P \approx 10\%$ for both heuristics. For Montage our calculated perturbation level is $P \approx 20\%$ for ASAP and $P \approx 5\%$ for AFAP. Using a similar metric, [7] also observed most deviations to be within 10% of the average runtime when working on Amazon EC2 instances with dedicated CPUs.

*Simulation Execution.* The execution of an MCS is implemented through a series of scripts created to automate large numbers of simulations. The simulation driver first passes an application template, including dependencies and task expected runtimes, to a generator script. The generator creates the necessary number of simulation input files, with task runtimes randomised following the input model. The driver script can then execute an instance of SimSchlouder for every input file, sequentially or concurrently. Once all the instances of SimSchlouder have been executed, the result are aggregated in the MCS output file. This process is supple enough to accommodate other simulator and models as long as the user can provide a command to generate input files and another to parse simulation outputs.

## 5 Evaluation

We ran a 500-iteration MCS for every heuristic $\times$ application group using the task model described in the previous section. The resulting distributions are shown in Fig. 3. The makespan density graphs show the simulation result distribution as filled curves. The real observed executions, as in Fig. 2, are shown as non-filled curves. On the BTU count graphs, the left bar represents the empirical data, and the right bar the results from the simulation. These graphs show

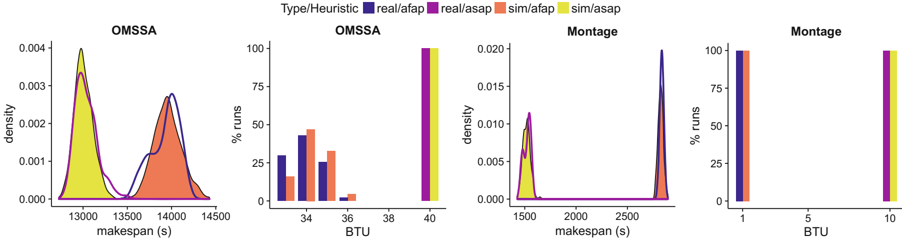


**Fig. 3.** Makespan and #BTU distributions for MCS compared to reality for $P = 10\%$.

the simulation results cover the same ranges as the real observation, but do not present the same distribution within those ranges. We quantify our simulation results correctness using statistical confidence intervals. Since the makespan is in essence the sum of the tasks' runtimes in the execution critical path and tasks are all distributed using the uniform distribution which has a finite variance, we consider the Central Limit Theorem applicable. Fitting to a normal distribution gives us an average makespan $\mu$, and a standard deviation $\sigma$. These can be used to build confidence intervals (CIs). For the normal distribution the 95% CI, defined as $[\mu - 2\sigma, \mu + 2\sigma]$ and the 99% CI, $[\mu - 3\sigma, \mu + 3\sigma]$. The capture rate expresses the number of observed real makespans that fall within a given CI relative to the total number of real observations. Table 1 presents the capture rate obtained by each interval computed after normal fitting. Additionally we provide for each interval its size relative to the average makespan. Regarding OMSSA, the MCS captures at least 90% of real observed makespans. The divergence between the capture rate and the CI expected capture rate is due to the fact that the empirical makespan distribution does not follow a perfect normal distribution. Using a 99% CI improves the capture rate up to 98%, hence very close to the theoretical expectation. Regarding Montage the MCS achieves a capture rate of 100% for any CI.

Our MCS and a simple task model can capture 90% of reality all the while producing makespan intervals of limited size, a 3% relative size representing

**Table 1.** Makespan and BTU capture rate depending on CI for P = 10%.

| Application | Heuristic | Makespan (size of CI) | | BTU |
|---|---|---|---|---|
| | | CI 95% | CI 99% | |
| OMSSA | ASAP | 90% (3%) | 98% (5%) | 100% |
| | AFAP | 92% (4%) | 100% (6%) | 100% |
| Montage | ASAP | 100% (2%) | 100% (4%) | 100% |
| | AFAP | 100% (1%) | 100% (2%) | 100% |

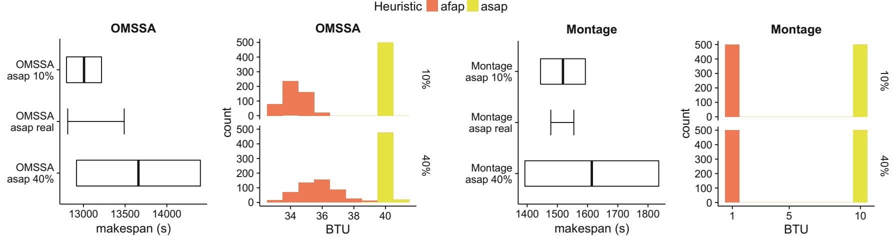


**Fig. 4.** Makespan intervals and #BTU distributions for OMSSA and Montage at different perturbation levels. In the makespan interval graph the boxes represent the 95% CI resulting from the normal fit of the MCS's results, and the bar the results of a single unperturbed simulation.

7 min on a 3 h 45 m long makespan. We consider this result a satisfactory trade-off between the simplicity of the input model and the accuracy with regards to the theoretical CI.

## 6 Perspectives

Outside of the realm of reproduction or predictions, we believe that MCS can have other more research oriented applications. In this section we will illustrate one such application. Then we will discuss limitations we have encountered in our work with MCS.

*High Perturbation Simulations.* We have so far set the perturbation level to a value that was relevant to the real system observed (see Sect. 4.3). A subsequent question is how does the prediction change when increasing this perturbation level. In this section we will focus on simulation of makespans using the ASAP heuristic. Figure 4 presents the 95% CIs obtained through the normal distribution fitting of simulations with both $P = 10\%$ and $P = 40\%$. Notice that a 40% perturbation level may be experienced in current cloud provider offers when renting shared instances ([9]). On the makespan interval graphs (first and third subfigures from left to right) the boxes represent the span of the CI interval. The mean simulated makespan ($\mu$) is represented by the vertical bar inside the interval. The middle row shows the interval of real observed makespans. Simulation of OMSSA using $P = 40\%$ exhibits a clear drift upward of the ranges of simulated makespans and BTU. This drift is significant compared to the growth of the capture interval to the point that the capture rate of the simulation with $P = 40\%$ is of only 83% when the $P = 10\%$ simulation had a 90% capture rate. Montage simulations exhibit the upwards drift but not to the extent that it affects the simulation's capture rate. These results have two interesting implications. Firstly, the perturbation level can not be used as a trade-off variable to augment capture rate at the expense of CI compactness. The lower capture rate at $P = 40\%$ is a strong indication that our real platform exhibits a variability

closer to 10% than to 40%. Misestimation of the perturbation level will have the same implication for the MCS as a wrong effective runtime given to DES. Users for whom higher capture rates are more important than interval compactness should use statistical methods to build higher rate CIs, like the 99% normal distribution CI used in Sect. 5. Secondly, this result shows that MCSs can be used to exhibit heuristic behaviours. This upwards shift of the CI shows that ASAP, an heuristic geared towards reducing the makespan regardless of cost, is not as effective when scheduling bag-of-tasks with task runtimes that might vary widely. However, the same observation on Montage shows that when scheduling workflows ASAP remains capable of low makespans. This can be explained by the scheduler's behaviour and the workflow's nature. In workflows the makespan depends only on execution of tasks in the critical path, and remains unaffected by variability of tasks outside the path. This is compounded by the just-in-time scheduling used in Schlouder, later scheduling decisions take into account the tasks' deviation from their expected runtimes. This kind of analysis can be used to gain insight in the strengths and weaknesses of any heuristic, regardless of complexity.

*Limitations of the Enriched Simulation.* In this paper all the MCS presented used 500 iterations. Such an MCS requires in average 15 min of CPU time, and iterations can be parallelized. We determined that this was enough in the context of our simulation as additional simulations did not change the results and only marginally increased the confidence of the fitting process. The number of simulations necessary in an MCS depends on the number of input variables and the distribution of these variables. A MCS works by sampling the possible scenarios to get a distribution of possible outcomes, hence when more scenarios are possible then more samples are required. The relative quick convergence (as compared to other scientific fields where MCS is used) is explained by the relatively low number of input variables found in batch job scheduling. In our case, there are respectively 223 and 184 tasks for OMSSA and Montage. As the perturbation level influences the input variable distribution, we are currently studying its relationship with the number of required MCS-iterations.

## 7 Conclusion

Predicting the execution behaviour of complex workloads in the cloud is an important challenge. While a number of works have proposed model-driven simulators, much remains to be done for their adoption in production-grade cloud settings. As advocated by Puchert et al. [16], the trust we can put in the prediction demands certainty and precision that only comes from validating simulation against empirical observation. This paper contributes to this effort in two ways. First, we propose a Monte-Carlo simulation extension to a discrete event simulator based on SimGrid. This extension provides stochastic predictions which are more informative than single values of billing cost and makespan produced by traditional discrete event simulators. The Monte-Carlo simulation must be

parameterized to draw random values from relevant value spaces. In this work we show that the variability we seek to account for can be modeled by a single parameter, called perturbation level and applied to all task runtimes. Second, we apply our model in a real setting, on two different applications, for which we have collected execution traces. At the light of these empirical observations, our study shows that the proposed model could capture over 90% of the observed makespans for all combinations of application and scheduling heuristics given an appropriate perturbation level. We now aim to test our simulator on more use-cases and platforms. In particular as a number of studies on public clouds have reported variability levels similar to our platform [7,9], we intend to reproduce these results on public clouds.

## References


1. 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). https://doi.org/10.1007/s10723-016-9390-y
2. 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)
3. Canon, L., Jeannot, E.: Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments. IEEE Trans. Parallel Distrib. Syst. **21**(4), 532–546 (2010). https://doi.org/10.1109/TPDS.2009.84
4. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. J. Parallel Distrib. Comput. **74**(10), 2899–2917 (2014). https://doi.org/10.1016/j.jpdc.2014.06.008
5. Geer, L.Y., et al.: Open mass spectrometry search algorithm. J. Proteome Res. **3**(5), 958–964 (2004)
6. Jacob, J.C., et al.: Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. Int. J. Comput. Sci. Eng. **4**(2), 73–87 (2009)
7. Kim, I.K., Wang, W., Humphrey, M.: PICS: a public IaaS cloud simulator. In: Pu, C., Mohindra, A. (eds.) 8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, 27 June–2 July 2015, pp. 211–220. IEEE Computer Society (2015). https://doi.org/10.1109/CLOUD.2015.37
8. Kliazovich, D., Bouvry, P., Khan, S.U.: GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. J. Supercomput. **62**(3), 1263–1283 (2012)
9. Leitner, P., Cito, J.: Patterns in the chaos - a study of performance variation and predictability in public IaaS clouds. ACM Trans. Internet Technol. **16**(3), 15:1–15:23 (2016). https://doi.org/10.1145/2885497
10. Li, Y.A., Antonio, J.K.: Estimating the execution time distribution for a task graph in a heterogeneous computing system. In: 6th Heterogeneous Computing Workshop, HCW 1997, Geneva, Switzerland, 1 April 1997, pp. 172–184. IEEE Computer Society (1997). https://doi.org/10.1109/HCW.1997.581419

11. Lim, S., Sharma, B., Nam, G., Kim, E., Das, C.R.: MDCSim: a multi-tier data center simulation, platform. In: Proceedings of the 2009 IEEE International Conference on Cluster Computing, 31 August–4 September 2009, New Orleans, Louisiana, USA, pp. 1–9. IEEE Computer Society (2009). https://doi.org/10.1109/CLUSTR.2009.5289159
12. Ludwig, A., Möhring, R.H., Stork, F.: A computational study on bounding the makespan distribution in stochastic project networks. Annals OR **102**(1–4), 49–64 (2001). https://doi.org/10.1023/A:1010945830113
13. Mehrotra, P., et al.: Performance evaluation of Amazon elastic compute cloud for NASA high-performance computing applications. Concurr. Comput.: Pract. Exp. **28**(4), 1041–1055 (2016). https://doi.org/10.1002/cpe.3029
14. Michon, E., Gossa, J., Genaud, S., Unbekandt, L., Kherbache, V.: Schlouder: a broker for IaaS clouds. Future Gener. Comput. Syst. **69**, 11–23 (2017). https://doi.org/10.1016/j.future.2016.09.010
15. Nuñez, A., Vázquez-Poletti, J.L., Caminero, A.C., Castañé, G.G., Carretero, J., Llorente, I.M.: iCanCloud: a flexible and scalable cloud infrastructure simulator. J. Grid Comput. **10**(1), 185–209 (2012). https://doi.org/10.1007/s10723-012-9208-5
16. Pucher, A., Gul, E., Wolski, R., Krintz, C.: Using trustworthy simulation to engineer cloud schedulers. In: 2015 IEEE International Conference on Cloud Engineering, IC2E 2015, Tempe, AZ, USA, 9–13 March 2015, pp. 256–265 (2015). https://doi.org/10.1109/IC2E.2015.14
17. van Slyke, R.M.: Monte carlo methods and the PERT problem. Oper. Res. **11**(5), 839–860 (1963). http://www.jstor.org/stable/167918
18. Stanisic, L., Thibault, S., Legrand, A., Videau, B., Méhaut, J.: Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures. Concurr. Comput.: Pract. Exp. **27**(16), 4075–4090 (2015). https://doi.org/10.1002/cpe.3555
19. Tang, X., Li, K., Liao, G., Fang, K., Wu, F.: A stochastic scheduling algorithm for precedence constrained tasks on grid. Future Gener. Comput. Syst. **27**(8), 1083–1091 (2011). https://doi.org/10.1016/j.future.2011.04.007
20. Velho, P., Schnorr, L.M., Casanova, H., Legrand, A.: On the validity of flow-level tcp network models for grid and cloud simulations. ACM Trans. Model. Comput. Simul. **23**(4), 23:1–23:26 (2013). https://doi.org/10.1145/2517448
21. Zheng, W., Sakellariou, R.: Stochastic DAG scheduling using a monte carlo approach. J. Parallel Distrib. Comput. **73**(12), 1673–1689 (2013). https://doi.org/10.1016/j.jpdc.2013.07.019