Simulation of a workflow execution as a real Cloud by adding noise

Roland Mathá[a], Sasko Ristov[a,b], Radu Prodan[a,c]

[a] *Distributed and Parallel Systems Group, Institute for Computer Science, University of Innsbruck Technikerstr. 21a, A-6020 Innsbruck, Austria*
[b] *Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, 1000 Skopje, Macedonia*
[c] *Institute of Information Technology, University of Klagenfurt, Universitätsstr. 65-67, 9020 Klagenfurt, Austria*

Abstract

Cloud computing provides a cheap and elastic platform for executing large scientific workflow applications, but it rises two challenges in prediction of makespan (total execution time): performance instability of Cloud instances and variant scheduling of dynamic schedulers. Estimating the makespan is necessary for IT managers in order to calculate the cost of execution, for which they can use Cloud simulators. However, the ideal simulated environment produces the same output for the same workflow schedule and input parameters and thus can not reproduce the Cloud variant behavior. In this paper, we define a model and a methodology to add a noise to the simulation in order to equalise its behavior with the Clouds' one. We propose several metrics to model a Cloud fluctuating behavior and then by injecting them within the simulator, it starts to behave as close as the real Cloud. Instead of using a normal distribution naively by using mean value and standard deviation of workflow tasks' runtime, we inject two noises in the tasks' runtime: noisiness of tasks within a workflow (defined as *average runtime deviation*) and noisiness provoked by the environment over the whole workflow (defined as *average environmental deviation*). In order to measure the quality of simulation by quantifying the relative difference between the simulated and measured values, we introduce the parameter *inaccuracy*. A series of experiments with different workflows and Cloud resources were conducted in order to evaluate our model and methodology. The results show that the inaccuracy of the makespan's mean value was reduced up to 59 times compared to naively using the normal distribution. Additionally, we analyse the impact of particular workflow and Cloud parameters, which shows that the Cloud performance instability is simulated more correctly for small instance type (inaccuracy of up to 11.5%), instead of medium (inaccuracy of up to 35%), regardless of the workflow. Since our approach requires collecting data by executing the workflow in the Cloud in order to learn its behavior, we conduct a comprehensive sensitivity analysis. We determine the minimum amount of data that needs to be collected or minimum number of test cases that needs to be repeated for each experiment in order to get less than 12% inaccuracy for our noising parameter. Additionally, in order to reduce the number of experiments and determine the dependency of our model against Cloud resource and workflow parameters, the conducted comprehensive sensitivity analysis shows that the correctness of our model is independent of workflow parallel section size. With our sensitivity analysis, we show that we can reduce the inaccuracy of the naive approach with only 40% of total number of executions per experiment in the learning phase. In our case, 20 executions per experiment instead of 50, and only half of all experiments, which means down to 20%, i.e. 120 test cases instead of 600.

Keywords: inaccuracy, makespan, metric, modelling, precision, simulator.

1. Introduction

Cloud has evolved in a promising platform for many scientific applications and workflows' execution [1]. Still, there are many challenges that could impact on the decision whether to migrate the execution to the Cloud [2]. Due to Cloud's performance instability [3], the cost and makespan (execution time) cannot be predicted correctly as they

Email addresses: roland@dps.uibk.ac.at (Roland Mathá), sashko@dps.uibk.ac.at (Sasko Ristov), radu@dps.uibk.ac.at (Radu Prodan)

depend on the amount of leased resources and time period of leasing. This is emphasised even more for dynamic scheduling of workflows [4], since they consist of control and data dependent tasks. On the other hand, the same instance can behave totally unexpected in two different periods of time [5], and even a small fluctuation in the task runtime will change the scheduling, which can result in huge discrepancy of makespan caused by additional network traffic between instances and shifting the starting time of tasks. Utilising more resources can reduce the makespan, but in the same time it will increase the cost; this tradeoff between the cost and makespan is analysed by many authors in the literature [6], which also impacts the Cloud's performance instability.

Instead of executing a workflow schedule (usually in hours) to estimate its makespan and cost, one can use a Cloud simulator, which can simulate the execution within seconds [7]. However, most simulators are static and will always predict the same makespan for a specific workflow schedule and configured input parameters, such as workflow structure and computation and communication requirements, available resources and their capacity or scheduling policy. For the same schedule, each simulator will almost always return the constant result for the makespan. In real Cloud environment, the makespan and cost will be always different, due to its pay-as-you-go pricing model, dynamic starting of instances [8] and performance fluctuation [9] caused by heterogeneity of underlying hardware, multi-tenancy, migrations and relocations, or other issues in order to satisfy the constraints in service level agreements.

Simulators should reproduce the execution in a real Cloud; therefore, a methodology is necessary that will make the simulations accurate and precise [10]. In this paper, we propose a new methodology how a simulator can *learn* from the Cloud's behavior and then to *configure* the model for tasks' runtime instability by introducing a noise in order to reproduce the real Cloud dynamic environment for workflow executions. As a baseline, we consider the *accuracy* of the model, which is represented through the *trueness* (i.e. how close the simulation's mean values are with the true mean values of Cloud executions) and the *precision* (i.e. how close are the corresponding standard deviations) of the simulation, as defined in ISO-5725 standard [11]. This two-phase process consists of learning and configuration phases. In the former, an agent learns the Cloud's behavior by measuring the deviation of each pair of two executions (repetitions) of the same experiment in the real Cloud. Further on, in the latter phase, the agent configures the simulator by injecting the learned behavior as a noise within the simulation. By this process, the simulation becomes "instable" by generating task execution times as a random variable distributed with some probability function, in order to behave more closely as the real Cloud. Instead of using a normal distribution naively by using mean value and standard deviation of workflow tasks' runtime, we inject two noises in the tasks' runtime: noisiness of tasks within a workflow (defined as *average runtime deviation*) and noisiness provoked by the environment over the whole workflow (defined as *average environmental deviation*). Our noised model reduces the workflow makespan simulation inaccuracy up to 59 times compared with the simulation that naively uses the mean value and standard deviation of workflow tasks' runtime. Apart of the improved accuracy of the introduced approach (or model), the sensitivity analysis shows that we need only 15 to 20 "repetitions" of some experiments in the learning phase. That is, we can reuse the learned behavior for other workflow experiments.

The rest of the paper is organised as follows. Section 2 presents the related works in modeling the workflow execution instability and the features of Cloud simulators in this domain. In Section 3, we present a short background related to specific terms of workflows and Cloud. The theoretical analysis of Cloud instability and simulating its behavior is conducted in Section 4. Our model of adding a noise in simulation is described in Section 5, followed by process of noising in Section 6. In Section 7, we evaluate our model with a series of experiments, while in Section 8, we conduct a comprehensive sensitivity analysis in order to determine the minimal number of repetitions of each experiment in real Cloud for determining our noise metric. The strength and application domain of the model, along with additional insights are discussed in Section 9. Finally, we conclude the paper and present our future work in Section 10.

2. Related Work

In this section we present the related work divided in two parts: the review of the cloud performance instability and the existing simulators and their features to simulate such instability.

2.1. Cloud performance instability

Many cloud features and parameters can cause the performance instability: heterogeneity of resources, instance types, number of instances, instance straggling, instance failures, multi-tenancy, networking bottlenecks, resource

time-sharing, etc. We present several examples of the variant cloud behavior.

An instance of the same type provides different performance for the same task over some time period. Dejun et al. [12] reported a high deviation in Amazon EC2. Jackson et al. [13] have determined that the different underlying hardware for similar instances caused performance perturbation. Schad et al. [14] detected a long-term performance instability of Amazon EC2, which was correlated also to the CPU model of the same instance type, the hour in a day and day of the week. Iosup et al. [15] determined yearly and daily patterns of performance variability, but also periods of constant performance. All these behaviors depend also on the application that is executed.

After presenting how unstable the cloud could be, the next subsection presents the existing simulators that can predict the execution of a workflow or an application in the elastic cloud environment.

2.2. Simulators

GridSim [16] was a successful pioneer in simulating the scheduling algorithms for budget and deadline constraints in grids. Although its successor CloudSim [17] extends the simulation of not only the scheduling algorithms, but also the resource provisioning in the elastic cloud environment, still, the cloud performance instability is not simulated. Chen and Deelman [18] extended the Cloudsim into WorkflowSim, in which they introduced several parameters specific for workflows, while CloudAnalyst [19] is an extension towards evaluation of large-scale cloud applications. Still, all of these extensions do not introduce the cloud performance instability.

Other works switched to develop scalable simulators that will resemble up to hundreds of thousands, even heterogeneous, machines [20]. For example, Donassolo et al. [21] extended the SimGrid [22] in order to be used for volunteered computing simulation. GroudSim [23] is also a simulator that simulates grid and cloud scalable environments, which is used in this paper for the evaluation of our methodology. Adding the scalability will improve their simulation features, but in the same time it will reduce their accuracy [24, 25]. Although all mentioned simulators estimate the execution of some application or workflow, still none of them considers the cloud performance instability.

Bux and Leser [26] went further in this direction. They developed the DynamicCloudSim simulator, also as an extension of CloudSim, in which they introduced several additional characteristics to simulate the cloud instability, such as different kind of tasks, heterogeneity, struggles and failures, and long and short term fluctuations at runtime. GloudSim [27] is also a simulator that introduces some dynamics, but only in resizing the instances and not for the task runtime. Still, the performance per resource is constant during a time period.

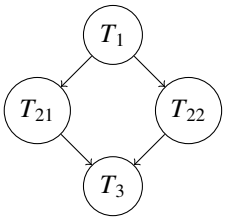
Therefore, we go a step further as we use both the accuracy and the precision in our methodology, and introduce the new metric - inaccuracy, in order to quantify and reduce the bias between the behavior of simulated and real cloud environments. Schad et al. [14] have reported that several performance parameters are unstable with a normal distribution. We also use the normal distribution to add a noise to the tasks runtime, but instead of naively generating the random variables distributed with mean value and standard deviation for runtime of a single task, we inject the noisiness of all tasks within a workflow and noisiness provoked by the environment over the whole workflow, to each task runtime. We show that our model provides better simulation than naively using the normal distribution. Even more, our experiments reported that the distribution is not normal for instance types with more computing resources (vCPUs).

3. Background

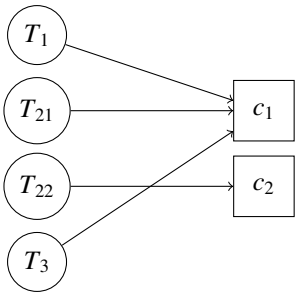
In this section we present the important background of the workflow and challenges of its execution in Cloud for a better understanding of the paper.

3.1. Workflow description

A workflow is defined as a directed acyclic graph $G = (N, E)$, where N is a set of nodes and E is a set of edges. The nodes represent the computational tasks, while the edges model the dependencies between the tasks. There are two different types of dependencies: control and data dependencies. In this paper, we consider as data dependencies file transfers, which are conducted to copy a file or files from a task that has already finished (output) to a task that needs these files as an input. We have to mention that a task can send files to one or several tasks, and a task can receive files from one or several tasks.



(a) Workflow example with 4 tasks and a parallel section



(b) Scheduling example of workflow on two Cloud instances c_1 and c_2

Figure 1: Examples for workflow and workflow execution

Fig. 1a depicts an example of a simple workflow with four tasks ($N = \{T_1, T_{21}, T_{22}, T_3\}$) and dependencies between them. Due to these dependencies, a task is ready for execution, if and only if all its predecessor tasks and incoming file transfers are finished. For instance, T_3 is ready for execution, only after both T_{21} and T_{22} are finished and all their output files are completely transfered to T_3. This shows an example when a task is receiving files from several tasks. Note that T_{21} and T_{22} are ready for execution only after T_1 is finished and all corresponding file transfers will be received. This shows an example of a task that sends files to other tasks. Since there is no dependency between each other, T_{21} and T_{22} build a parallel section and can be executed in parallel. The size of a parallel section can differ and tasks within can be the same (leading to data parallelism) or different (task parallelism).

3.2. Cloud resource and scheduling

For the workflow executions in this work, we use the Infrastructure as a Service (IaaS) Cloud service model, which delivers computing resources i.a. as Cloud instances. A cloud instance, or short instance, is a virtual machine (VM) hosted on the Cloud providers' infrastructure. The VM properties of an instance, such as the number of virtual CPUs (vCPUs), memory and storage size, etc. can be selected among a set of predefined instance types. Usually, this instance types are defined by the Cloud provider. For example, a small instance type has 1 vCPU, 2GB RAM and 20GB hard drive, while medium instance type has 2 vCPU, 4GB RAM and 40GB hard drive. The pricing model is always similar and linear: more powerful (in terms of higher computational speed or computational capacity), or greater number of instances will speedup the workflow execution, but simultaneously will increase the cost.

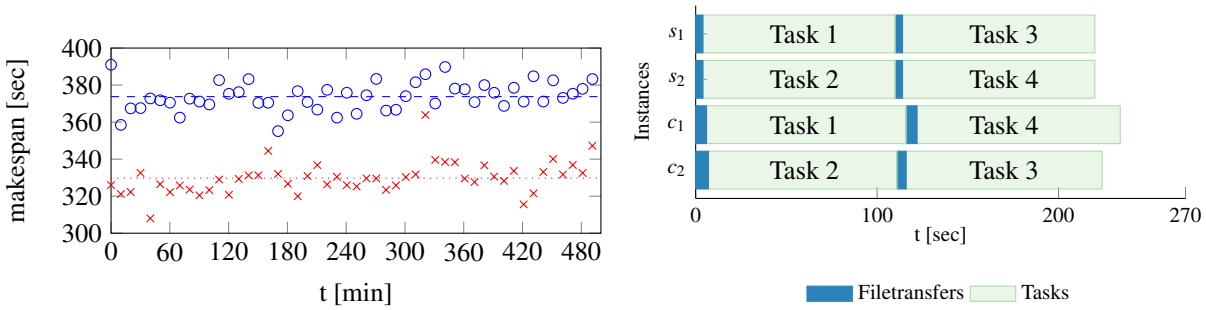
Fig. 1b presents a reasonable scheduling of the example workflow (see Fig. 1a) on two Cloud instances c_1 and c_2. In detail, T_1 is executed as first task on instance c_1. Then, according to the dependencies, the tasks T_{21} and T_{22} are executed in parallel and thus scheduled on two different instances. Finally, T_3 is ready for execution and is scheduled to instance c_1. Note that this is only one possible scheduling plan to explain the terms task scheduling and workflow execution. A state-of-the-art scheduler is dynamic, complex and will consider more information, such task requirements (e.g. minumum memory), cost, energy consumption etc. Since we want to simulate the instability of a workflow execution in a Cloud, we will use a simple job queue scheduling policy, while the scheduling policies improvement is out of the scope of this paper.

4. Theoretical Analysis

This section elaborates the reasons for Cloud performance instability and the importance of introducing a proper model in simulations. It also defines the parameters that make the performance unstable, which is especially emphasised for workflow executions.

4.1. Unstable performance in Cloud

Cloud simulators, such as Cloudsim [17] or Groudsim [23], provide a detailed workflow description and Cloud instance setup to improve the accuracy of simulations as in real distributed environment. A crucial parameter in this



(a) Unstable execution of the same workflow using two instance types in Cloud for a time period of 8 hours

(b) Example of a workflow schedule plan in two same Cloud instances c_1 and c_2 and simulated instances s_1 and s_2

Figure 2: Examples of cloud performance instability for workflow execution

scope is the task computation complexity, which is usually provided in MIPS (Million Instructions Per Second). From this parameter, along with the CPU computation speed, we can simulate the task runtime and, using some scheduling policy, also the workflow makespan. While in almost all simulators this parameter is considered to be a constant for the same workflow task and traditional computing environment, it varies during some time period in Cloud environment due to Cloud's multi-tenant and heterogeneous environment.

We have conducted two experiments, presented in Fig. 2a, which show the Cloud performance instability during a period of several hours. The experiment executes the WIEN2k/44 workflow (explained in Section 7.1), once using four small and the second time four medium instances. Both workflow executions have variant makespan of up to 10% comparing with their mean values that are presented with dashed and dotted lines.

Let's analyse what causes this performance instability. Several events impact the length and instability of the workflow makespan, such as starting time of instances, file transfer time before each task, execution time of each task and terminating the instances. Fig. 2b presents an example of executing a workflow in two environments, simulator (S) and Cloud (C). Note that we consider instance start up time as warm up period and thus we omit it in this example and in the experiments later. Similarly, we omit the termination time because the customer is not charged during this period. Without loosing any generality, let us assume that all tasks are independent, and ranked to be executed in ascending order. Also, let each environment has two available instances of the same type (s_1 and s_2 in simulator, and c_1 and c_2 in cloud). As shown, simulation will provide the results with the constant values for all mentioned parameters for both instances and all four tasks. The file transfers will be constant for the same schedule since the bandwidth and file sizes are the same, as well as each task will not change the instance where it will be executed. Therefore, the schedule of task execution, their execution time and file transfers, and thus the whole makespan, will be retained.

In contrast to the simulated environment, the Cloud behavior varies when the same workflow is executed with the same schedule. As the lower part of Fig. 2b shows, both times (for file transfers and task execution) will be different due to Cloud performance instability. The greatest difference will appear in the runtime of computational intensive tasks due to Cloud heterogeneous environment, virtualisation layer and cloud multi-tenancy. All these differences could even change the schedule of execution, that is, as presented, Task 3 could be executed on instance c_2, instead of planned instance c_1. This change in scheduling could probably cause even additional file transfers, or possibly reduce them. Nevertheless, both cases will generate performance instability.

4.2. Performance instability parameters definitions

There are many factors that can make the makespan discrepant. In this subsection we formally define those parameters and present some dependencies that will be used in the next section for modeling the noised task runtime and workflow makespan in a simulation. Table 1 presents the sets that will describe the environments where the experiments are executed, along with their details, which are input parameters in the experiments. For each parameter, we also present example values for better understanding. The set ENV presents the environments, at least by one

simulated and Cloud environments. Cloud environment can have specific types of instances, which are represented with the set IT, and a customer can use a specific number of instances of particular type, represented with the set IN. Elements of the set W denote different workfows that can be executed, each of which consists of a set T_w of tasks T_{w_w}, as explained in Section 3. Since our intention is not to improve the scheduling in simulator and Cloud, but to equalise the simulator's behavior as close as possible to the real Cloud's one, we are not interested in analysing the scheduling of tasks on instances.

Table 1: Definitions of input (environmental) parameters

Parameter description	**Formal definition**	**Example values**
Set of environments	$ENV = \{ENV_1, ENV_2, \ldots, ENV_{e_n}\}$	$Cloud, Simulator$
Set of instance types	$IT = \{IT_1, IT_2, \ldots, IT_{n_t}\}$	$Small, Medium, Large, XLarge$
Set of instance number	$IN = \{IN_1, IN_2, \ldots, IN_{n_i}\}$	1, 2, 3, 4, ...
Set of workflows	$W = \{W_1, W_2, \ldots, W_{n_w}\}$	$WIEN2k, Montage, Synthetic$
Set of tasks for W_w	$T_w = \{T_{w1}, T_{w2}, \ldots, T_{w_{n_w}}\}$	T_1, T_{21}, T_{22}, T_3 (Fig. 1a)
Set of experiments	$EXP = \{(W_w, IT_t, IN_i) \mid w \in W, t \in IT, i \in IN\}$	$(WIEN2k, small, 3)$: Execute the $WIEN2k$ workflow using 3 instances of type *small*
Set of executions	$EXE = \{1, 2, \ldots, N_x\}$	$N_x = 10$ executions (repetitions) of an experiment EXP
Set of test cases	$TC = \{(ENV_{e_i}, EXP(w,t,i), EXE_x)\}$	$(Cloud, (WIEN2k, small, 3), 7)$: *7-th* execution of the *WIEN2k* workflow using *3 small* instances in *Cloud* environment

After definition of environments and workflows, as well as their properties, the lower part of Table 1 presents the trials' parameters. We define a set of experiments EXP, such that each *experiment* $EXP(w,t,i)$ is represented as a 3-tuple of a workflow W_w that runs on specific number of instances IN_i of the same type IT_t. For example, $(WIEN2k, small, 3)$ denotes an experiment that executes $WIEN2k$ workflow using *three* instances of type *small*. In order to calculate the average values of some parameter, we need to repeat the *execution* EXE_x of each experiment the same number of times, which is denoted with the set EXE. For example, $N_x = 10$ means that each experiment will be repeated (executed) 10 times and in this case, the elements of the set EXE are integers from 1 to 10. Finally, a *test case* $TC(ENV_e, EXP(w,t,i), EXE_x)$ represents a specific execution EXE_x of the experiment $EXP(w,t,i)$ on a specific environment ENV_e. For example, $(Cloud, (WIEN2k, small, 3), 7)$ denotes the test case, which is *7-th* execution (repetition) of the experiment $(WIEN2k, small, 3)$ in Cloud environment, while $(WIEN2k, small, 3)$ denotes the experiment with *WIEN2k* workflow and *three (3) small* instances. Examples of real values for these specific parameters are presented in Section 7.1.

Output parameters, which should be measured in each test case of an experiment, are presented in Table 2. In order to simplify the presentation, we are using T for a task T_{w_w}, TC for a test case $TC(ENV_e, EXP(w,t,i), EXE_x)$ and EXP for an experiment $EXP(w,t,i)$. Each task T of a test case TC, is executed from the *starting time* $t_{start}(T, TC)$ until the *finish time* $t_{fin}(T, TC)$. We define this elapsed time as *runtime* t_{run} of a task T, which is formally defined as $t_{run}(T, TC) = t_{fin}(T, TC) - t_{start}(T, TC)$. Further on, the *makespan* $M(TC)$ of each test case TC of an experiment EXP is measured as the elapsed time from the start time of the *entry task* T_{entry} until the finish time of the *end task* T_{end} of that test case TC. After executing all test cases of an experiment EXP in an environment ENV, the mean value of the makespan $\overline{M}(EXP, ENV)$ is calculated, considering all conducted test cases of that experiment EXP in that environment ENV.

4.3. Simulation inaccuracy

After definition of all parameters, we need a methodology in order to quantify the simulation and compare how much it is accurate and precise. For this purpose, we introduce the metric *inaccuracy* δ as a relative distance of a simulated value to the true measured value, which is defined in (1). This is a template definition, which can be

Table 2: Definitions of output (measured) parameters

Parameter description	**Formal definition**
Start time of T	$t_{start}(T, TC)$
Finish time of T	$t_{fin}(T, TC)$
Runtime of T	$t_{run}(T, TC) = t_{fin}(T, TC) - t_{start}(T, TC)$
Makespan of TC	$M(TC) = t_{fin}(T_{end}, TC) - t_{start}(T_{entry}, TC)$
Mean makespan of EXP in ENV	$\overline{M}(EXP, ENV) = \overline{M(TC)}, \forall EXE_x$ in EXP

overloaded with mean values of each defined parameter in the previous subsection.

$$\delta = \frac{| < Measured > - < Simulated > |}{< Measured >} \tag{1}$$

Since the accuracy is defined as the closeness of the simulated value to the measured value, reducing the simulation's inaccuracy means improving its accuracy.

5. Modeling the noising

Since most common Cloud simulators provide a user defined, but constant setup for workflow tasks, the output workflow makespan will be constant. However, as both experiments show in Section 4.1, the Cloud behavior is discrepant during some period of time, in our case up to 10%, while simulators should be configured to reproduce the same behavior. A naive approach of statistics would be to generate an execution time of each task as a random variable with a normal distribution. In this section we present a novel model how to add noise in the runtime of each task type into a simulation.

The goal of our noising model is to be as simple as possible, but in the same time to be most accurate. The problem here is that we do not have a single value that will be the goal to be targeted, but an unstable behavior. So, we are looking to reduce its inaccuracy δ.

Since we want a simple model, our focus is not to add a noise in all parameters, but to the runtime of each task execution only, which will automatically make the makespan unstable. The modeling has two steps. First, we model the noisiness of tasks within a workflow (defined as *average runtime deviation*), and then we model the noisiness provoked by the environment over a workflow (*average environmental deviation*).

5.1. Tasks noisiness within a workflow

In order to model the noisiness of each task within a workflow, first we model the noisiness of a task T itself with the parameter *task runtime deviation* $\rho(T, TC_1, TC_2)$, which is defined in (2) as a relative runtime difference of that task, comparing executions of two test cases of the same experiment in Cloud.

$$\rho(T, TC_1, TC_2) = \frac{|t_{run}(T, TC_1) - t_{run}(T, TC_2)|}{max(t_{run}(T, TC_1), t_{run}(T, TC_2))} \tag{2}$$

Now we use the task runtime deviation $\rho(T, TC_1, TC_2)$ in order to introduce a parameter *average runtime deviation* $\overline{\Delta}_{runtime}$, which will model the noisiness of each task runtime in a workflow, as defined in (3). It represents a normalised mean runtime difference of all corresponding tasks $T \in T_w$ of two test cases $TC_1 = TC(ENV, EXP, EXE_j)$ and $TC_2 = TC(ENV, EXP, EXE_k)$, which are two different executions of the same experiment $EXP = EXP(w, t, i)$ in the same environment ENV. The same experiment $EXP = EXP(w, t, i)$ means that the workflow W_w should be executed using the same number of instances i, each of instance type t.

$$\overline{\Delta}_{runtime}(TC_1, TC_2) = \frac{1}{|T_w|} \cdot \sum_{\forall T \in T_w} \rho(T, TC_1, TC_2) \tag{3}$$

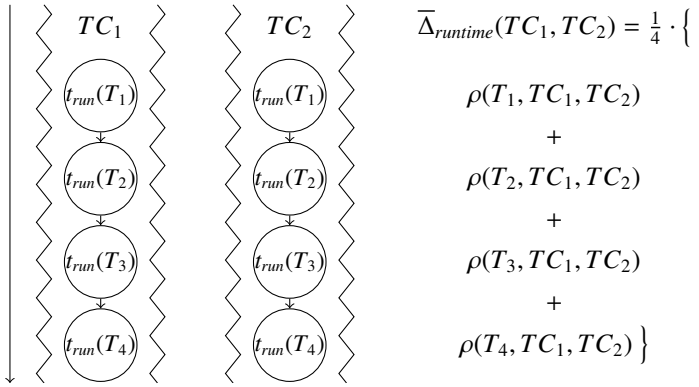


Figure 3: Example of modeling the average runtime deviation $\overline{\Delta}_{runtime}$ with two test cases of a workflow execution

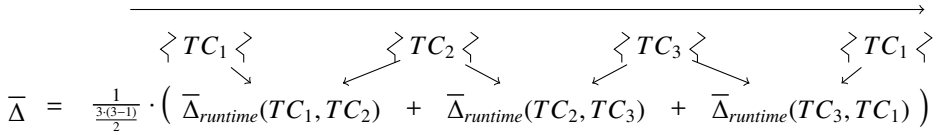


Figure 4: Example of modeling the average environmental deviation $\overline{\Delta}$ with three test cases of a workflow execution of the same experiment in the same environment

Let's give an example with a simple workflow that has four tasks $T_j, \forall j \in \{1, 2, 3, 4\}$ and is executed in two test cases TC_1 and TC_2, as shown in Fig. 3. In each test case, we measure the runtime of each task $t_{run}(T)$ of the workflow. After the execution, we calculate task runtime deviation $\rho(T, TC_1, TC_2)$ of each task T according to (2), and finally the average runtime deviation $\overline{\Delta}_{runtime}$ according to (3).

As defined, the average runtime deviation $\overline{\Delta}_{runtime}$ can be used for two purposes: learn the noisiness of a single environment (real Cloud) or measure the inaccuracy of simulation by comparing its values for different environments (real Cloud versus simulation). This parameter includes vertical average of all task runtimes within a workflow.

5.2. Environment noisiness

In order to be more accurate in a specific environment ENV, we introduce the *average environmental deviation* $\overline{\Delta}(EXP, ENV)$, which measures the average of an experiment's makespan instability when being repeatedly executed N_x times in a single environment, as defined in (4). That is, we measure the average runtime deviation $\overline{\Delta}_{runtime}$ of each unique pair of test cases, which represent two different executions (repetitions) EXE_j and EXE_k of the same experiment EXP (workflow, instance type, instance number) in the same environment ENV (some real Cloud).

$$\overline{\Delta}(EXP, ENV) = \frac{1}{\frac{N_x \cdot (N_x - 1)}{2}} \cdot \sum_{\forall j < k \leq N_x} \left(\overline{\Delta}_{runtime} \Big(TC(ENV, EXP, EXE_j), TC(ENV, EXP, EXE_k) \Big) \right) \tag{4}$$

Fig. 4 shows an example of modeling the average environmental deviation $\overline{\Delta}(EXP_x, ENV_e)$ with three test cases of a workflow execution of the same experiment in the same Cloud environment. The average runtime deviation $\overline{\Delta}_{runtime}$ is calculated for each pair of test cases, as presented in Fig. 3, and then the mean value of all of them represents the average environmental deviation $\overline{\Delta}(EXP, ENV)$. This parameter includes horizontal average of all workflow executions within a single Cloud environment.

We present the test case TC_1 two times in Fig. 4 just for better understanding how the average environmental deviation $\overline{\Delta}(EXP_x, ENV_e)$ is modeled. For greater number of N_x executions, the total number of calculations for the average runtime deviation $\overline{\Delta}_{runtime}$ will be $\binom{N_x}{2} = \frac{N_x \cdot (N_x - 1)}{2}$, which is the number of pairs that can be generated from a set with N_x elements.

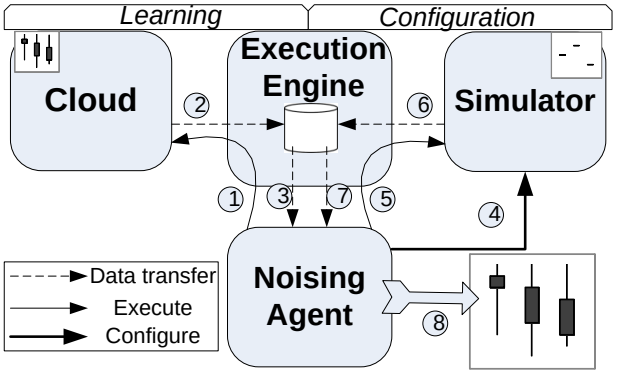


Figure 5: Noise process architecture

5.3. Putting it all together: Modeling the noise for the task runtime

Now, when we have all necessary deviations, we define how to noise the runtime of tasks, in order to improve the *trueness*, which is represented with the accuracy and precision of the simulations. Let $\sigma(\overline{\Delta})$ denotes the standard deviation of the average environmental deviation $\overline{\Delta}(EXP, ENV)$. The *noised runtime* $\tau_{noise}(T)$ of a task T is defined in (5), where $\bar{t}_{run}$ denotes the runtime's mean value of N_x executions (repetitions) of the same task T of an experiment in real Cloud environment, while $gaussian(Mean, STDEV)$ is a random function with gaussian (normal) distribution.

$$\tau_{noise}(TC) = (1 + \overline{\Delta}) \cdot \bar{t}_{run}(T) + gaussian(0, \sigma(\overline{\Delta})) \tag{5}$$

The noise in noised runtime τ of a task is modeled as a gaussian distribution, where the mean value is the runtime's mean value $\bar{t}_{run}(T)$ of the tasks in the cloud environment, shifted with the average environmental deviation $\overline{\Delta}$ in order to improve the accuracy, while its standard deviation, denoted as $\sigma(\overline{\Delta})$, is used to add noise to the precision. Note, that we are also introducing a normal (Gaussian) distribution of the task runtime τ_{naive}, as, for example, Bux et al. [26] or Poola et al. [28]. However, we shift the mean runtime and add a noise in the precision (deviation) of each task. Let us explain the significance of shifting the runtime's mean value $\bar{t}_{run}(T)$ with average environmental deviation $\overline{\Delta}$, i.e, $(1 + \overline{\Delta}) \cdot \bar{t}_{run}(T)$. Since the average environmental deviation $\overline{\Delta}$ depends on the average runtime deviation $\overline{\Delta}_{runtime}$, which accumulates noises of all tasks within a workflow, we inject this common noise of all tasks within their mean value $\bar{t}_{run}(T)$.

A normal distribution for the task runtime, on the other side, can be naively used as defined in (6). We show in our evaluation that our noising model (5) provides smaller inaccuracy than using the normal distribution naively (6) for tasks' runtime.

$$\tau_{naive}(TC) = \quad \bar{t}_{run}(T) + gaussian(0, \sigma(\bar{t}_{run}(T))) \tag{6}$$

6. Process of Noising

In this section we present a generic architecture of our noising process and map it to our case study that will be used to evaluate the correctness of our noising model.

6.1. Architecture

The generic architecture of the noising process is depicted in Fig. 5. It consists of four parts: Noising Agent, Execution Engine, and two environments, Cloud and Simulator. The process is divided into a learning and configuration phase, which are described in the following subsections.

Table 3: Instance Types of our private OpenStack cloud

Type	VCPUs	RAM	HDD
small	1	2 GB	20 GB
medium	2	4 GB	40 GB

6.1.1. Learning Phase

The learning phase gathers knowledge of real Cloud test cases and uses it for the later configuration of the simulator. To generate real data, Noising Agent uses Execution Engine to run N_x test cases TC of experiment EXP in the Cloud environment, i.e. a specific workflow using some number of instances of one type. The measured data of each test case is stored in a central database, where Noising Agent has a read access. After executing all N_x test cases, the Noising Agent queries the database and calculates the runtime's mean value $\bar{t}_{run}(T)$, the average environmental deviation $\overline{\Delta}$ and its standard deviation $\sigma(\overline{\Delta})$. In order to determine or estimate the minimum number of test cases N_x that is enough to be executed in order to provide "acceptable" trueness simulation of a single experiment, we conduct a comprehensive sensitivity analysis in Section 8. Additionally, the result of our sensitivity analysis shows that our methodology can reduce even the number of different experiments, which significantly reduces the time and cost for the learning phase. For example, instead of learning the execution of a workflow WIEN2k/44 by using for example three small instances, we can use the prior learning phase of executing another experiment with the smaller workflow WIEN2k/13, which is faster and cheaper.

6.1.2. Configuration Phase

After the learning phase, Noising Agent injects the noised runtimes $\tau(TC)$ into the simulator. Then, Noising Agent executes the same experiment EXP in the simulator and stores the measured data in the database. Finally, Noising Agent analyses and presents the simulator's inaccuracy.

6.2. Case study

We created a case study to evaluate the process of noising. Askalon [29] is used as Execution Engine, which supports workflow executions in different Clouds and Cloud simulator Groudsim [23]. As a Cloud environment, we use a private Openstack Cloud with a total of 40 CPU cores of Intel(R) Xeon(R) CPU E5-2680 v2 with 2GHz. The Openstack resources of two instance types are specified in Table 3. The VM image is with CentOS 6.3 as a guest operating system. As a simulator, we use the already included simulator Groudsim [23] in Askalon. As part of Askalon, a monitoring tool stores the monitored data in a central PostgreSQL database.

7. Evaluation

In this section we present the results of a series of experiments with different workflows, instance types and number of instances in order to evaluate the accuracy of our model and noising methodology. Each experiment is repeated in the Cloud to learn its behavior and then reproduced in the simulator more successful than the naive approach.

First, we describe the testing methodology as one can reproduce the results and then we present the experimental results. In the experimental results, we analyse the inaccuracy of the results by comparing the mean workflow makespans of experiments in Cloud and simulator.

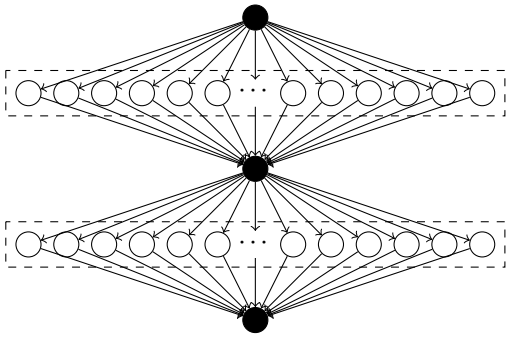
7.1. Testing methodology

The presented testing methodology offers multitude options not only to evaluate our noising model, but also the impact of various parameters on the Cloud performance instability. We present all necessary data in detail in order to make the experiments easily reproducible.

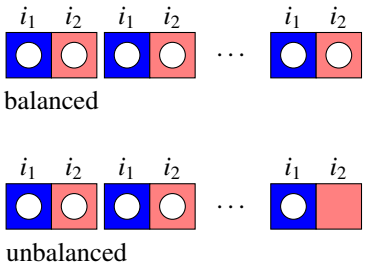
Table 4 presents different values (elements) for all defined parameters (sets) that are used in evaluation. The experiments were executed in four different environments C, S, S_{naive}, and S_{noise}. The cloud environment C is our private Openstack cloud, where many users execute multiple tasks and workflows concurrently. Other three

Table 4: Test methodology

Parameter description	Test values
Set of environments	$ENV = \{C, S, S_{naive}, S_{noise}\}$
Set of instance types	$IT = \{Small, Medium\}$
Set of instance number	$IN = \{2, 3, 4\}$
Set of workflows	$W = \{WIEN2k/13, WIEN2k/44\}$
Set of executions	$EXE = \{1, 2, \ldots, 50\}$



(a) An example of the WIEN2k workflow



(b) Balanced and unbalanced execution of a parallel part

Figure 6: The structure of the WIEN2k workflow and example of balanced and unbalanced execution of its parallel parts.

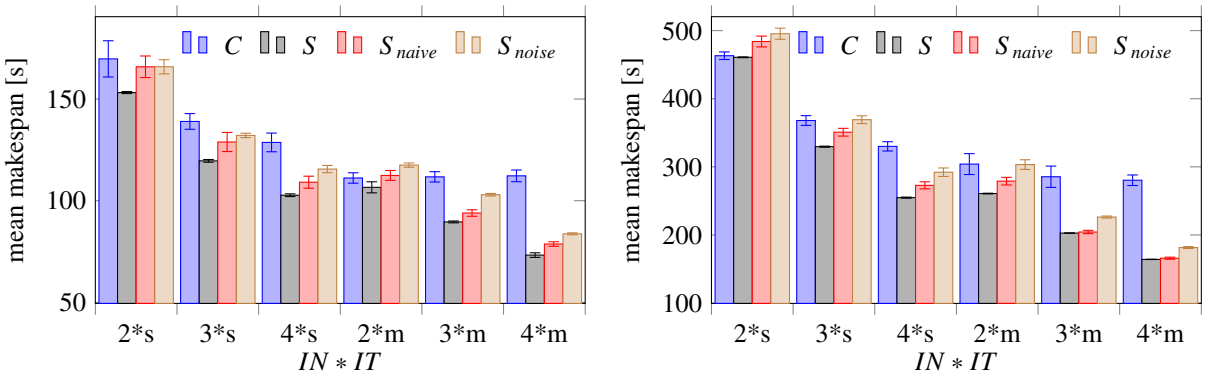
environments are simulated with the GroudSim simulator. The environment S is a simulation without any noise, or $\tau_S(TC) = \bar{t}_{run}(T)$. The S_{naive} assumes a normal distribution of the task runtimes, which is represented as a normally distributed random variable with the mean value and the standard deviation of all executions of a single experiment in Cloud environment for that task T, as defined in (6). The S_{noise} environment is our approach simulated with noised task runtime defined in (5).

In all four environments, we define a set of instance types as $IT = \{Small, Medium\}$ and the number of instances as $IN = \{2, 3, 4\}$. The set of workflow consists of two elements (workflows), i.e. WIEN2k [30], with 13 and 44 same tasks in the parallel section. It consists of two parallel sections with synchronizing tasks in between, as presented in Fig. 6a. Although WIEN2k is a simple workflow, still, it offers various characteristics to be investigated, which could impact the performance instability. The number of tasks within a parallel section is defined by an input parameter workflow size. In the experiments we use the workflow sizes 13 and 44, as defined in Table 4. The sizes of a workflow's parallel section are chosen such that 13 is a prime number, while 44 can be divided with the number of instances 2 and 4. Therefore, it is expected that the test cases with two and four instances will execute the parallel sections of the workflow $WIEN2k/44$ with optimal load balancing, while the workflow $WIEN2k/13$ execution will be unbalanced. Both workflow executions with three instances will be unbalanced. Fig. 6b shows an example of unbalanced and balanced execution of the parallel section of the WIEN2k workflow on two instances i_1 and i_2.

To summarise, we execute 12 experiments in the Cloud, each executed (repeated) $N_x = 50$ times. Further on, all these 600 test cases are reproduced in three simulator environments, leading to a total number of 2400 executed test cases.

7.2. Corollary of tasks' runtime noising: noise of the makespan

Adding the noise in the task runtimes generates noise in the makespan, as well. This subsection evaluates how the workflow makespans in real Cloud (C) is reproduction with other three simulator environments: *i*) S without any noise, *ii*) the naive approach S_{naive}, and *iii*) our noise approach S_{noise}.



(a) WIEN2k/13

(b) WIEN2k/44

Figure 7: Mean makespans of WIEN2k/13 and WIEN2k/44 on small and medium instance types, The presented variation of mean makespan is $\pm\sigma(mean\ makespan)$

7.2.1. WIEN2k/13

The results of mean makespan, along with its standard deviation, for all experiments with the WIEN2k/13 workflow are depicted in Fig. 7a. Each experiment with the WIEN2k/13 workflow is denoted as the product of number of instances and the abbreviation of its type (*s* for small and *m* for medium). For each experiment, each of four columns represents the makespan for the environment where it is executed.

Our model S_{noise} shows the lowest inaccuracy for almost all experiments in simulated environments. For small instance types, our model leads the race with the inaccuracy of 2.29% up to 10.19%, while S_{naive}'s inaccuracy is from 2.29% up to 15.21% and S's is in the range of 9.75% and 20.18%.

Similar results are achieved for medium instances, but with greater inaccuracy than experiments with small instance types. Our model still shows the smallest inaccuracy, in the range of 5.68% up to 25.40%, while the S_{naive}'s inaccuracy is between 1.13% and 29.87%, and S's is from 4.13% up to 34.72%.

We observe that in experiments with two medium instances, both simulation environments S_{naive} and S_{noise} show up to 5.68% higher makespans, while S remains again below the real Cloud experiments C. We explain this behavior due to the fact, that the makespan's inaccuracy of S is already very small with 4.13 and both noising simulation environments are based on S. Additionally, the greater inaccuracy of the experiments with medium instance types is due to tasks' interference within the medium instances, while better task isolation in small instance types. That is, in the experiments that use small instance types, each instance gets at most one task per time, due to the preemptive execution, while in those with medium instance, the instances execute two tasks concurrently, which generates tasks interference [31].

Let us discuss an additional paradoxical observation. The inaccuracy is also greater for all simulated environments with greater number of instances. We explain this insight with the fact that file transfers are more discrepant in these experiments, as tasks can be executed in different instances and thus generate additional or reduce the number of file transfers. On the other hand, although one can conclude that these experiments provide greater inaccuracy in the makespan's mean value, still our opinion is that the Cloud's behavior is unexpected. That is, the mean value of experiments with two, three and four medium instances is similar in Cloud, which is a paradox since we use more instances and expect some speedup, as the results of the experiments with small instances. Observing the simulators behavior for all three models, we can conclude that their behaviors are expected. Once again, the Cloud shows its unpredicted performance behavior.

7.2.2. WIEN2k/44

Fig. 7b depicts the average makespan results for all experiments with WIEN2k/44 workflow. Our noising S_{noise} model shows again the lowest inaccuracy of the workflow makespan in all simulated environments. When using small instances, the inaccuracy is in the range of only 0.33% up to 11.49%, compared to the S_{naive}'s inaccuracy of 4.5%

to 17.34% and S's from 0.5% to 22.81%. For the experiments with medium instances, the inaccuracy of our noising S_{noise} model is from only 0.24% up to 35.27%. The other simulated environments provided again worse inaccuracy. The S_{naive}'s reported from 8.23% to 40.83% inaccuracy, while the S's is in the range of 14.22% to 41.46%.

We observe that in the experiment with two small instances, S_{naive} and S_{noise} show up to 6.97% higher makespans than the real Cloud C. As we already mentioned in the previous subsection for $WIEN2k/13$, those results are due to the fact that both noising approaches are based on S and add additional task runtime noise. More precisely, S has already a very small inaccuracy of 0.5% in experiments with two small instances and adding task runtime noise results in higher makespans, which is important especially for increasing number of instances and vCPUs.

The maximal improvement of inaccuracy is observed for experiment with two medium instances, where our model shows the inaccuracy of only 0.24% compared to S_{naive}'s 14.22%, which is an improvement of even 59 times. However, we observe the same high inaccuracy for all simulation environments when using medium instances, especially with four medium instances. The explanation for this is the same as for WIEN2k/13 workflow, although the Cloud experiments slightly reduced the average makespan when more medium instances are used. That is, using more medium instance achieves small, but inefficient speedup, while the simulations provided the range of the speedup for the makespan as expected.

8. Sensitivity Analysis

In this section, we present the results of a sensitivity analysis in order to reduce the total number of test cases that should be executed in the learning phase and still to get accurate simulation. First, we determine the minimum number of test cases per experiment (repetitive workflow executions) that are need in the real Cloud in order to minimise the learning phase effort represented as execution time and cost. Second, we determine the minimum number of Cloud experiments by comparing and analysing three aspects: workflow type, instance type, and number of instances. Our idea is to determine a correlation between two experiments that differ with exactly one parameter and use the results of one experiment for another, instead of conducting all N_x executions of that experiment.

8.1. Determine the minimum number of workflow executions (test cases) per experiment

In Section 7, we calculated the average environmental deviation $\overline{\Delta}$ by using 50 workflow executions per experiment in the learning phase. In order to analyse the sensitivity of the average environmental deviation $\overline{\Delta}$ according to different number of repeating workflow executions in real Cloud, we recalculate $\overline{\Delta}$ with only a subset of the original workflow execution sets, denoted as $\overline{\Delta}_{subset}$. For verification and validation, we choose first 5, 10, 15, 20, and 25 workflow executions as a subset, which corresponds to 10%, 20%, 30%, 40%, and 50% of the original sets, and for each of them we calculate $\overline{\Delta}_{subset}$.

The results of the sensitivity analysis for all experiments are depicted in Fig. 8. In detail, Fig. 8a and Fig. 8b show the relative subset deviation $\overline{\rho}_{subset}(n_x) = \overline{\Delta}_{subset}(n_x)/\overline{\Delta}$ for each experiment using the WIEN2k/13 and WIEN2k/44 workflows, correspondingly. The parameter n_x shows the size of subset.

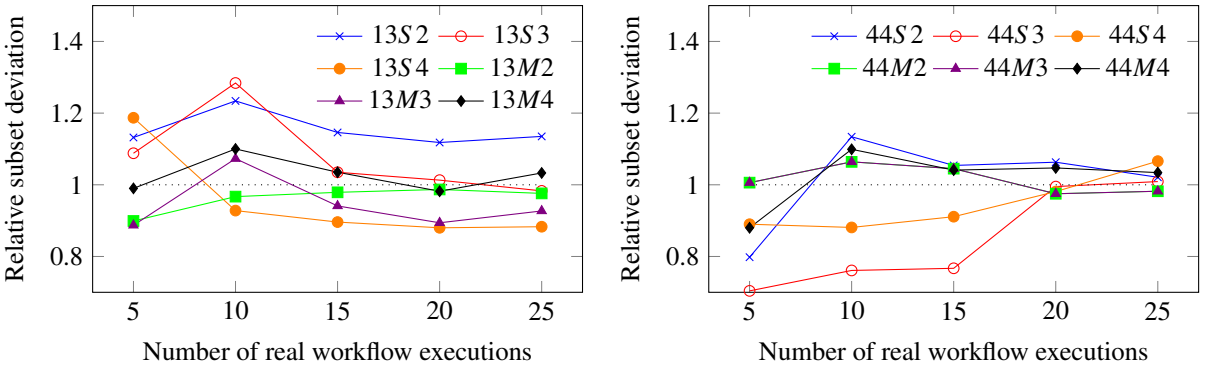
We observe that for all experiments, the relative subset deviation saturates after 20 workflow executions. In detail, after 20 workflow executions, the relative subset deviation is less than 12.02% for all WIEN2k/13 experiments and even less than 6.34% for all WIEN2k/44 experiments.

Compliant with our observation, Bux and Leser [26] used 20 workflow executions per experiment to evaluate the accuracy and precision of their model without presenting a sensitivity analysis. Our sensitivity analysis reports similar number of required workflow executions in the learning phase, regardless the difference of used workflow, simulator and Cloud environments, that is, they used the Montage workflow on Amazon EC2 and CloudSim, while our experiments used two WIEN2k workflows, on OpenStack Cloud and Groudsim simulator.

8.2. Determine minimum number of Cloud experiments

Previous subsection presents how we can determine the number of minimal executions per experiment, while still achieving the accurate simulation. In this subsection we go further, that is, we analyse the correlation of various experiments in order to reduce even the number of experiments with all executions (repetitions) in the learning phase.

In order to determine the impact of a specific parameter (workflow, instance type or instance number) to our noising parameter average environmental deviation $\overline{\Delta}$, we introduce the *average parameter deviation* $\overline{\Delta}_{<Parameter><Value>}$,



(a) WIEN2k/13

(b) WIEN2k/44

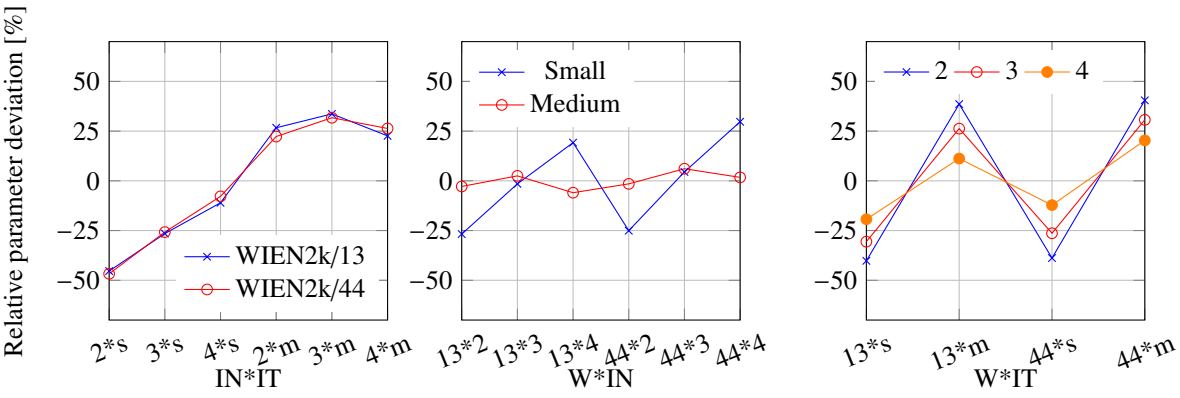
Figure 8: Sensitivity analysis for WIEN2k/13 and WIEN2k/44 by using different subset of executions (repetitions)

Table 5: Average environmental deviation $\overline{\Delta}(EXP_x, ENV_e)$ for each experiment

Experiment	$\overline{\Delta}_{W13}$	$\overline{\Delta}_{W44}$	$\overline{\Delta}_{ITS}$	$\overline{\Delta}_{ITM}$	$\overline{\Delta}_{IN2}$	$\overline{\Delta}_{IN3}$	$\overline{\Delta}_{IN4}$
13S2	0.079		0.079		0.079		
13S3	0.106		0.106			0.106	
13S4	0.129		0.129				0.129
13M2	0.183			0.183	0.183		
13M3	0.193			0.193		0.193	
13M4	0.177			0.177			0.177
44S2		0.081	0.081		0.081		
44S3		0.113	0.113			0.113	
44S4		0.140	0.140				0.140
44M2		0.186		0.186	0.186		
44M3		0.200		0.200		0.200	
44M4		0.192		0.192			0.192
AVG	**0.145**	**0.152**	**0.108**	**0.189**	**0.132**	**0.153**	**0.160**

which determines the average value of average environmental deviation $\overline{\Delta}$ for all experiments keeping exactly one parameter constant, and changing others two (e.g. SELECT AVG($\overline{\Delta}$) FROM experiments WHERE EXP.Parameter = *value*) Index in each average parameter deviation $\overline{\Delta}_{W13}$, $\overline{\Delta}_{W44}$, $\overline{\Delta}_{ITS}$, $\overline{\Delta}_{ITM}$, $\overline{\Delta}_{IN2}$, $\overline{\Delta}_{IN3}$, and $\overline{\Delta}_{IN4}$ consists of two parts $< Parameter >< Value >$ and denotes the experiments with that value of the parameter that should be considered. Table 5 presents the values of all seven average parameter deviation and how they are calculated. The rows represent the value of average environmental deviation $\overline{\Delta}$ for a specific experiment. For example, $13M3$ represents the experiment $EXP(WIEN2k/13, Medium, 3)$, that is, executing the workflow $WIEN2k/13$ using *three Medium* instances. The cells are fulfilled with the value of average environmental deviation $\overline{\Delta}$ for those experiments that have the corresponding parameter of the average parameter deviation. For example, the second column, or $\overline{\Delta}_{W44}$, considers values for average environmental deviation $\overline{\Delta}$ of all experiments with the $WIEN2k/44$ workflow, regardless of instance type and number of instances, or experiments that start with 44.

Observing the values for average parameter deviations, we can conclude that average parameter deviations $\overline{\Delta}_{W13}$ and $\overline{\Delta}_{W44}$ are similar, or that different workflow has neglected impact compared to other two parameters. Instance number has greater impact than the workflow, but the greatest impact has the instance type. For both parameters, increasing the size or number of instances provides greater performance instability.



(a) Analysis by workflow type (b) Analysis by instance type (c) Analysis by instance number

Figure 9: Cloud experiments analysis by workflow type, instance type and number of instances

Although Table 5 yields valuable conclusions, still we want to quantify the impact of each parameter. Therefore, we continue our sensitivity analysis by introducing *relative parameter deviation* $\delta_{<Parameter><Value>}(EXP)$ for an experiment, which shows the relative difference of average environmental deviation $\overline{\Delta}$ to the corresponding average parameter deviation, as defined in 7. For example, $\delta_{W13}(13M2) = (0.183 - 0.145)/0.145 = 26.21\%$.

$$\delta_{<Parameter><Value>}(EXP) = \frac{\overline{\Delta}(EXP) - \overline{\Delta}_{<Parameter><Value>}}{\overline{\Delta}_{<Parameter><Value>}} \tag{7}$$

Fig. 9 presents the relative parameter deviation for each experiment, separately for each parameter (a) workflow, b) instance type and c) instance number). We clearly observe that there is a completely matching for the curves in Fig. 9a for each value of other two parameters. However, the relative parameter deviation varies from −46.74% to 33.64% such that small instance type has lower deviation than the medium type. Figs. 9b and 9c show that other two parameters (instance type and number of instances) have impact to the performance deviation, as the relative parameter deviation is different for each value of both parameters.

9. Discussion

This section discusses about some additional insights of our versatile methodology and evaluation. It also discusses which parameters mostly impact the performance instability.

9.1. Strength of the model

We have conducted 2400 test cases with various parameters and in different environments, that is, different workflows, instance types, instance number and environments. The results confirmed that our noising methodology can be used for simulating the Cloud performance instability for workflow execution, and it is more accurate than naively noising with the normal distribution approach. Note that both, noised and naive approach, use normal distribution, but in contrast to Bux et al. [26], we assume that a task runtime is more likely to be slower instead of faster than the average task runtime. Thus, similar to negative skew we shift the mean runtime and add a noise in the precision (deviation) of each task. This improves both the accuracy and precision of simulation.

The strength of our S_{noise} model, comparing with the naive normal distribution approach S_{naive}, relies on its parameters, especially the average environmental deviation $\overline{\Delta}$ and average runtime deviation $\overline{\Delta}_{runtime}$. The former estimates the environmental noise by calculating the horizontal average of the same task, while the latter smooths the impact of the workflow structure by calculating the average of all tasks in a workflow. With these two parameters, we inject not only the noise of a task itself, when being executed in Cloud, but we inject the impact of common tasks'

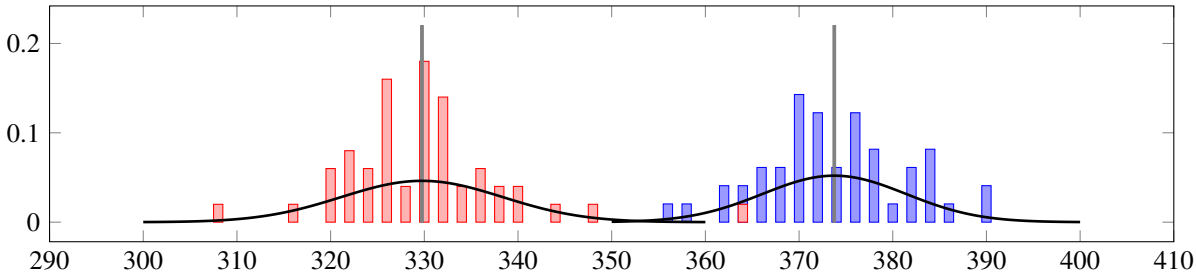


Figure 10: Distribution of makespans from Fig. 2a

noises within the workflow and Cloud environmental noise provided when a whole workflow is being repeatedly executed in that environment.

9.2. Additional insights

For small instances, all environments behave as expected, that is, simulated makespan follows the Cloud's behavior when the instance number is increased. However, all simulated models failed to follow the Cloud's behavior for medium instances, although they behave as expected - they reduce the makespan when the number of instances is increased.

In addition to this conclusion, let us analyse the Cloud's instable behavior when the same number of resources are used, but organised differently. That is, experiments 4 x small (horizontal scaling) vs 2 x medium (vertical scaling), where both experiments use a total of four cores and 8GB RAM. Simulation shows smaller average makespan for horizontal scaling, while the cloud provides smaller average makespan for vertical scaling. This insight opens new research challenges for simulating multi-tenant execution of tasks in a single instance, or known as task interference. Probably this is one of the reasons why simulation failed to follow Cloud's behavior for experiments with medium instances.

9.3. Distribution of noise

The distribution function of makespan can explain why simulations based on normal distribution provides worse results for medium instance experiments. Let us discuss this paradox. Fig. 10 presents the distribution of both test experiments (four medium and four small) presented in Fig. 2a, by grouping all points to the nearest even number of seconds. We observe that both distributions are similar to normal distribution, but still, they are far from ideal symmetric curve. The distribution function of small instances is closer to the ideal and its mean value divides the cumulative probability function to 0.49 and 0.51. However, the distribution function of medium instances differs more than the ideal normal distribution and its mean value divides the cumulative distribution function to 0.44 and 0.56. It is also more discrepant, since one result was 30 seconds more than the mean value. As explained in the previous subsection, one of possible reasons is the task interference, which increases the instability and unpredictability even more. We can conclude that the instance type impacts differently on the performance instability on the same Cloud environment. Additionally, the Kolmogorov-Smirnov (KS) test shows that medium instance fails the test to be a normal distribution, with KS p-value $p = 0.01 < 0.05$, while the small instance complies with a normal distribution with $p = 0.96$, with a confidence of 85.48% according to the Anderson-Darling normality test.

9.4. Application domain

This subsection discusses about the applicability of our approach for various Cloud or workflow types.

9.4.1. Public or private Clouds

Cloud computing is the latest paradigm of distributed systems, and as such, it is trying to be fully transparent to users, i.e. to hide all internal architectures and organisations and all internal administrative and maintenance processes [32]. Therefore, our Cloud noisiness approach consists in observing the Cloud as a black box without going into details

about its architecture and behavior. This is especially emphasised for the commercial Cloud providers that usually try to hide the internal architecture of their Clouds and make it fully transparent to their customers.

In summary, the Cloud noisiness approach is applicable for both, the commercial and private on-site Clouds, with the trade-off of some cost for learning the behavior. Nevertheless, using parameters that are learned with other applications, in different time period, with different heterogeneity factor will probably lead to wrongly simulation. The results show that the Cloud instability mostly depends on the instance type, much less on the number of that instance, and neglectable dependence on the workflow's parallel section size. This fact reduces the experiments necessary in the learning phase, i.e., a user should execute smaller workflows that will finish faster and thus cheaper.

We must note that, still, due to very complex and uncertain Cloud architecture that changes its behavior during the periods of time [14, 15], the accuracy and precision of each model could be slightly reduced. However, by repeating the learning phase users can again improve the accuracy of our model compared to the latest Cloud's behavior.

9.4.2. Workflow types

Workflows can be represented as a directed acyclic graph with three main types of complexities: *computational*, *I/O* and *bandwidth*. The chosen workflow in our experiments consists of various ratios of computational complexity on one side, and I/O and bandwidth on other side. With this, we wanted to have multiplicity of these three complexities in order to cover different types of distributed workflow applications, with various tasks within, such as latency sensitive, I/O and throughput oriented, and computational-intensive.

10. Conclusion and future work

This paper presents a new model and methodology for noising the runtime of a workflow tasks while its execution is simulated in order to behave as the real cloud instable performance environment. The series of experiments and many repetitive test cases show that our model improves the inaccuracy of up to 59 times compared to the basic simulation without adding any noise.

Using our generic model, Cloud providers can offer a Simulation-as-a-Service to their customers after learning how their Clouds behave for specific workflow executions. The noising model and methodology can be implemented in each simulator in order to act more instable and as close as each Cloud. With this realistic simulation, the IT managers can predict the inaccuracy about workflow execution more reliably in order to meet the cost and deadline constraints.

Since the workflow's parallel section does not impact on the makespan's instability, it can be neglected in further experiments in order to save resources, time, energy cost etc. Our sensitivity analysis shows that we can reduce the number of experiments and their repetitions in learning phase. That is, we need only 20% of total number of test cases in order to achieve the inaccuracy up to 12%. This makes our model to be a generic and independent of workflow application type (parallel section size) and instance number that are used.

Our model and methodology improve the simulator's accuracy for almost all instance types and number of instances. The simulation makespan's inaccuracy is up to 11.5% for small instance type, while up to 35% for medium ones. In order to improve the accuracy for simulating instances with multiple vCPUs, such as medium or large instances, we will extend our model and consider additionally file transfers instability. Especially for file transfer intensive workflows, such as Montage [26], this extension could have huge impact on the workflow makespan.

Finally, while other approaches configure the simulator with the values of Cloud parameters, our model sees the Cloud as a black box and configures the simulator considering the values of executions in learning phase. Nevertheless, up to now, all approaches introduce a static noise. According to our best knowledge, no simulator exist that will dynamically change its model during the time, as this is very difficult to achieve. Therefore, we plan to extend our GroudSim simulator to dynamically change its behavior like a real Cloud.

References

[1] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good, On the use of cloud computing for scientific workflows, in: eScience, 2008. eScience '08. IEEE Fourth International Conference on, 2008, pp. 640–645. doi:10.1109/eScience.2008.167.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: A berkeley view of cloud computing, Tech. rep., University of California at Berkeley (February 2009).
URL http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html

[4] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, in: Proc. of the Int. Conf. on HPC, Networking, Storage and Analysis, SC '12, 2012, pp. 1–11.

[5] A. O. Ayodele, J. Rao, T. E. Boult, Performance measurement and interference profiling in multi-tenant clouds, in: Cloud Computing (CLOUD), 2015 IEEE 8th Int. Conf. on, 2015, pp. 941–949. doi:10.1109/CLOUD.2015.128.

[6] E. N. Alkhanak, S. P. Lee, R. Rezaei, R. M. Parizi, Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues, Journal of Systems and Software 113 (2016) 1 – 26.

[7] Open-source simulators for cloud computing: Comparative study and challenging issues, Simulation Modelling Practice and Theory 58, Part 2 (2015) 239 – 254, special issue on Cloud Simulation. doi:http://dx.doi.org/10.1016/j.simpat.2015.06.002.

[8] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, E. ghazali Talbi, Towards understanding uncertainty in cloud computing resource provisioning, Procedia Computer Science 51 (2015) 1772 – 1781, int. Conf. on Computational Science, 2015.

[9] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, The Journal of Supercomputing 71 (9) (2015) 3373–3418. doi:10.1007/s11227-015-1438-4.

[10] A. Basu, S. Fleming, J. Stanier, S. Naicken, I. Wakeman, V. K. Gurbani, The state of peer-to-peer network simulators, ACM Comput. Surv. 45 (4) (2013) 46:1–46:25.

[11] ISO, ISO 5725:1994.
URL https://www.iso.org/obp/ui/#iso:std:11833:en

[12] J. Dejun, G. Pierre, C.-H. Chi, Service-oriented computing. icsoc/servicewave 2009 workshops: International workshops, icsoc/servicewave 2009, stockholm, sweden, november 23-27, 2009, revised selected papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, Ch. EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications, pp. 197–207.

[13] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright, Performance analysis of high performance computing applications on the amazon web services cloud, in: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010, pp. 159–168. doi:10.1109/CloudCom.2010.69.

[14] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz, Runtime measurements in the cloud: Observing, analyzing, and reducing variance, Proc. VLDB Endow. 3 (1-2) (2010) 460–471.

[15] A. Iosup, N. Yigitbasi, D. Epema, On the performance variability of production cloud services, in: Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on, 2011, pp. 104–113. doi:10.1109/CCGrid.2011.22.

[16] R. Buyya, M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and computation: practice and experience 14 (13-15) (2002) 1175–1220.

[17] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[18] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: E-Science (e-Science), 2012 IEEE 8th International Conference on, 2012, pp. 1–8.

[19] B. Wickremasinghe, R. N. Calheiros, R. Buyya, Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications, in: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 2010, pp. 446–452.

[20] W. Depoorter, N. De Moor, K. Vanmechelen, J. Broeckhove, Scalability of grid simulators: An evaluation, in: E. Luque, T. Margalef, D. Benítez (Eds.), Euro-Par 2008 – Parallel Processing: 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 544–553.

[21] B. Donassolo, H. Casanova, A. Legrand, P. Velho, Fast and scalable simulation of volunteer computing systems using simgrid, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, 2010, pp. 605–612.

[22] H. Casanova, A. Legrand, M. Quinson, SimGrid: A generic framework for large-scale distributed experiments, in: Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on, 2008, pp. 126–131.

[23] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, Groudsim: An event-based simulation framework for computational grids and clouds., in: M. R. Guarracino, F. Vivien, J. L. Trff, M. Cannataro, M. Danelutto, A. Hast, F. Perla, A. Knpfer, B. D. Martino, M. Alexander (Eds.), Euro-Par Workshops, Lecture Notes in Computer Science, Springer, pp. 305–313.

[24] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter, Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing 74 (10) (2014) 2899 – 2917.

[25] A. Nuñez, J. L. Vázquez-Poletti, A. C. Caminero, J. Carretero, I. M. Llorente, Design of a New Cloud Computing Simulation Platform, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 582–593.

[26] M. Bux, U. Leser, Dynamiccloudsim: Simulating heterogeneity in computational clouds, Future Generation Computer Systems 46 (2015) 85 – 99.

[27] S. Di, F. Cappello, Gloudsim: Google trace based cloud simulator with virtual machines, Softw. Pract. Exper. 45 (11) (2015) 1571–1590.

[28] D. Poola, S. Garg, R. Buyya, Y. Yang, K. Ramamohanarao, Robust scheduling of scientific workflows with deadline and budget constraints in clouds, in: Advanced Information Networking and Applications, 2014 IEEE Int. Conf. on, 2014, pp. 858–865. doi:10.1109/AINA.2014.105.

[29] S. Ostermann, R. Prodan, T. Fahringer, Extending grids with cloud resource management for scientific computing, in: Grid Computing, 2009 10th IEEE/ACM International Conference on, IEEE, 2009, pp. 42–49.

[30] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, J. Luitz, Wien2k, An augmented plane wave+ local orbitals program for calculating crystal properties.

[31] A. M. Sampaio, J. G. Barbosa, R. Prodan, Piasa: A power and interference aware resource management strategy for heterogeneous workloads in cloud data centers, Simulation Modelling Practice and Theory 57 (2015) 142 – 160. doi:https://doi.org/10.1016/j.simpat.2015.07.002.

URL http://www.sciencedirect.com/science/article/pii/S1569190X15001069
[32] A. S. Tanenbaum, M. Van Steen, Distributed systems, 2nd Edition, Prentice-Hall, 2007.