Simplified Workflow Simulation on Clouds based on Computation and Communication Noisiness

Roland Mathá, Sasko Ristov, Thomas Fahringer, *Member, IEEE*, and Radu Prodan, *Member, IEEE*

Abstract—Many researchers rely on simulations to analyze and validate their researched methods on Cloud infrastructures. However, determining relevant simulation parameters and correctly instantiating them to match the real Cloud performance is a difficult and costly operation, as minor configuration changes can easily generate an unreliable inaccurate simulation result. Using legacy values experimentally determined by other researchers can reduce the configuration costs, but is still inaccurate as the underlying public Clouds and the number of active tenants are highly different and dynamic in time. To overcome these deficiencies, we propose a novel model that simulates the dynamic Cloud performance by introducing noise in the computation and communication tasks, determined by a small set of runtime execution data. Although the estimating method is apparently costly, a comprehensive sensitivity analysis shows that the configuration parameters determined for a certain simulation setup can be used for other simulations too, thereby reducing the tuning cost by up to 82.46 percent, while declining the simulation accuracy by only 1.98 percent on average. Extensive evaluation also shows that our novel model outperforms other state-of-the-art dynamic Cloud simulation models, leading up to 22 percent lower makespan inaccuracy.

Index Terms—Cloud computing, simulation, workflow applications, burstable instances, performance instability and noisiness

1 Introduction

To address complex research problems such as capacity planning, resource management and scheduling in Cloud data centers, researchers typically rely on Cloud simulators that allow faster, more flexible and more efficient coverage of search space parameter configurations without costly real deployment and execution. Unfortunately, the dynamic nature of the underlying Cloud infrastructure leads to unstable performance [1] and to high variations [2], making accurate simulation a very hard problem. To overcome this challenge, state-of-the-art Cloud simulators, such as DynamicCloudSim [3], support dynamic Cloud setup by configuring several parameters, which is challenging even for experienced users and hampers the accuracy of the Cloud simulation because of three deficiencies.

First, commercial Cloud providers neither publish nor ensure all performance parameters related to their Clouds. For example, Amazon does not publish to its customers all underlying hardware details of `t2` virtual machine (VM) instances designed for general purpose use. For such instances, the exact network performance is only indicated in fuzzy terms, such as "low", or even "low to moderate". These terms represent a set of instance type-dependent network performance parameters, such as sustained and burst limits or burst duration limits [4]. A similar case are the CPU resources of the `t2` instances that provide a baseline performance, but have the ability to burst above.

Second, researchers often provide only partial data about the used public Cloud [4]. Even worse, this explored knowledge has often a short lifetime and quickly becomes outdated, as Cloud infrastructures evolve over time [5].

Third, although Schad *et al.* [6] reported several Cloud performance parameters as unstable following a normal distribution, we revealed in [7] that naively using this normal distribution does not provide an accurate simulation.

To overcome these deficiencies that lead to an incomplete or inaccurate Cloud simulation setup, we introduced in previous work [8] a simple approach to determine and configure the Cloud performance instability by noising the task execution times. However, many scientific applications, such as workflows, are data intensive and spend up to 90 percent of their makespan on file transfers [9]. Therefore, we go in this paper a step further by introducing a novel model that simplifies the complex setup of the large set of dynamic configuration parameters to two metrics modelling the *Cloud task execution time (TET) noisiness* and the incoming *Cloud file transfer (FT) noisiness*. Our approach estimates the Cloud performance instability through a small set of training executions, and injects the noisiness affecting both the TET and FT times instead of naively generating values with a normal distribution.

Similar to related work [10], [11], we limit our experiments to workflow executions on various homogeneous VM instances, and analyze the Cloud performance instability related to the number of provisioned instances, instance types, and

- *R. Mathá is with the Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria, and also with the Institute of Information Technology, University of Klagenfurt, 9020 Klagenfurt, Austria. E-mail: roland@dps.uibk.ac.at.*
- *R. Prodan is with the Institute of Information Technology, University of Klagenfurt, 9020 Klagenfurt. E-mail: radu@itec.aau.at.*
- *S. Ristov and T. Fahringer are with the Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria. E-mail: {sashko, tf}@dps.uibk.ac.at.*

Manuscript received 3 Feb. 2019; revised 2 Dec. 2019; accepted 14 Jan. 2020. Date of publication 21 Jan. 2020; date of current version 25 Feb. 2020. (Corresponding author: Roland Mathá.) Recommended for acceptance by R. Yahyapour. Digital Object Identifier no. 10.1109/TPDS.2020.2967662

(different and carefully selected) scientific workflows. The restriction to homogeneous types (apart from being realistic [3]) is the initial step towards future analysis of heterogeneous types. Similar to the state-of-the-art DynamicCloudSim [3] simulator, we measure and analyse the pure Cloud performance related to instance types, while excluding additional Cloud performance instability sources related to different heterogeneous VM instance permutations.

We evaluate our model on an extensive experimental setup with different homogeneous VM sets, varying the number of instances and the instance types in Amazon EC2. We used three real-world scientific workflows with different computation and communication ratios (low and high) and various control- and data-flow communication patterns. We carefully selected the size of each workflow for the instance types and number of VMs to maximise the instability in workflow parallel sections and of the overall makespan. The results show that our model outperforms existing state-of-the-art simulation models in all experiments, reporting up to 22 percent smaller makespan inaccuracy.

Estimating the TET and FT noisiness requires a large set of experiments, which motivated our analysis of reusing their values from other experiments affects the accuracy of the model. A sensitivity analysis exposed that the experimental costs can be significantly reduced by up to 82.46 percent, while worsening the simulation inaccuracy by a negligible 1.98 percent on average. The lower inaccuracy compared to other state-of-the-art simulations is especially emphasized for burstable VM instances (`t2`) in Amazon EC2.

The paper is organized in seven sections. Section 2 formally models workflow applications and Cloud infrastructures, used in Section 3 defining the noisiness process and model. Section 4 presents the extensive evaluation methodology, followed by the experimental results in Section 5. Section 6 discusses the related work and a threat to validity. Section 7 concludes the paper and outlines the future work.

2 Workflow and Cloud Models

This section formally models the workflow and Cloud environment used in our noisiness model.

2.1 Workflow Application Model

We model a *workflow application* W as a directed acyclic graph (DAG) represented as a tuple (T, D), consisting of a set $T = \bigcup_{i=1}^{n}\{T_i\}$ of n *tasks* T_i and a set D of *precedence dependencies*. Two interconnected tasks $T_i, T_j \in T$, where $1 \leq i, j \leq n$, $i \neq j$ can have a control or data dependency between each other. The task precedence dependencies modeled as a set $D = \{(T_i, T_j, D_{ij}) | (T_i, T_j) \in T \times T\}$, where (T_i, T_j, D_{ij}) express that task T_j needs to be started after task T_i finished (control flow) and D_{ij} bytes of data need to be transferred from T_i to T_j over the network (data flow). We assume task scheduling as non-preemptive, i.e., tasks cannot be suspended and resumed after being started.

The function $pred : T \rightarrow \mathcal{P}(T)$, where $\mathcal{P}$ denotes the power set, returns the set of immediate *predecessors* of each task $T_i \in T$ (i.e., $T_j \in pred(T_i) \Leftrightarrow (T_j, T_i, D_{ji}) \in D$), while the function $succ : T \rightarrow \mathcal{P}(T)$ returns the set of immediate *successors* of task T_i (i.e., $T_j \in succ(T_i) \Leftrightarrow (T_i, T_j, D_{ij}) \in D$). Each workflow has a *start task* T_s with no predecessors (i.e., $T_s \in T : pred(T_s) = \emptyset$) and an *end task* T_e with no successors (i.e., $T_e \in T : succ(T_e) = \emptyset$).

Each task T_i has a *requirement vector* R_i, which defines its hardware or software constraints, such as the minimum number of virtual CPUs (vCPU), or the minimum size of memory or storage needed for execution. We express the *computational work* w_i (i.e., work) of each task T_i in million of instructions (MI), and the total incoming data sizes d_i of a task T_i as $d_i = \Sigma_{\forall T_j \in pred(T_i)} D_{ji}$.

Multiple entry and exist tasks, as well as staging of input and output data, can be expressed in this simplified workflow model by introducing new "dummy" start and end tasks with zero computational work.

2.2 Cloud Infrastructure Model

We represent the set of available (provisioned) *VM instances* as $I = \bigcup_{k=1}^{m}\{I_k\}$, where m is constant during the workflow execution and each instance I_k is of the same homogeneous type IT. Since our analysis explores Cloud performance instability related to VM instance types and number of VM instances, we use homogeneous environments to minimize any additional performance instabilities caused by heterogeneous VM types. Nevertheless, in order to generalize our model, we evaluate it with different instance types and number of instances, as presented in Section 5.

The *completion time* $ct(T_i)$ of task T_i is the latest completion time of all its predecessors plus the sum of its *task execution time (TET)* t_i and *incoming files transfer time (FT)* f_i of its total incoming data size d_i:

$$ct(T_i) = \begin{cases} f_i + t_i, & T_i = T_s; \\ \max\limits_{T_p \in pred(T_i)} \Big[ct(T_p)\Big] + f_i + t_i, & T_i \neq T_s. \end{cases} \quad (1)$$

The completion time depends on the instance type and the number of instances, and can differ even in the same environment due to Cloud performance instability [1].

Finally, we express the *workflow makespan* M through the completion time $ct(T_e)$ of end task T_e:

$$M = ct(T_e). \quad (2)$$

Our goal is not to optimize the makespan using a new scheduling algorithm, but to provide an accurate simulation of the makespan for various workflow applications using different instance types and number in various Clouds.

3 Noisiness Model

This section gives first a high-level overview of the process of noising the TET and FT. We further model an experiment and its test case repetitions to estimate the workflow and Cloud performance. Then, we present our model in detail, represented by two Cloud noisiness metrics and their corresponding intermediate workflow noisiness. Finally, we present our model for noising the overall makespan through noising *TET* and *FT* and introduce a metric to quantify its inaccuracy.

3.1 Noising Process

State-of-the-art simulators, such as CloudSim [12] or GroudSim [13], regard the Cloud as a white box and require

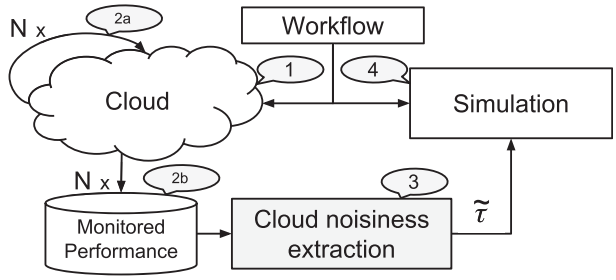


Fig. 1. Noising process of Cloud workflow simulations.

configuring a large set of parameters, which are partially oblivious to the user, difficult to determine, or changing over time [5]. Our novel approach differs from the state-of-the-art simulations and aims to extract the noisiness generated from the Cloud during a workflow's execution, and then to inject this noise into the simulation. We therefore aim to simulate the Cloud by determining its behavior through a small set of execution performance data. The main idea is to extract the Cloud performance instability from a set of repeated workflow executions and to introduce it as Cloud noisiness into the Cloud simulator to approximate its real behavior.

We propose a noising process with four consecutive steps, depicted in Fig. 1. In the first step (1), we repeatedly run the workflow on a Cloud for N times and with the same setup (2*a*), as the Cloud performance is usually uncertain [14] and changes over time [6]. Every workflow execution contains essential information about the Cloud performance instability and stores the results in a database (2*b*). The monitored performance data is used in step (3) that extracts the Cloud performance instability from the workflow execution traces and determines the Cloud noisiness. The fourth step (4) simulates the workflow makespan by adding noise ($\widetilde{\tau}$) to TET and incoming FTs in the simulator.

3.2 Experiment and Test Case Model

We define an *experiment* as a triple (W, IT, m), which denotes that a workflow W is executed on a set of m provisioned VM instances I of type IT. For example, $(\text{BWA}, \texttt{t2.small}, 2)$ denotes that the BWA workflow is executed on two VM instances of type `t2.small`.

To quantify the unstable Cloud behavior, we repeat each experiment N times. We call each execution as *test case* TC_c, which represents the cth repetition of the experiment. We denote the *measured TET* of task T_i of the test case TC_c as $t_{i,c}$, and the *measured incoming FT* as $f_{i,c}$.

3.3 Workflow Noisiness Metrics

We first model the noisiness in two workflow executions (test cases). For this, we introduce the *relative TET difference* $\rho_{cd}(t_i)$ and the *relative FT difference* $\rho_{cd}(f_i)$ of the task $T_i \in T$ for two test cases TC_c and TC_d of the same experiment using a common scaling function $\max$:

$$\rho_{cd}(t_i) = \frac{|t_{i,c} - t_{i,d}|}{\max(t_{i,c}, t_{i,d})}; \quad \rho_{cd}(f_i) = \frac{|f_{i,c} - f_{i,d}|}{\max(f_{i,c}, f_{i,d})}. \tag{3}$$

Using the relative TET difference $\rho_{cd}(t_i)$ and the relative FT difference $\rho_{cd}(f_i)$ for each task T_i, we determine the instability or *workflow noisiness* of workflow W for two test cases TC_c and TC_d of the same experiment. Both deviations introduce the *workflow TET noisiness* $\overline{\Delta}_{TET,cd}$ and the *workflow FT noisiness* $\overline{\Delta}_{FT,cd}$ metrics:

$$\overline{\Delta}_{TET,cd} = \frac{1}{n} \cdot \sum_{i=1}^{n} \rho_{cd}(t_i); \quad \overline{\Delta}_{FT,cd} = \frac{1}{n} \cdot \sum_{i=1}^{n} \rho_{cd}(f_i). \tag{4}$$

Both workflow TET noisiness $\overline{\Delta}_{TET,cd}$ and workflow FT noisiness $\overline{\Delta}_{FT,cd}$ metrics quantify the average noisiness of two repetitions of all n tasks of a single workflow W.

3.4 Cloud Noisiness Metrics

As both workflow noisiness metrics show the instability in workflow TET and FT for two different test cases, we need to define a metric for the Cloud environment instability determined by all test cases of a single experiment (i.e., by each pair of N repeated test cases). Thus, we introduce the Cloud noisiness as the mean workflow noisiness of a set of test cases for an experiment. Analogue to the two workflow noisiness metrics, we introduce two Cloud noisiness metrics, one for the TET and one for FT.

We define the *Cloud TET noisiness* $\overline{\Delta}_{TET}$ as the average workflow TET noisiness $\overline{\Delta}_{TET,cd}$ of each pair TC_c and TC_d from the set of N test cases of the same experiment:

$$\overline{\Delta}_{TET} = \frac{1}{\frac{N \cdot (N-1)}{2}} \cdot \sum_{\forall c,d | 1 \le c < d \le N} \overline{\Delta}_{TET,cd}. \tag{5}$$

Similarly, the *Cloud FT noisiness* $\overline{\Delta}_{FT}$ is the average workflow FT noisiness $\overline{\Delta}_{FT,cd}$ of each pair of test cases TC_c and TC_d from the set of N test cases of the same experiment:

$$\overline{\Delta}_{FT} = \frac{1}{\frac{N \cdot (N-1)}{2}} \cdot \sum_{\forall c,d | 1 \le c < d \le N} \overline{\Delta}_{FT,cd}. \tag{6}$$

The goal is to use the TET and FT Cloud noisiness metrics, i.e., $\overline{\Delta}_{TET}$ and $\overline{\Delta}_{FT}$, instead of several parameters as in other state-of-the-art simulations [3]. The Equations (5) and (6) capture the contribution of each test case of an experiment to the Cloud noisiness metrics.

3.5 Noising Model

After defining both Cloud noisiness metrics, we inject them into a simulation framework to generate the *noised TET* $\widetilde{\tau}(t_i)$ and *noised FT* $\widetilde{\tau}(f_i)$ and simulate the *noised makespan* $\widetilde{M}$.

3.5.1 Noised TET Model

To analyze the distribution of TET per experiment, we define the *TET's mean value* of task T_i:

$$\overline{t}_i = \frac{1}{N} \cdot \sum_{u=1}^{N} t_{i,u}. \tag{7}$$

The *noised TET* $\widetilde{\tau}(t_i)$ of the task T_i is the mean TET value $\overline{t}_i$, noised by the Cloud TET noisiness $\overline{\Delta}_{TET}$ and the Gaussian normal distribution $N(0, \sigma(\overline{\Delta}_{TET}))$ with the standard deviation of the Cloud TET noisiness:

$$\widetilde{\tau}(t_i) = \left[1 + \overline{\Delta}_{TET} + N\left(0, \sigma(\overline{\Delta}_{TET})\right)\right] \cdot \overline{t}_i. \tag{8}$$

Similar to Bux and Leser [3], we also use a normal distribution. However, we introduce two innovations in our noised TET model. First, we shift the mean TET values by the Cloud TET noisiness and add a normal distributed noise. Second, as $\overline{\Delta}_{TET}$ depends on the average workflow TET noisiness that quantifies the average noises over all tasks within a workflow, we inject the standard deviation of $\sigma(\overline{\Delta}_{TET})$ as a noise of shifted $\overline{t}_i$. We reported in previous research [7] the problems of naively using the standard deviation $\sigma(\overline{t}_i)$, as any delay in the execution time on the workflow critical path directly affects the overall makespan. For example, the execution time $\overline{t}_i - \sigma(\overline{t}_i)$ may not affect the overall makespan if the execution time of another task t_j is $\overline{t}_j + \sigma(\overline{t}_j)$ and both run in parallel. Our noised model estimates these effects and inter-task dependencies.

3.5.2 Noised FT Model

To analyze the distribution of FT per experiment, we define the *incoming files transfer time mean value* of task T_i:

$$\overline{f}_i = \frac{1}{N} \cdot \sum_{u=1}^{N} f_{i,u}. \tag{9}$$

Similar to $\widetilde{\tau}(t_i)$, the *noised FT* $\widetilde{\tau}(f_i)$ of the task T_i is the mean FT time $\overline{f}_i$, noised by the Cloud FT noisiness $\overline{\Delta}_{FT}$ and the normal distribution with the standard deviation of the Cloud FT noisiness $N(0, \sigma(\overline{\Delta}_{FT}))$:

$$\widetilde{\tau}(f_i) = \left[1 + \overline{\Delta}_{FT} + N\Big(0, \sigma(\overline{\Delta}_{FT})\Big)\right] \cdot \overline{f}_i. \tag{10}$$

3.5.3 Noised Makespan Model $\widetilde{M}$

We use the noised TET $\widetilde{\tau}(t_i)$ and FT $\widetilde{\tau}(f_i)$ in Equation 1 to compute the *noised completion time* $\widetilde{ct}(T_e)$:

$$\widetilde{ct}(T_i) = \begin{cases} \widetilde{\tau}(f_i) + \widetilde{\tau}(t_i), & T_i = T_s; \\ \max\limits_{T_p \in pred(T_i)} \Big[\widetilde{ct}(T_p)\Big] + \widetilde{\tau}(f_i) + \widetilde{\tau}(t_i), & T_i \neq T_s. \end{cases} \tag{11}$$

From Equations (11) and (2), we define the *noised makespan* $\widetilde{M}$:

$$\widetilde{M} = \widetilde{ct}(T_e) \tag{12}$$

3.6 Noised Makespan Inaccuracy

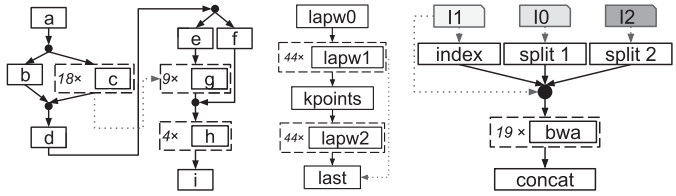
To analyze the distribution of makespans per experiment, we define the *mean makespan*:

$$\overline{M} = \frac{1}{N} \cdot \sum_{u=1}^{N} M_u, \tag{13}$$

where M_u denotes the makespan of test case TC_u, defined in Equation (2) for experiments in a Cloud environment and in Equation (12) in a simulated noised environment.

To quantify the accuracy of the simulations, we define the *inaccuracy* metric $\delta \in [0, 100\%)$ as the relative difference of the *simulated mean makespan* $\overline{M}_{Simulated}$ to the *true measured mean makespan* $\overline{M}_{Measured}$ in an experiment:

$$\delta = \frac{\left|\overline{M}_{Measured} - \overline{M}_{Simulated}\right|}{\overline{M}_{Measured}} \cdot 100. \tag{14}$$



(a) Montage. (b) Wien2k. (c) BWA.

Fig. 2. Schematic structure of the experimental workflows (solid lines show the control flow, dotted lines the data flows, and dashed rectangles the parallel sections size).

4 Testing Methodology

We present in this section the reproducible testing methodology for evaluating our noisiness model from Section 3. The methodology covers three types of workflow applications with high and low computation and communication ratios (Section 4.1), executed in Amazon EC2 Cloud with various types and number of VM instances that generalise our findings. Our methodology not only maximises the testbed diversity (Section 4.2), but also maximises the makespan instability by rigorous selection of amount and types of VMs in each experiment (Section 4.3). We also compare our model with the most common state-of-the-art simulation approaches (Section 4.4).

4.1 Workflow Applications

We carefully selected three scientific workflows with significantly different characteristics (evenly, high and low computation versus communication load) and sources of instability, as described in the following sections.

4.1.1 Montage

Montage [15] is a well known astronomy application, which assembles flexible image transport system images into custom mosaics of the sky. Montage is used in many scientific researches due to its complex structure [3], [16], [17], schematically depicted in Fig. 2a. To simplify the presentation of this complex workflow, Table 1 defines several task name abbreviations. Montage consists of two branches after tasks `a` and `d`, and three parallel sections. The first parallel section consists of a single task `b` and multiple instances of a task `c`, whose number is determined by the input parameter. The second parallel section has two branches: one is a sequence of a task `e` and a parallel section of multiple instances of a task `g`, while the other comprises a single task `f`. The third parallel section consists of multiple instances of the same task `h`. Every instance of the task `c` downloads a single image from an online archive and re-projects it, while instances of task `g` apply corrections to multiple images from instances of task `c` according to the correction table determined by the task `e`. Finally, multiple instances of a task `h` perform a coaddition of images belonging to a tile of the final mosaic, shrunk by a predefined factor. The data flow mostly follows the control flow, such that each successor task waits not only for the control flow, but also for the data from its predecessors. The only exception are instances of task `g`, which receive files from the instances of task `c`, apart from input data from the predecessor task `e`.

TABLE 1
Total Incoming Data Sizes d_i in each Montage, Wien2k, and BWA Workflow Task

Montage			*Wien2k*		*BWA*	
ID	*Task*	d_i [MB]	*Task*	d_i [MB]	*Task*	d_i [MB]
a	RetrieveImageList	0.0	lapw0	1.14	index	204.58
b	CalculateOverlaps	0.003	lapw1	1.06	split1	204.58
c	DownloadAndProject	0.005	kpoints	0.28	split2	204.58
d	CalcDiffFitMulti	143.43	lapw2	1.77	bwa	380.75
e	CalcBackgroundModel	0.015	last	49.09	concat	485.18
f	CalcTiles	0.003				
g	BgCorrectionMulti	143.43				
h	AddAndShrink	143.44				
i	AddTiles	38.74				

TABLE 2
Montage, Wien2k, and BWA diversity: A Parallel Section has Tasks of Same (*Hom.*) or Different (*Het.*) Types

Characteristic		*Montage*	*Wien2k*	*BWA*
Communication vs. computation		$\simeq$	$<$	$>$
Structure	No. of start tasks	1		3
	No. of parallel sections	3	2	
	Parallel section sizes	19,11,4	44, 44	3, 19
	Parallel section types	Het.	Hom.	Het. / Hom.
Data flow	Follows control flow	mainly		
	Exceptions	c to g	lapw1 to last	I1 to index

There are many sources of TET and FT makespan instability in Montage. The former is mainly affected by the instability of the task a, the parallel section of tasks c, the task d, the task e, the parallel section of tasks g, the parallel section of tasks h and the task i. The makespan instability is not directly affected by tasks b and f, since their TETs and FTs are lower than the other concurrent branches (tasks c, e and g). Another source of makespan instability is due to the FTs between the same computational tasks, and the FTs between tasks c and g.

4.1.2 Wien2k

The Wien2k workflow [18] uses a full-potential Linearized Augmented Plane Wave (LAPW) approach for the computation of crystalline solids. The workflow structure consists of two parallel sections (lapw1 and lapw2) of the same size with three synchronization tasks (lapw0, kpoints, and last) in between, as depicted in Fig. 2b. Similar to Montage, the data flow follows the control flow for all tasks. Additionally, the input of task kpoints is also transferred as an input to task last. The size of parallel sections is defined by an input parameter.

Although the structure of Wien2k is simpler than Montage, its makespan is still affected by instability in all TET and, therefore, by all FT too. Wien2k conducts a higher computation versus communication load than to Montage.

4.1.3 BWA

The Burroughs-Wheeler Alignment tool (BWA) [19] is a genomics analysis workflow, which maps low-divergent sequences against a large reference genome, such as the human genome. BWA has a specific structure starting with three parallel tasks (index, split1 and split2), followed by a parallel section of multiple instances of a task bwa, and a final task concat, as depicted in Fig. 2c. The first three tasks load one input file each (i.e., I0, I2 as databases for comparison and I1 as sequences to test), and produce a set of output files scattered among the parallel section of multiple instances of task bwa. Finally, the concat task merges all output files from all instances of task bwa into a single file. Similar to the other workflows, the data flow follows the control flow, except the input file in task index scattered to each bwa task of the parallel section.

Although BWA has a simple structure too, its makespan is affected by the TET of each task, as well as by each FT. As opposed to Wien2k, this workflow generates a higher communication versus computation load.

4.2 Workflow Diversity

We carefully selected the three workflows (Montage, Wien2k and BWA) with different structures, computation and communication loads, to maximise the testbed diversity and generalise our findings. We rigorously selected specific input parameters, which determine the size of the parallel sections and FTs between the tasks. According to our previous research [7], the parallel section size has negligible impact on the makespan instability. We also configured the input parameters for the three workflows according to their performance using two t2.small instances, such that their makespans are of approximately 10min.

Table 2 shows that all three workflows have specific communication and computation characteristics, different structures, and data flows. For instance, Wien2k has lower communication than computation load transferring only a few MB between tasks. In contrast, Montage is a more communication intensive workflow that transfers up to 143.44 MB, while BWA transfers up to 485.18 MB. Table 1 details the total incoming data sizes d_i in MB transferred to the VM instance executing task T_i for all three workflows. For example, the task last of Wien2k collects its required data from all lapw1 and lapw2 tasks, totalling 49.09 MB. Besides the different data sizes, the workflows differ in the number of transferred files. More precisely, Wien2k transfers many smaller files compared to Montage and BWA.

4.3 Experimental Testbed

We carefully selected an extensive Cloud experimental setup and VM instances with different features to generalise our noising model evaluation. To maximise the performance instability, we chose the number of VM instances multiplied by the corresponding number of vCPUs smaller than the size of the largest parallel section in each workflow. This ensures that some of the parallel tasks need to be serialised during execution, increasing their performance instability.

We used the public Amazon EC2 Cloud in Table 3 with two instance families: (i) *burstable* general purpose t2 with unstable network, computing and I/O performance, and (ii) *compute-optimised* c5 with stable compute performance, dedicated elastic block storage and high network bandwidth up to 10 Gbps. We used three t2 instance types (i.e., t2.small with 1 vCPU, t2.medium with 2 vCPUs, and t2.xlarge

TABLE 3
Experimental Amazon EC2 Cloud Instance Types

Instance type	*Performance*	*Baseline [%]*	*vCPU*	*Memory [GiB]*	*Storage [GiB]*	*Network*	*Physical processor*	*Clock [GHz]*
t2.small	burstable	20	1	2	EBS-Only	low to moderate	Intel Xeon family	≤ 3.3
t2.medium	burstable	40	2	4				
t2.xlarge	burstable	90	4	16		moderate		≤ 3.0
c5.large	stable	NA	2	4		≤ 10 Gbps	Intel Xeon Platinum	≤ 3.5

with 4 vCPUs), and one compute optimised instance type c5.large with 2 vCPUs. We selected these instance families to evaluate our model for highly unstable performance, and stable computation with limited communication performance. For example, t2 instances provide variable elastic compute unit (ECU) performance, while c5.large constantly provide 9 ECU.[1] We started from t2.small and avoided t2.micro free for Amazon EC2 users in the first year, possibly biasing the results. To avoid sequential workflow executions on t2.small instances, we started all experiments of each family with two instances. Using horizontal scaling, the number of instances varies between two (2^1), three, four (2^2) and eight (2^3) in each experiment. For t2.xlarge instances, we omit experiments with eight instances to avoid expensive idle instances, as 19 parallel bwa tasks and 18 parallel c tasks can only be inefficiently scattered among $8 \cdot 4 = 32$ vCPUs. Since the maximum size of the parallel section for each workflow varies from 19 for Montage, to 18 for BWA and 44 for Wien2k, we run each experiment with maximum 16 cores to have at least one parallel section with less vCPUs than its size, thereby maximising the makespan instability.

We used the US East Amazon EC2 region with two default Linux images, ami-1ecae776 for the burstable t2 instances and ami-14c5486b for the compute-optimised c5 instances, as they include AWS command line tools, Python, Ruby, Perl, and Java packages. We executed and monitored all workflows using the ASKALON execution engine [20]. We ran 20 test cases of each experiment, according to the sensitivity analysis from our previous work on the minimum number of repetitions required for an accurate simulation [7], leading to a total of 900 test cases.

4.4 Simulation Environment

We compared the $S_{TET/FT}$ model with our previous work S_{TET} [8] (which noises TET only), the standard DynamicCloudSim model S_{DCS} with its standard heterogeneity parameter values, and the real Cloud experimental results (denoted as C). We implemented both S_{TET} and $S_{TET/FT}$ models in DynamicCloudSim besides its default S_{DCS} model. To decouple S_{DCS} from S_{TET} and $S_{TET/FT}$, we deactivated the dynamic characteristics of S_{DCS} during the evaluation of the other two models by setting all its dynamic features to negligible values ϵ (i.e., $\epsilon \to 0$ to avoid "division by zero" exceptions). For a fair comparison, we configured all three simulations with the same network, storage and computation speed settings. We simulated the same experiments and all test cases for each simulation model as in the real Cloud experiments. Since we did not detect any VM failures during experiments, we configured the corresponding parameters to ϵ for S_{DCS} too to avoid nonexistent makespan delays, as reported by Bux and Leser [3] for S_{DCS}. We used the same greedy task queue workflow scheduling for all three simulations, since it outperforms static schedulers in performance instability for S_{DCS} [3].

1. https://aws.amazon.com/en/ec2/pricing/on-demand

5 Evaluation

In this section, we present the experimental results that evaluate the accuracy of our nosing model. First, we draw several important conclusions for the Cloud noisiness metrics. Second, we present the makespan inaccuracy for each simulated experiment and evaluate the impact of specific workflow and Cloud resources. Third, we analyse the network performance instability and present sensitivity analysis results, showing that the number of test cases and experiments can be significantly reduced in the estimating phase with negligible impact on the simulation inaccuracy. Finally, we evaluate the scalability of our model with an enhanced number of t2.small instances.

5.1 Cloud Noisiness Analysis

We first analyse the correlation between the Cloud noisiness metrics and specific workflow applications, instance types, and number of instances. Tables 4, 5, and 6 present the Cloud TET noisiness $\overline{\Delta}_{TET}$ and the Cloud FT noisiness $\overline{\Delta}_{FT}$ with their corresponding standard deviations $\sigma(\overline{\Delta}_{TET})$ and $\sigma(\overline{\Delta}_{FT})$ for the Montage, Wien2k, and BWA workflow experiments with t2.small, t2.medium, t2.xlarge, and c5.large VM instances. Each column presents a specific experiment denoted as the product of the number of instances m and the abbreviation of VM type IT. For example, $2 \times$ TS denotes an experiment with two VM t2.small instances, while $4 \times$ TXL denotes an experiment with four t2.xlarge instances.

The main observation is that the *Cloud FT noisiness* $\overline{\Delta}_{FT}$ is higher than the *Cloud TET noisiness* $\overline{\Delta}_{FT}$, which clarifies its importance. This feature is more emphasised for Montage and BWA, as they transfer larger files.

5.1.1 Montage

Table 4 presents the Cloud TET and FT noisiness metrics for the Montage experiments. For all t2 instances with two up to four instances, we observe a high $\overline{\Delta}_{TET}$ between 0.154 and 0.206. An exception are the $\overline{\Delta}_{TET}$ values of 0.101 and 0.102 for eight t2.small and t2.medium instances, which are burstable. Another reason is the changed VM placement on eight instances to balance the CPU use among physical machines. The $\overline{\Delta}_{FT}$ value is not affected. Section 5.5 presents further details about the scaling performance.

TABLE 4
Cloud Noisiness Metrics and their Standard Deviation Values (σ) for the Montage Workflow

	2×TS	3×TS	4×TS	8×TS	2×TM	3×TM	4×TM	8×TM	2×TXL	3×TXL	4×TXL	2×CL	3×CL	4×CL	8×CL
$\overline{\Delta}_{TET}$	0.154	0.188	0.158	**0.101**	0.193	0.206	0.166	**0.102**	0.179	0.197	0.187	0.212	0.195	0.227	0.185
$\sigma(\overline{\Delta}_{TET})$	0.024	0.029	0.029	**0.020**	0.033	0.034	0.025	**0.021**	0.030	0.031	0.028	0.033	0.028	0.032	0.026
$\overline{\Delta}_{FT}$	0.154	0.166	0.158	0.158	0.181	0.252	0.213	0.207	0.216	0.261	0.253	0.229	0.224	0.261	0.290
$\sigma(\overline{\Delta}_{FT})$	0.045	0.033	0.027	0.041	0.030	0.055	0.047	0.033	0.028	0.060	0.070	0.037	0.029	0.049	0.059

TABLE 5
Cloud Noisiness Metrics and their Standard Deviation (σ) for the Wien2k Workflow

	2×TS	3×TS	4×TS	8×TS	2×TM	3×TM	4×TM	8×TM	2×TXL	3×TXL	4×TXL	2×CL	3×CL	4×CL	8×CL
$\overline{\Delta}_{TET}$	0.092	0.121	0.171	**0.089**	0.124	0.151	0.169	**0.132**	0.174	0.188	0.192	0.146	0.192	0.216	0.184
$\sigma(\overline{\Delta}_{TET})$	0.031	0.045	0.107	**0.010**	0.013	0.013	0.015	**0.012**	0.015	0.014	0.014	0.013	0.015	0.016	0.014
$\overline{\Delta}_{FT}$	0.134	0.168	0.181	0.201	0.165	0.193	0.224	0.231	0.225	0.232	0.241	0.169	0.187	0.201	0.245
$\sigma(\overline{\Delta}_{FT})$	0.011	0.012	0.017	0.023	0.014	0.017	0.022	0.025	0.019	0.017	0.026	0.023	0.014	0.014	0.030

TABLE 6
Cloud Noisiness Metrics and their Standard Deviation (σ) for the BWA Workflow

	2×TS	3×TS	4×TS	8×TS	2×TM	3×TM	4×TM	8×TM	2×TXL	3×TXL	4×TXL	2×CL	3×CL	4×CL	8×CL
$\overline{\Delta}_{TET}$	0.080	0.087	0.091	0.106	0.102	0.106	0.111	0.114	0.121	0.131	0.125	0.128	0.131	0.131	0.137
$\sigma(\overline{\Delta}_{TET})$	0.022	0.020	0.024	0.022	0.019	0.021	0.022	0.022	0.021	0.028	0.028	0.020	0.019	0.019	0.027

For c5 instances, $\overline{\Delta}_{TET}$ remains on similar high level from 0.185 up to 0.227. The $\overline{\Delta}_{FT}$ values show an increasing tendency with the number of instances, up to an overall maximum value of 0.290, due to the limited network bandwidth of 10 Gbps with undefined baseline performance. Section 5.3 presents an analysis of the network performance instability for various system setups and file sizes.

5.1.2 Wien2k

Table 5 lists the Cloud TET and FT noisiness metrics for the Wien2k experiments. We observe a correlation between both Cloud noisiness metrics ($\overline{\Delta}_{TET}$ and $\overline{\Delta}_{FT}$), and the number of instances for each instance type. Using more VM instances increases the Cloud noisiness metrics. Similar to Montage, we observe two outliers for $8 \times$ TS and $8 \times$ TM instances, where the Cloud noisiness $\overline{\Delta}_{TET}$ becomes comparably small again, as for the corresponding $2 \times$ TS and $2 \times$ TM experiments. This occurs because t2 instances are burstable, or the VM placement changes on eight instances.

We identify a further indication for the burstable VM instance in $\sigma(\overline{\Delta}_{TET})$, which is unstable (up to 0.107) for t2.small, and more stable (up to 0.015) for t2.medium and t2.xlarge instances. In contrast, $\sigma(\overline{\Delta}_{FT})$ is more stable regardless of the instance types, with values of up to 0.026. As expected, the standard deviation $\sigma(\overline{\Delta}_{FT})$ slightly increases for more VM instances, since the network communication increases between VMs across different physical servers compared to the intra-server communication. The experimental results with c5 instances show similar high $\overline{\Delta}_{TET}$ values as for the t2.xlarge instances, while the $\overline{\Delta}_{FT}$ values follow the t2.medium instance behavior.

5.1.3 BWA

Table 6 shows the Cloud noisiness $\overline{\Delta}_{TET}$ and $\overline{\Delta}_{FT}$ with their standard deviations for the BWA experiments with t2.small, t2.medium, t2.xlarge, and c5.large VM instances. We observe that $\overline{\Delta}_{TET}$ slightly increases with the number of t2 instances, as BWA has lower computation load and utilises the CPUs less. Additionally, the performance peaks are small enough to be compensated by the t2 bursts. All t2 instances show high Cloud FT noisiness $\overline{\Delta}_{FT}$ between 0.210 and 0.292, and a high standard deviation $\sigma(\overline{\Delta}_{FT})$ between 0.030 and 0.100. We expected such high values since BWA utilises the network bandwidth more intensively than Wien2k or Montage, which exhausts the burstable network offered by the t2 instance types.

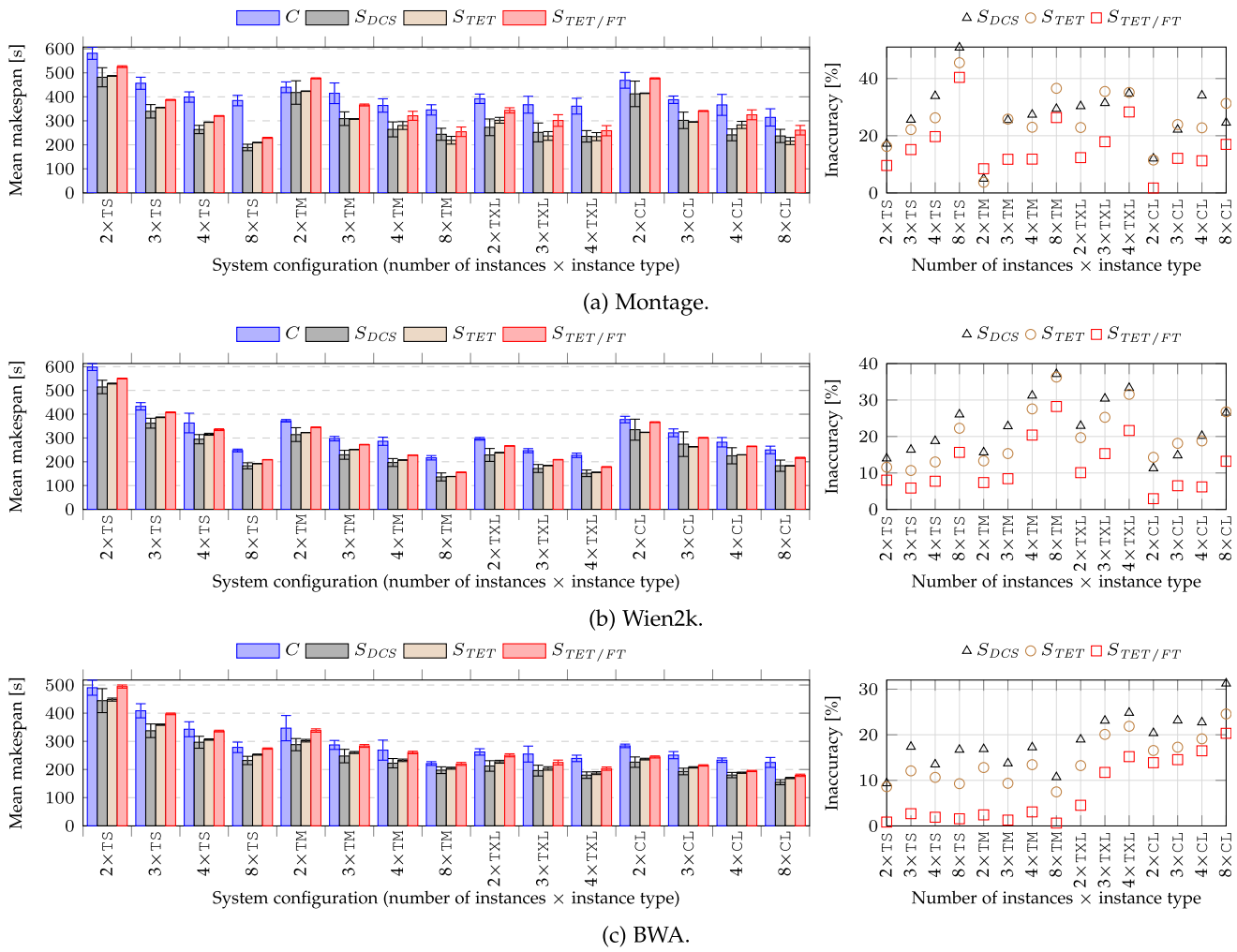
For the compute-optimized c5.large instances, we notice an expected stable and constant $\overline{\Delta}_{TET}$. In contrast, $\overline{\Delta}_{FT}$ and $\sigma(\overline{\Delta}_{FT})$ increase with the number c5.large instances, which matches the Wien2k and Montage results. The reason for the increased $\overline{\Delta}_{FT}$ and $\sigma(\overline{\Delta}_{FT})$ is the ambiguous baseline network performance of the c5 instances, of "up to 10Gbps" (see Table 3). The standard deviation $\sigma(\overline{\Delta}_{TET})$ is stable and congruent with our observation with values of up to 0.028 over all VM instance types.

5.2 Simulation Accuracy Analysis

We analyse in this section the simulation accuracy for each workflow and each instance type separately.

5.2.1 Workflow Simulation Accuracy

Fig. 3 depicts the mean makespan along with the standard deviations for the three workflows. The X-axis denotes



(a) Montage.

(b) Wien2k.

(c) BWA.

Fig. 3. Mean makespan and its standard deviation (left, where C denotes the Cloud) and the corresponding makespan inaccuracies δ relative to the Cloud makespan (right) for different workflows and instance types.

different experiments as the product between the number of instances m and the VM instance type abbreviation.

Montage: We observe in Fig. 3a that $S_{TET/FT}$ achieves in almost all experiments the smallest makespan inaccuracy from 1.73 percent up to 28.31 percent, while for eight `t2.small` instances the inaccuracy has a peak of 40.40 percent. Comparing the real Cloud makespans C for $4 \times$ `TS` and $8 \times$ `TS`, doubling the number of instances improves the makespan by only 15s and the speedup stagnates due to the FT times of the `g` tasks. While the FT times of other the Montage tasks are negligibly affected by the number of instances, the FT time of `g` tasks is 17 s slower for $8 \times$ `TS` than for $4 \times$ `TS` and almost double compared to $2 \times$ `TS`. We explain this through the limited network bandwidth of `TS` instances, neglected by all simulation models that simulate higher bandwidth per VM and consequently, lower overall average makespan. The S_{TET} model obtains values from 3.74 percent up to 45.52 percent and S_{DCS} from 4.98 percent even up to 50.84 percent. Especially for experiments with `c5.large` instances, $S_{TET/FT}$ shows reduced makespan inaccuracy between 1.73 and 16.98 percent. Comparing the differences in inaccuracies per experiment, we notice that $S_{TET/FT}$ has between 4.71 percent (i.e., $8.45\% - 3.74\%$) and 17.56 percent smaller makespan inaccuracy than S_{TET}, and between 3.46 and 22.81 percent lower inaccuracy than S_{DCS}. The only experiment where $S_{TET/FT}$ reported higher but still comparable simulation inaccuracy is for $2 \times$ `TM` explained by the incoming FTs of the `g` tasks, which are on average at least 10s faster for $2 \times$ `TM` than for a higher number of `TM` instances.

Wien2k: Fig. 3b shows that $S_{TET/FT}$ exhibits in all experiments the smallest makespan inaccuracy between 2.95 and 28.22 percent, while S_{TET} achieves values from 10.68 percent up to 36.29 percent, and S_{DCS} from 11.29 percent up to 37.13 percent. Especially on `c5.large` instances, $S_{TET/FT}$ shows the lowest makespan inaccuracy with the lowest value of 2.95 percent for two instances and the highest value of 13.21 percent for eight instances. This is explained by the stable computing performance of `c5` instances and the increasing in makespan inaccuracy with $\overline{\Delta}_{FT}$, as presented in Table 3b. Comparing the differences in makespan inaccuracies per experiment, $S_{TET/FT}$ shows between 3.56 and 13.55 percent (i.e., $26.76\% - 13.21\%$) smaller inaccuracy than S_{TET}, and between 5.95 and 15.02 percent smaller inaccuracy than S_{DCS}.

BWA: Similar to the other two workflows, Fig. 3 c confirms that $S_{TET/FT}$ obtains the smallest makespan inaccuracy for all experiments, from negligible 0.63 percent up to 20.32 percent. Similarly, S_{TET} achieves from 7.47 percent up to 24.56 percent, and S_{DCS} from 9.36 percent even up to 31.25 percent. Especially for experiments with `t2.small` and `t2.medium` instances, $S_{TET/FT}$ reduced the makespan inaccuracy to almost

negligible values between 0.85 and 2.69 percent for `t2.small`, and between 0.63 and 3.08 percent for `t2.medium`. Comparing the inaccuracy per experiment, $S_{TET/FT}$ has between $2.63\% - 10.37\%$ lower makespan inaccuracy than S_{TET}, and between $6.24\% - 15.15\%$ lower inaccuracy than S_{DCS}.

Summary: Since $S_{TET/FT}$ introduces communication noise besides computation, it is more accurate than S_{TET} by 7.02 percent for BWA, 8.46 percent for Wien2k, and 9.23 percent for Montage workflows. Although it generates high instability in networking, $S_{TET/FT}$ achieved the highest improvement for Montage, explained by the incoming FTs of the `g` tasks. Introducing the instability in simulated FT improves the simulation accuracy. Section 5.3 elaborates in detail how various file sizes and instance types affect the network performance. We can also correlate the accuracy of $S_{TET/FT}$ with the specific workflow characteristics and instance types. While $S_{TET/FT}$ is evenly accurate for Montage across all instance types, it generates higher accuracy for Wien2k (with higher computation) on `c5.large` instances (compute optimised) and for BWA (with higher communication) on `t2` instances (computing unstable, assigning only fraction of the cycles to a VM).

5.2.2 Instance Type Simulation Accuracy

In this section, we analyse the impact of Cloud instance types on simulation inaccuracy. For `t2.small` instances, we observe that $S_{TET/FT}$ achieves values between 6.16 percent ($2 \times$ `TS`) and 19.21 percent ($8 \times$ `TS`), S_{TET} between 12.15 and 25.68 percent, and S_{DCS} between 13.52 and 31.20 percent. Our $S_{TET/FT}$ model shows an average inaccuracy of 10.76 percent, which is 6.60 percent more accurate than S_{TET}, and 10.87 percent better than S_{DCS}. We obtained similar inaccuracies for `t2.medium` instances as for `t2.small`, with values from 6.08 percent ($2 \times$ `TM`) up to 18.40 percent ($8 \times$ `TM`) for $S_{TET/FT}$, compared to $9.96\% - 26.78\%$ for S_{TET}, and $12.50\% - 25.74\%$ for S_{DCS}. The average inaccuracies are also similar, 10.85 percent for $S_{TET/FT}$, which is 7.87 percent less than S_{TET}, and 10.20 percent less inaccurate than S_{DCS}, explained by their similar properties (see Table 3). For experiments with `t2.xlarge` instances, all three models achieve higher inaccuracy between 8.99 percent ($2 \times$ `TXL`) and 21.72 percent ($4 \times$ `TXL`) for $S_{TET/FT}$, between 18.60 and 29.52 percent for S_{TET}, and between 24.07 and 30.98 percent for S_{DCS}. The average inaccuracy for $S_{TET/FT}$ is 15.23 percent, which is 9.78 percent better than S_{TET} and 12.55 percent better than S_{DCS}. For the `c5.large` experiments, $S_{TET/FT}$ obtains inaccuracies between 6.18 percent ($2 \times$ `CL`) and 16.84 percent ($8 \times$ `CL`), compared to $14.12\% - 27.54\%$ for S_{TET}, and $14.57\% - 27.46\%$ for S_{DCS}. The average makespan inaccuracy of $S_{TET/FT}$ is 11.34 percent, which is 9.07 percent better than S_{TET} and 10.58 percent better than S_{DCS}.

We conclude that $S_{TET/FT}$ is more accurate among all instance types. Comparing the average makespan inaccuracies per instance type independent from the number of instances and models, `t2.small` and `t2.medium` report the smallest and similar average inaccuracies, shortly before `c5.large` and followed by `t2.xlarge`. All simulation models generate the highest inaccuracy for `t2.xlarge` because it has the highest number of vCPUs (four). The first-come-first-served scheduling policy used does not aim to minimise the traffic between VMs, which generates a higher instability in experiments with the higher number of instances comprising more vCPUs. We achieved the lowest inaccuracy for $2 \times$ `TS`, and the highest inaccuracy for $4 \times$ `TS`.

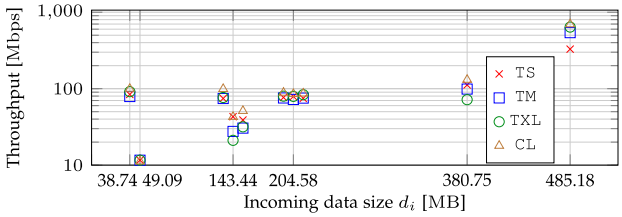


Fig. 4. Average FT throughput per instance type for Montage, Wien2k, and BWA workflows.

5.3 Network Performance Variability

We observe in Fig. 3 that the simulation models underestimate the Cloud makespan in almost all experiments. We identify two reasons for this behavior: the performance variability of Cloud networking and improper simulation support in DynamicCloudSim for network variability.

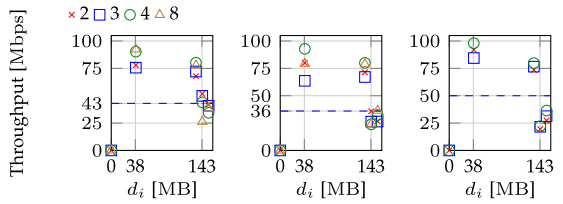
5.3.1 Cloud Network Variability

To analyse the network variability in real Clouds, we measured the throughput for the tasks in Table 1. Fig. 4 shows the dispersed average throughput for each instance type and total incoming data sizes d_i (introduced in Section 2.1) larger than 2 MB, affected by: (i) the incoming data sizes, (ii) the network bandwidth for a specific instance type, and (iii) the number of instances associated to tasks and VM schedules.

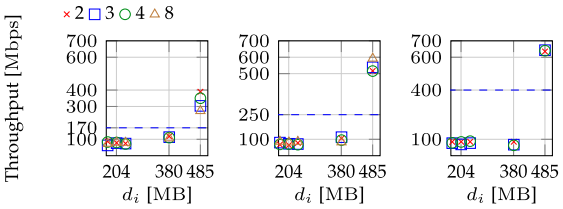
Impact of the incoming data size: Fig. 4 shows three important observations considering the incoming data size d_i. First, each instance type achieves different average throughput when transferring data of different size to a successor task. For example, `c5.large` reported the highest discrepancy for the average throughput, starting from 11.2 Mbps for a data size of 49.09 MB (Wien2k), up to 696.6 Mbps for transferring 485.18 MB (BWA). Second, each instance type achieves different average throughput when transferring the same incoming data size d_i. For example, Montage transfers 143.4 MB in 36×4 MB files to one (`d`), nine (`g`), and four (`h`) tasks with a throughput of 74 Mbps to 99 Mbps, 21 Mbps to 43 Mbps, respectively 30 Mbps to 51 Mbps. Finally, we do not identify any file size related pattern. For example, the `TS` instances show an average throughput of 84.1 Mbps for 38.74 MB, 11.6 Mbps for 49.09 MB, while 111.9 Mbps for 380.75 MB (again high).

We conclude that the Cloud offers a variable throughput for a VM instance. The FT time is not a simple linear function of the data size d_i, but various (even the same) data generate different throughput depending both on the total size and number of files.

Impact of the instance type: Fig. 4 shows that instance types do not follow their declared network performance for the same incoming data size. For example, in almost all experiments, `t2` instances do not follow their network performance definitions, i.e., `TXL` should achieve higher throughput than `TM` and `TS`. But, only for FTs of 485.18 MB (19×25.5 MB to one task), we detect `TXL` instances being the fastest, followed by `TM` and `TS` instances. In contrast, `c5.large` instances achieve in almost all experiments the highest throughput as they are equipped with 10 Gbps network interface cards. However, even if AWS declares the lowest network bandwidth



(a) Montage with TS (left), TM (center), and TXL (right), where $d_i = 0$ denotes incoming data size of 3 kB to 15 kB.



(b) BWA with TS (left), TM (center), and TXL (right).

Fig. 5. Average throughput per number of t2 instances for Montage and BWA workflows for various incoming data size d_i. The dashed horizontal lines show the bandwidth per instance type configured in DynamicCloudSim.

with "low to moderate" for TS instances, TS achieved even the maximum throughput of 43 Mbps among all instance types for incoming file transfers of Montage's g tasks.

We conclude, that the specification about the network is unreliable and its performance does not follow the instance type specification. Moreover, instance types specified with smaller network bandwidth (TS) could achieve higher throughput than more powerful instances (e.g., TXL or CL), even for the same incoming data size.

Impact of the number of instances: Besides incoming data size and instance type, we analyze in Fig. 5 the impact of the number of instances on the throughput for the Montage and BWA communication intensive workflows.

First, we observe that different number of instances achieve various network throughput for the same instance type and workflow. For example, Fig. 5a shows that an incoming data size of 38.74 MB with $4 \times$ TS and $8 \times$ TS shows similar throughput of 90.30 Mbps to 93.44 Mbps, while $2 \times$ TS and $3 \times$ TS show only 75.65 Mbps to 77.9 Mbps. For the same incoming data size, we observe another pattern for TM. Namely, $2 \times$ TM and $8 \times$ TM shows almost identical throughput with 80.08 Mbps and 79.32Mbps, while $3 \times$ TM is the slowest with 63.61 Mbps and $4 \times$ TM is the fastest with 92.86 Mbps.

Second, we observe that the same instance type, number of instances, and workflow achieve different performance for specific incoming data sizes. For instance in Fig. 5a, $8 \times$ TS achieves throughput from 26.30 Mbps to 77.69 Mbps for 143.44 MB, while in Fig. 5b, $2 \times$ TS achieves a throughput from 74.46 Mbps to 117.76 Mbps for 204.58 MB.

Summary: The analysis for Cloud network performance variability reveals that various incoming data sizes generate different throughput for experiments with same instance type, number of instances, and task of the same workflow. As a consequence, the network bandwidth should be individually configured for each task in each experiment. Moreover, the Cloud network performance does not follow the instance type specification and cannot be defined as a simple linear function of the incoming data size. There is no pattern for network performance for specific incoming data size, number of files, instance type, number of instances and the workflow.

5.3.2 Network Simulation Challenges

Besides the network variability, the proper network configuration in a simulator is challenging due to several reasons. First, the specifications of public Cloud providers are vague, such as "≤10 Gbps" or "low to moderated". For example, while c5.large instances in Fig. 4 provide the highest performance for almost all workflows regardless of incoming data size, their network performance is still far lower than the specified upper limit of 10 Gbps. On the other side, the TS instances with "low to moderate" network may outperform other, more powerful instances, as in the Wien2k experiments with a file size of 49.09 MB.

Second, the network is a well-known bottleneck in Cloud environments with a FT variation of up to 65 percent in Amazon EC2, as reported by Jackson *et al.* [21].

Third, many simulators rely on flow-level or simplistic packet-level network models to simulate FTs. Velho *et al.* [22] showed a few invalidation experiments for four well-known simulators including CloudSim and GroudSim. Based on this, Casanova *et al.* [23] identified two reasons for discrepancies between a simulation and real Cloud measurements using the WorkflowSim example. First, WorflowSim uses the simplistic network model from CloudSim that leads to simulation bias related to FT times. Second, WorkflowSim does not capture all relevant details of the system and the execution. Both reasons hold for DynamicCloudSim, since its network model is also based on CloudSim.

5.3.3 Cloud Networking Overestimation

Unfortunately, DynamicCloudSim supports the setup of only one network bandwidth value per instance type. Following the Cloud provider network specification, we assume that the same instance types achieve the same or at least comparable performance. Nevertheless, we approximated the network performance as the average throughput of all tasks per workflow and instance type. We observe in Fig. 5 that our determined values are well-suited tradeoffs between the slowest and the fastest FTs, but the throughput is sensitive to the number of instances. For instance, the configured simulation network bandwidth (the dashed lines in Fig. 5) for Montage TM instances (36 Mbps) is lower than for TS instances (43 Mbps), while for BWA is the other way around (i.e., 250 Mbps for TM versus 170 Mbps for TS), as shown in Fig. 5. On the other side, both TM and TS have the same "low to moderate" network specification. This means that some simulation experiments capture slower FT due to network performance underestimation, caused by a lower average simulation bandwidth configuration. Similarly for smaller files, we capture faster FT due to network performance overestimation. As extreme example, we consider the file sizes of 3 kB to 15 kB, grouped as 0 in Fig. 5a, to show the possible effect of selecting only one tradeoff network bandwidth. For such files, the measured Cloud throughput ranges from 5.5 Kbps to 12.8 Kbps with FT times of 1.62 s to 17.92 s. In contrast, the approximated network performance for Montage and TS is 36 Mbps, which leads to overestimated FT times of only 0.66 ms to 3.33 ms. The extent of such misestimations on the overall simulated makespan depends on the workflow structure and its critical path. For example, Figs. 5a and 3a overestimate the average throughput of tasks g and h of Montage on $8 \times$ TS instances to 16.93 s and 3.27 s. Since both task types are within parallel sections, this

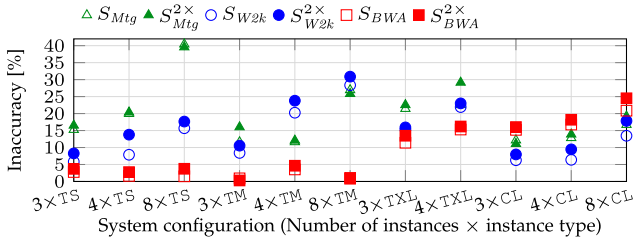


Fig. 6. Makespan inaccuracy δ for Montage (*Mtg*), Wien2k (*W2k*), and BWA by using proper (denoted as S) and exchanged (denoted as $S^{2\times}$) Cloud noisiness values.

overestimation is considered twice for g tasks (9 tasks divided by $8 \times$ TS) and once for h tasks (4 tasks divided by $8 \times$ TS).

In summary, although we carefully configured the network performance per instance type and workflow, we still do not achieve accurate results due to Cloud network performance variations, vague specifications, and simulation model discrepancies. We conclude that a simple VM bandwidth configuration is inadequate, but this goes beyond our current scope and will be part of the future work.

5.4 Sensitivity Analysis

To reduce the number of experiments and costs charged by the public Cloud provider, we conduct two sensitivity analyses that investigate the tradeoff in simulation accuracy by reusing Cloud noisiness values determined in other experiments: *resource sensitivity* and *workflow sensitivity*.

5.4.1 Resource Sensitivity

Resource sensitivity investigates how to conduct test cases of one experiment with two instances of a specific type, using the Cloud noisiness $\overline{\Delta}_{TET}$ and $\overline{\Delta}_{FT}$ determined in other experiments of the same workflow with more instances of the same type.

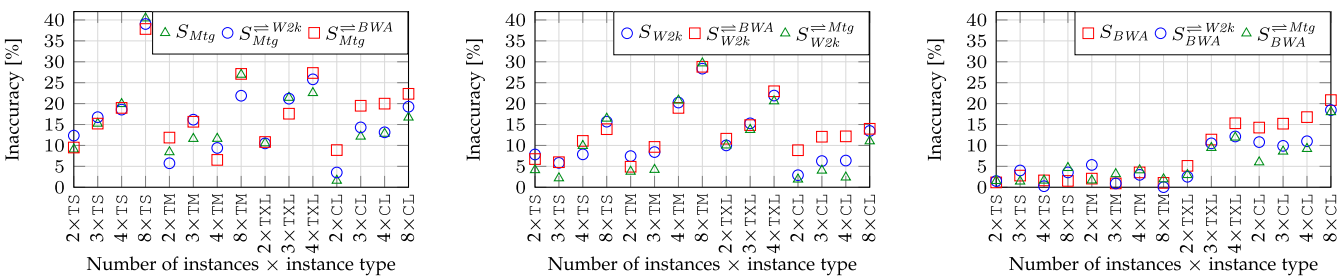
Fig. 6 depicts the makespan inaccuracy of the Montage, Wien2k, and BWA workflows. The simulations S_{W2k}, S_{BWA}, S_{Mtg} use the Cloud noisiness values displayed in Tables 4, 5, and 6. The $S^{2\times}_{Mtg}$, $S^{2\times}_{W2k}$, and $S^{2\times}_{BWA}$ simulations use the Cloud noisiness of the experiments with two instances of the same instance type. All experiments on `t2.small`, `t2.medium`, `t2.xlarge` and `c5.large` instances use a Cloud noisiness of $2 \times$ TS, $2 \times$ TM, $2 \times$ TCL, respectively $2 \times$ CL for the same workflow. Thus, Fig. 6 omits the results on two instances of any type. The results for Montage show that S_{Mtg} and $S^{2\times}_{Mtg}$ have similar inaccuracies for each experiment, starting from inaccuracy differences of 0.44 percent for $4 \times$ TS up to 6.59 percent for $4 \times$ TXL, with an average difference of 1.89 percent. The results for Wien2k show that S_{W2k} and $S^{2\times}_{W2k}$ have similar inaccuracies for each experiment, starting from inaccuracy differences of 0.64 percent for $3 \times$ TXL up to only 5.91 percent for $4 \times$ TS, with an average difference of 2.67 percent. The BWA results show even smaller inaccuracy differences for each experiment, starting from negligible inaccuracy differences of 0.24 percent for $8 \times$ TM up to only 3.67 percent for $8 \times$ CL, with an average difference of 1.37 percent. We expected these results, as Table 6 shows similar Cloud noisiness values and deviations per instance type. We also observe that $S^{2\times}_{BWA}$ leads in some experiments to smaller makespan inaccuracy than S_{BWA}, with precisely determined Cloud noisiness. This occurs in experiments $3 \times$ TM and $8 \times$ TM, where the inaccuracy of S_{BWA} is negligible (lower than 1 percent).

This analysis shows that our model has a huge cost saving potential of 82.46 percent due to 73.33 percent less experiments to learn the values of Cloud noisiness metrics, while declining the simulation inaccuracy by 1.98 percent in average.

5.4.2 Workflow Sensitivity

The workflow sensitivity investigates the increase in simulation inaccuracy by using the Cloud noisiness metrics $\overline{\Delta}_{TET}$ and $\overline{\Delta}_{FT}$ determined in an experiment with another workflow on the same resources. Fig. 7 shows the makespan inaccuracies of the Montage (Mtg), Wien2k ($W2k$), and BWA (BWA) workflows, where S_{Mtg}, S_{W2k}, and S_{BWA} denote again the simulations with the correct Cloud noisiness for the three workflows, while $S^{\rightleftharpoons b}_{a}$ denotes the simulation of a workflow a with the exchanged Cloud noisiness of workflow b. For example, $S^{\rightleftharpoons BWA}_{W2k}$ uses the Cloud noisiness of the BWA workflow (i.e., $\overline{\Delta}_{TET} = 0.114$ and $\overline{\Delta}_{FT} = 0.227$ from Table 6) for $8 \times$ TM experiments, instead of the Wien2k one (i.e., $\overline{\Delta}_{TET} = 0.132$ and $\overline{\Delta}_{FT} = 0.231$ from Table 5).

Fig. 7a depicts the makespan simulation inaccuracies of Montage, where $S^{\rightleftharpoons W2k}_{Mtg}$ reports values from 0.06 percent for $2 \times$ TXL up to 5.13 percent for $2 \times$ CL, with an average difference of 2.38 percent. The $S^{\rightleftharpoons BWA}_{Mtg}$ shows values from 0.02 percent for $3 \times$ TS up to 10.49 percent for $2 \times$ CL, with an average difference of 3.75 percent. For the Wien2k workflow analysed in Fig. 7b, $S^{\rightleftharpoons BWA}_{W2k}$ reports inaccuracies between 0.16 percent for $3 \times$ TS and 5.98 percent for $2 \times$ CL, with an average difference of only 2.19 percent. Similarly, $S^{\rightleftharpoons Mtg}_{W2k}$ shows inaccuracies in the range from 0.02 percent for $2 \times$ TXL up to 4.76 percent for $2 \times$ CL, with an average difference of 2.42 percent. We observe similar results for the BWA workflow presented in Fig. 7c, with $S^{\rightleftharpoons W2k}_{BWA}$ from 0.02 percent



(a) Montage. (b) Wien2k. (c) BWA.

Fig. 7. Workflow sensitivity of makespan inaccuracy δ for Montage (*Mtg*), Wien2k (*W2k*), and BWA using proper (denoted as S) and exchanged (denoted as $S^{\rightleftharpoons}$) Cloud noisiness values.

for $3 \times$ TM up to 5.77 percent for $4 \times$ CL, with an average difference of 2.44 percent. Finally, $S^{\rightleftharpoons Mtg}_{BWA}$ achieves values from 0.005 percent for $4 \times$ TS up to 8.28 percent for $2 \times$ CL, with an average difference of 2.91 percent.

In summary, we expected the results with t2 instances for the exchanged Wien2k and BWA values, since both workflows have different computation to communication ratios, making BWA with higher communication load more stable against changes in $\overline{\Delta}_{TET}$, but sensitive on changes in $\overline{\Delta}_{FT}$. Wien2k, however, exhibits the opposite behaviour. Considering the $2\times$TM experiments, we observe a decrease in inaccuracy for $S^{\rightleftharpoons BWA}_{W2k}$ and an increase for $S^{\rightleftharpoons W2k}_{BWA}$. Comparing the corresponding values in Tables 5 and 6, we notice a positive difference of 21.56 percent for $\overline{\Delta}_{TET}$, and a negative difference of 57.57 percent for $\overline{\Delta}_{FT}$. Consequently, the inaccuracy increases for the BWA workflow sensitive to changes in $\overline{\Delta}_{FT}$. For Wien2k, the high difference in $\overline{\Delta}_{FT}$ compensates the decrease in $\overline{\Delta}_{TET}$ and achieves an even lower inaccuracy. We observe similar results for Montage with its intensive communication and complex structure, where both $S^{\rightleftharpoons W2k}_{Mtg}$ and $S^{\rightleftharpoons BWA}_{Mtg}$ report the same average difference of 2.33 percent for t2 instances.

Overall, the experimental results for CL instance types are different than most t2 instance types. For instance, even if the $\overline{\Delta}_{TET}$ results for CL show comparatively higher values per workflow than TXL, the inaccuracy is sensitive to the $\overline{\Delta}_{FT}$ values due to the different data communication sizes. Nevertheless, we notice that Wien2k and Montage report similarly high $\overline{\Delta}_{FT}$ values, higher than BWA. Thus, the exchanged results $S^{\rightleftharpoons W2k}_{Mtg}$ and $S^{\rightleftharpoons Mtg}_{W2k}$ achieve average differences of 2.54 and 3.36 percent, respectively.

This workflow sensitivity analysis shows that our model provides lower inaccuracy for t2 than for CL VM instance types for the tested workflows. Since all three workflows have similar makespans and Amazon charges its instances on an hourly basis, we were able to save costs, number of experiments and time for using t2 instances by 66.6 percent, while reducing simulation accuracy by 1.92 percent in average.

5.5 Scalability Analysis

To evaluate the accuracy of our model at a higher scale, we designed a sample testbed with 64 t2.small instances and increased the size of the parallel sections of Montage (to $36\times$c, $18\times$g, $9\times$h), Wien2k (to $146\times$lapw1, $146\times$lapw2), and BWA (to $99\times$bwa) to obtain comparable makespans of approximately 12min (denoted as Montage-36, Wien2k-146, and BWA-99). We selected the t2.small instances to minimise the overall costs.

Fig. 8 shows the mean makespan and standard deviation of the scaled workflow executions. Similar to the previous results at a lower scale, $S_{TET/FT}$ achieves the smallest makespan inaccuracy for all three workflows between $1.19\% - 9.73\%$, while S_{TET} achieves between $16.58\% - 21.47\%$, and S_{DCS} between $4.68\% - 26.47\%$. Despite the higher deviation in the makespan for all three workflows, $S_{TET/FT}$ is appropriate for higher scale experiments with t2.small instances achieving even lower inaccuracy on average than at lower scale. In contrast to our experimental results in Section 5.2, the communication intensive workflow BWA-99 obtained the highest inaccuracy for t2.small instances, explained this by their network bandwidth limitations. More precisely, BWA-99 transfers 364.384 MB to all 64 instances in parallel, which causes a mean FT times of 104.78 ± 47.08s.

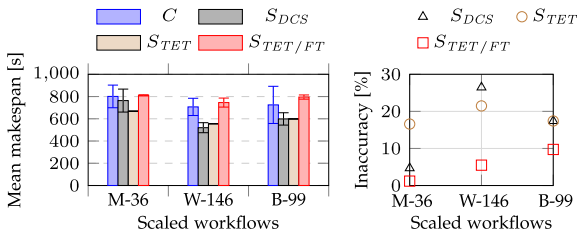


Fig. 8. Mean makespan and its standard deviation (left), and the corresponding makespan inaccuracies relative to the Cloud makespan (right) for Montage-36 (*M-36*), Wien2k-146 (*W-146*), and BWA-99 (*B-99*) with $64\times$TS.

6 Discussion

This section compares our approach with other related research, presents how its advances beyond the state-of-the-art, and discusses a threat of its validity.

6.1 Related Work

6.1.1 Performance Instability

Public Cloud providers, such as Amazon EC2, offer different instance types with certain computation and communication capacity (e.g., CPU speed, network bandwidth), following a pay-as-you-go model. Nonetheless, they omit to publish all parameters and do not ensure them in service level agreements (SLAs) for all instance types. Moreover, they change the Cloud parameters over time, leading to performance instability. An instance of the same VM type may show different performance when executing the same task over some time period. The causes for performance instability are manifold and originate from different underlying hardware for similar instances [21], CPU model of the same instance type, the hour of the day, or the day of the week, as detected by Schad *et al.* [6]. Iosup *et al.* [5] observed yearly and daily patterns of performance instability, as well as periods of constant performance.

In computer workloads and natural sciences, heavy-tailed distributions are ubiquitous. For example, Feitelson [24] declares the distribution of process runtimes as a skewed distribution with many small elements dominated by rare large elements. Normal distributions are rarely used in workload modeling, because the symmetrical and short-tailed normal distribution is inappropriate to directly match skewed data [24]. Our work assumes task and data transfer lengths are normally distributed [3], [25], while Schad *et al.* [6] report such times as not normally distributed. We will address the model under this assumption in future work.

6.1.2 Cloud Simulation

A number of Grid simulators [26] have been developed in the last decades scaling up to hundreds of thousands of heterogeneous machines. Many such simulators such as GroudSim [13] have been extended for Cloud, giving researchers the opportunity to simulate scalable data centers resources (e.g., CPU, memory, network) and application executions on top. CloudSim [12] provides large-scale Cloud simulations for scheduling and resource provisioning purposes, but does not support Cloud performance instability by default. Instead, CloudSim

supports the configuration of various input workloads with rather constant computation and communication VM capacity. The GloudSim [27] simulator introduces partial execution dynamics by resizing the simulated instances, but keeps the VM performance constant over time. NetworkCloudSim [28] extends CloudSim with a scalable network and generalized application model, allowing a more accurate evaluation of scheduling and resource provisioning policies for Cloud infrastructure optimization. Nonetheless, an accurate simulator configuration is challenging, as most public Cloud providers do not provide sufficient network details, while the details of their underlying computing resources are generally unknown.

DynamicCloudSim, developed by Bux and Leser [3] as an extension to CloudSim, is the first significant step forward in performance instability and heterogeneity simulation. The heterogeneity parameters are extensive and cover heterogeneous underlying hardware for the same VM instance type, VM stragglers, VM failures, long and short term fluctuations, and so on. However, the configuration of several heterogeneity parameters is challenging and can easily overwhelm users, as some parameters may require hidden knowledge of the internal Cloud infrastructure. To configure DynamicCloudSim in its evaluation, the authors used a comprehensive set of external research results to be performed before each simulation, as the underlying hardware infrastructure continuously changes and updates. DynamicCloudSim does not model burstable resources and, therefore, does not offer appropriate configuration for simulation. In contrast, our model does not need to model burstable resources, which is an important advantagem, and becomes even more accurate when simulating experiments with burstable `t2` VM instances.

Bux and Leser [3] also evaluated DynamicCloudSim in a homogeneous environment with eight `m1.small` general purpose instances with baseline performance and without burstable features. In comparison, we used a more complex testing environment comprising various compute-optimised and general purpose burstable instances, highly unstable due to bursting above the baseline performance. Similar to [3] and other related works, we also used a homogeneous environment per experiment. A heterogeneous environment falls beyond the scope of this work.

6.1.3 Predictive Simulation

Another approach to simulating workflow execution is by predicting TET of each task [29] based on offline machine learning. Pham *et al.* [30] observed a difficulty in predicting short TET and bandwidth dependent tasks using a two-stage machine learning approach. Nadeem *et al.* [31] predicted TET by creating a template based on several parameters, including networking. However, they used evolutionary programming, which is computationally intensive in a large search space. Seneviratne and Levy [32] used linear regression to predict several runtime parameters (i.e., CPU utilization, disk load) and finally TET. All these works confirmed that predicting TET is not trivial, as the Cloud performance is unstable and varies [21]. Therefore, Cloud noisiness is a valuable metric providing a global view of performance variability generated by workflow and Cloud.

6.2 Threat to Validity

There is a tradeoff for higher accuracy in our simulation approach that requires several training repetitions to determine the workflow performance on specific Cloud resources. Our sensitivity analysis can significantly reduce the number of these experiments by determining the Cloud noisiness from simulations with two VM instances only and reusing it to simulate experiments with a higher number of instances. However, these benefits are limited and can be leveraged only for similar number of instances for the same workflow, when the sensitivity analysis does not cause major increase in error rate. The exchange of Cloud noisiness metrics does not hold true for scaling both the problem and the resources sizes, which produce a higher variation for different scheduling and resource contention strategies. For example, we observed a $\overline{\Delta}_{FT}$ of 0.434 for the scaled BWA-99 workflow, which is nearly double compared to the lower scale BWA with `t2.small` instances. Nevertheless, the additional pre-executions are negligible compared to the effort required for determining the correct values of the DynamicCloudSim parameters.

We evaluated our model for workflows running up to 10 min. A possible threat to the validity of our model are long running workflows, whose makespans spread across multiple hourly or daily patterns of performance instability, as reported by Iosup [5] and Schad [6].

7 Conclusion and Future Work

We presented a new method for simulating (reproducing) a scientific workflow execution in a dynamic Cloud environment. Our model is the first to offer a simple and accurate simulation of burstable instances and provides more accurate simulations than other systems such as DynamicCloudSim supporting stable baseline performance only. In contrast to state-of-the-art simulators, such as DynamicCloudSim that require configuring several of parameters, our approach needs only two metrics called Cloud TET and Cloud FT noisiness, determined by executing several repetitions of a single workflow application.

We conducted an extensive evaluation using three scientific workflows with different computing and communication requirements on Amazon EC2 resources. Our model was on average at least twice more accurate than the state-of-the-art models, regardless of the workflow, the instance types and the number of instances. We observed similar results on an extended testbed with 64 `t2.small` instances. However, the simulated values of the makespan for all simulation models, including our nosining model, are almost always below the actual values of the cloud. Our in-depth analysis revealed that this phenomenon appears due to the limits of DynamicCloudSim to configure all aspects of Cloud networking variability and various workflow dataflow dependencies with a single parameter.

Although determining the Cloud noisiness in public Clouds is costly, the comprehensive sensitivity analysis showed significant reduction in this overhead and in the cost of determining the Cloud instability at both workflow and resource levels. At the resource level, we reduced this time by 67.80 percent by reusing the Cloud noisiness from one experiment in another with different number of resources, thereby diminishing the costs by up to 82.46 percent

and the number of experiments by up to 73.33 percent. The tradeoff in this case is a negligibly lower inaccuracy of up to 1.98 percent in average. At the workflow level, we reduced the costs to determine the Cloud noisiness for burstable instances in public Clouds by 66.6 percent by reusing the results of experiments with other workflows, with a negligible inaccuracy increase of 1.92 percent.

Our proposed noising model accurately simulates the Cloud performance for homogeneous VM instances and workflows with various communication and computation ratios and data flows. In the future, we will generalize it to heterogeneous VM instances and workflows with various control flow patterns, and identify additional Cloud and application performance instability sources that further influence their behavior and execution. Additionally, we will evaluate other distributions than normal (e.g., exponential, heavy-tailed) for long-running workflows. Finally, we will analyse network simulation model discrepancies and evaluate different VM bandwidth configuration approaches.

Appendix A
Artifact Description

We present the reproducibility artifact of our experimental validation, following the template proposed by the IEEE Transactions on Parallel and Distributed Systems journal.

A.1 Overview

- Programs: DynamicCloudSim[2] and ASKALON[3];
- Compilation: see Section A.6;
- Data set: see Section A.5;
- Run-time environment: `Eclipse IDE 2019-03`[4];
- Hardware: see Section A.3;
- Deployment and execution: see Section A.2 and Section A.6;
- Cloud execution (see Section 4):
  - Amazon EC2 instance types (us-east):
    - `t2.small` (1 vCPU);
    - `t2.medium` (2 vCPUs);
    - `t2.xlarge` (4 vCPUs);
    - `c5.large` (2 vCPU);
  - Total expected experimental cost on Amazon EC2 (estimated theoretically): $\sim \$80$;
  - No third-party services.

A.2 How Delivered

We share a capsule[5] at the Code Ocean platform[6]. The capsule provides our source code with the experimental setup, the Cloud noisiness metric files, and the simulation workloads.

A.3 Hardware Dependencies

DynamicCloudSim and ASKALON do not have any special hardware requirements. We executed both on an Intel Core i7-5500U CPU@ 2×2.4GHz machine with 8G DDR3L SDRAM and 256 G of SSD storage. The operating system is Ubuntu 18.04 LTS.

A.4 Software Dependencies

DynamicCloudsim and ASKALON are both written in Java and have the following minimal software requirements.

- ASKALON requires `JDK` (1.7.x), and `Ant` (1.9.2+);
- DynamicCloudSim requires `JDK` (1.7.x), `Maven` (3.6).

A.5 Data Sets

We run the three workflows, Montage (version 6.0), Wien2k, and BWA, with the ASKALON environment available in gitlab[7]. Section 4 of the paper gives a detailed explanation of the workflows and Table A1 lists their parameters. Table A2 shows the parameter differences to Table A1, used to scale the workflows for the experiments with `64 t2.small` instances. We contributed all execution traces of Montage, Wien2k, and BWA to a novel open-access workflow trace archive (WTA[8]).

TABLE A1
Workflow Parameters

(a) Montage.

Name	*Value*
SURVEY	2mass
BAND	j
LOCATION	m51
WIDTH	0.5
HEIGHT	0.5
TILESIZE	1000
MAXLENGTH	8192
ALGORITHM	normal
SEQ_OVERLAPS	40
SEQ_CORRECTIONS	2

(b) Wien2k.

Name	*Value*
integerFraction	7
decimalFraction	0
startInput	STARTINPUT50.txt
fIn0	/atype/atype.in0
fIn1	/atype/atype.in1
fClmsum	/atype/atype.clmsum
fStruct	/atype/atype.struct
fIn2	/atype/atype.in2
fVspdn	/atype/atype.vspdn
testvalue	1
fInc	/atype/atype.inc
fInm	/atype/atype.inm
overFlag	true
doubleValue	9.0
experimentName	sample

(c) BWA.

Name	*Value*
input0	/inputdata/xmedium/input0
jobparams	/inputdata/xmedium/job.params
input1	/inputdata/xmedium/input1
input2	/inputdata/xmedium/input2
num_reads	5000

2. https://github.com/marcbux/dynamiccloudsim
3. http://www.askalon.org
4. https://www.eclipse.org
5. 10.24433/CO.6447616.v1
6. https://codeocean.com/
7. https://gitlab.itec.aau.at/roland-matha/askalon-workflows.git
8. https://wta.atlarge-research.com

TABLE A2
Scaled Workflow Parameters

(a) Montage-36.

Name	*Value*
WIDTH	0.8
HEIGHT	0.8
SEQ_OVERLAPS	200

(b) Wien2k-146.

Name	*Value*
startInput	STARTINPUT150.txt

(c) BWA-99.

Name	*Value*
num_reads	1000

A.6 Installation Outside Code Ocean

ASKALON is free to use for research and/or educational purposes. The source code can be requested at (http://www.askalon.org). For our experiments, we deployed ASKALON according to the *ASKALON User Guide v.1.2*[9].

We downloaded DynamicCloudSim from github[2] and run it without installation in `Eclipse IDE 2019-03`.

A.7 Experiment Workflow

The simulation experiments are executable in Code Ocean by the `run.sh` script. The script compiles the source files and runs different simulation scenarios. All simulation results are printed on the output console. The approximated execution time of all simulation scenarios is $\sim$ 30 s.

Acknowledgments

This work was supported by the ASPIDE Project funded by the European Union's Horizon 2020 Research and Innovation Programme under Grant agreement No. 801091.

References

[1] J. Dejun, G. Pierre, and C.-H. Chi, "EC2 performance analysis for resource provisioning of service-oriented applications," in *Proc. Int. Conf. Service-Oriented Comput.*, 2009, pp. 197–207.
[2] S. Ristov, R. Mathá, and R. Prodan, "Analysing the performance instability correlation with various workflow and cloud parameters," in *Proc. 25th Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2017, pp. 446–453.
[3] M. Bux and U. Leser, "DynamicCloudSim: Simulating heterogeneity in computational clouds," *Future Gener. Comput. Syst.*, vol. 46, pp. 85–99, 2015.
[4] V. Persico, P. Marchetta, A. Botta, and A. Pescapè, "Measuring network throughput in the cloud," *Comput. Netw.*, vol. 93, no. P3, pp. 408–422, Dec. 2015.
[5] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Proc. 11th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2011, pp. 104–113.
[6] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 460–471, Sep. 2010.
[7] R. Mathá, S. Ristov, and R. Prodan, "Simulation of a workflow execution as a real Cloud by adding noise," *Simul. Modelling Practice Theory*, vol. 79, pp. 37–53, 2017.
[8] R. Mathá, S. Ristov, and R. Prodan, "A simplified model for simulating the execution of a workflow in cloud," in *Euro-Par 2017: Parallel Processing*. Cham, Switzerland: Springer, 2017, pp. 319–331.
[9] P. Bryk, M. Malawski, G. Juve, and E. Deelman, "Storage-aware algorithms for scheduling of workflow ensembles in clouds," *J. Grid Comput.*, vol. 14, no. 2, pp. 359–378, Jun. 2016.
[10] G. Juve *et al.*, "Scientific workflow applications on amazon EC2," in *Proc. 5th IEEE Int. Conf. E-Science Workshops*, 2010, pp. 59–66.
[11] H. Moens, K. Handekyn, and F. De Turck, "Cost-aware scheduling of deadline-constrained task workflows in public cloud environments," in *Proc. IFIP/IEEE Int. Symp. oIntegr. Netw. Manage.*, 2013, pp. 68–75.
[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
[13] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "Groudsim: An event-based simulation framework for computational grids and clouds," in *Euro-Par 2010 Parallel Processing Workshops*. Berlin, Germany: Springer, 2011, pp. 305–313.
[14] H. M. Fard, S. Ristov, and R. Prodan, "Handling the uncertainty in resource performance for executing workflow applications in clouds," in *Proc. 9th Int. Conf. Utility Cloud Comput.*, 2016, pp. 89–98.
[15] G. B. Berriman *et al.*, "Montage: A grid-enabled engine for delivering custom science-grade mosaics on demand," in *Optimizing Scientific Return for Astronomy through Information Technologies*, vol. 5493. Bellingham, WA, USA: International Society for Optics and Photonics, 2004, pp. 221–233.
[16] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization," *Sci. Program.*, vol. 2015, pp. 5:5–5:5, Jan. 2015.
[17] H. Nawaz, G. Juve, R. Ferreira da Silva, and E. Deelman, "Performance analysis of an I/O-intensive workflow executing on google cloud and amazon web services," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2016, pp. 535–544.
[18] K. Schwarz, P. Blaha, and G. Madsen, "Electronic structure calculations of solids using the WIEN2k package for material sciences," *Comput. Phys. Commun.*, vol. 147, pp. 71–76, Aug. 2002.
[19] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrows-wheeler transform," *Bioinformatics*, vol. 26, pp. 589–95, Mar. 2010.
[20] S. Ostermann, R. Prodan, and T. Fahringer, "Extending grids with cloud resource management for scientific computing," in *Proc. 10th IEEE/ACM Int. Conf. Grid Comput.*, 2009, pp. 42–49.
[21] K. R. Jackson *et al.*, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, 2010, pp. 159–168. [Online]. Available: https://doi.org/10.1109/CloudCom.2010.69
[22] P. Velho, L. M. Schnorr, H. Casanova, and A. Legrand, "On the validity of flow-level TCP network models for grid and cloud simulations," *ACM Trans. Model. Comput. Simul.*, vol. 23, no. 4, pp. 23:1–23:26, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2517448
[23] H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, and R. F. da Silva, "Wrench: A framework for simulating workflow management systems," in *Proc. IEEE/ACM Workflows Support Large-Scale Sci.*, 2018, pp. 74–85.
[24] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*, 1st ed. New York, NY, USA: Cambridge Univ. Press, 2015.
[25] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," in *Proc. IEEE 28th Int. Conf. Advanced Inf. Netw. Appli.* 2014, pp. 858–865.
[26] W. Depoorter, N. De Moor, K. Vanmechelen, and J. Broeckhove, "Scalability of grid simulators: An evaluation," in *Proc. 14th Int. Euro-Par Conf. Parallel Process.*, 2008, pp. 544–553.
[27] S. Di and F. Cappello, "Gloudsim: Google trace based cloud simulator with virtual machines," *Softw.–Practice Experience*, vol. 45, no. 11, pp. 1571–1590, Nov. 2015.
[28] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, 2011, pp. 105–113.
[29] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

9. http://www.askalon.org/documents/ASKALONUserGuide-final.pdf

[30] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2017.2732344.
[31] F. Nadeem and T. Fahringer, "Optimizing execution time predictions of scientific workflow applications in the grid through evolutionary programming," *Future Genereration Comput. Syst.*, vol. 29, no. 4, pp. 926–935, Jun. 2013.
[32] S. Seneviratne and D. C. Levy, "Task profiling model for load profile prediction," *Future Gener. Comput. Syst.*, vol. 27, no. 3, pp. 245–255, Mar. 2011.

Roland Mathá received the BSc degree in computer science and the MSc degree from the University of Innsbruck, Austria, in 2011 and 2014, respectively. Since January 2015, he is working toward the PhD degree. His research interests include Cloud simulations, workflow applications, and multi-objective optimisations.

Sasko Ristov received the PhD degree in computer science from the University of Ss. Cyril and Methodius, Skopje, Macedonia (UKIM), in 2012, and was an assistant professor at UKIM until 2019. Since 2016 he is a postdoctoral university assistant at the University of Innsbruck. His research interests include performance analysis and optimization of distributed systems. He received the UKIM's Best Scientist Award in 2012 and an IEEE Conference Best Paper Award in 2013.

Thomas Fahringer (Member, IEEE) received the PhD degree from the Vienna University of Technology, in 1993. Since 2003, he is a full professor of computer science at the Institute of Computer Science, University of Innsbruck, Austria. His research interests include software architectures, programming paradigms, compiler technology, performance analysis, and prediction for parallel and distributed systems.

Radu Prodan (Member, IEEE) received the PhD degree from the Vienna University of Technology, in 2004, He was a associate professor until 2018 at the University of Innsbruck, Austria. He is currently professor in distributed systems at the Institute of Software Technology, University of Klagenfurt. His research interests include performance, optimisation, and resource management tools for parallel and distributed applications. He authored more than 100 publications and received two IEEE best paper awards.