Cloud simulation

September 15, 2016

Abstract

Simulating the cloud from the client point of view.
Assessment of the impact of different metrics.
Recommendations for using cloud simulators.
WARNING: figures in text might not be up to date.

1 Introduction

The problem of allocating cloud resources in performant, robust and energy-efficient ways is of paramount importance in today's usage of computing infrastructures, and a number of research papers have contributed new allocation techniques to address this issue. A pitfall of research on IaaS lies in the validation of the models and algorithms proposed, which requires infrastructures that are difficult to set up for individual researchers. As a consequence, many researchers evaluate their work through simulation. A number of simulators have been developed for that purpose ([**?**]). They are typically based on discrete-event simulation, using models for each elementary component of the infrastructure, which are then composed to simulate the whole system and applications running on it. This approach is attractive from the infrastructure provider perspective since the simulated system can be finely customized. However, such an *ab initio* construction poses a significant problem regarding the calibration and validation of the composed model against real-world measurements. Another approach is the simulation from the client-side (from the application perspective), which can specifically focus on the prediction of a couple of metrics, such as the makespan and the cost of the application. It is noticeable that the research adopting a client-side view generally evaluate the accuracy of their simulation model through experimental studies, while such an evaluation is generally missing for infrastructure-wide simulators.

Altough much fewer works address the simulation from the client perspective, several very different methods have been proposed to reach this goal. Among these works, detailed in the related work section, are EMUSIM [1] that use emulation, PICS [3] which implemens a simplified discrete event simulator, and [4] who build a statistical model from observations.

...

We advocate that a precise assessment of simulation should be carried out against real execution figures to better understand the limits of simulation applicability. In this paper, we study the simulation accuracy on several use cases ran on an IaaS clouds. The use cases comprise two different type of applications

(workflow and bag-of-tasks), with several size instances for each of them, and each application is operated on two different type of infrastructure (private and public).

We assume an automated process making the provisioning and scheduling decisions on behalf the user on the real infrastructure. To that purpose, we use *Schlouder* [], a client-side cloud resource broker. These scheduling algorithms of Schlouder have been reimplemented in a simulation system, based on the simulation toolkit SimGrid [2]. Our study aims to isolate the different parameters that influence the simulation accuracy and what degree of divergence between real execution and simulation might be expected in each case.

to study the techniques for allocating cloud resources. in more robust, efficient, and ecologically sustainable ways. Unfortunately, the wide-spread use of these techniques in production systems has, to date, remained elusive. One reason for this is that the state of the art for investigating these innovations at scale often relies solely on model-driven simulation.

Production-grade cloud software, however, demands certainty and precision for development and business planning that only comes from validating simulation against empirical observation. In this work, we take an alternative approach to facilitating cloud research and engineering in order to transition innovations to production deployment faster. In particular, we present a new methodology that complements existing model-driven simulation with platform-specific and statistically trustworthy results. We simulate systems at scales and on time frames that are testable, and then, based on the statistical validation of these simulations, investigate scenarios beyond those feasibly observable in practice. We demonstrate the approach by developing an energy-aware cloud scheduler and evaluating it using production and synthetic traces in faster than real time. Our results show that we can accurately simulate a production IaaS system, ease capacity planning, and expedite the reliable development of its components and extensions.

2 Setup

2.1 Xp traces

274 xps

2.1.1 Schlouder

2.1.2 Applications

OMSSA (BoT CPU-intensive) and Montage (WF data-intensive). 3 usecases each.

Two differents provisioning Strategies: AFAP and ASAP.

2.1.3 Platform

Openstack-ICPS (private, homogeneous)

BonFire (public, heterogeneous)

2.2 Simulations

2.2.1 Simulator

SimSchlouder/SCHIaaS/Simgrid
SimSchlouder reproducing Schlouder runs. Same input/output file.

2.2.2 Lab

2.2.3 Procedure

1. Real executions (xp) to test Schlouder and provisioning/scheduling strategies. Schlouder/SimSchlouder input:
   - Nodes: boottime prediction, amount limit, standard power
   - Tasks: walltime prediction
2. Normalization of xp traces (4 versions of schlouder, missing data)
3. Extraction of information about each xp:
   - Nodes: provisioning date, start date, end date, boottimes, instance type
   - Tasks: submission date, scheduling date, start date, end date, walltime, input time and size, runtime, output time and size, management time
4. Simulation of each xp, injecting different information from real xps
5. Comparison of Schlouder and SimSchlouder outputs (python)
6. Statistical analysis of all traces (R)
7. Close analysis of each outlier to understand the differences.

2.2.4 life-cycles and observed times

- Execution: $e \in E$
- Node of execution e: $n \in N_e$
- Task of execution e: $t \in T_e$
- Task handled by node n: $t \in T_n$
- The node running the task t is denoted $n_t \in N$
- v^R denotes the value v in the rality
- v^S denotes the value v in the simulation
-

During the execution, the node are in the following states:

1. Future: Once the decision to start the node is made;

2. Pending: Once the node is requested to the cloud-kit;
3. Booting: Once the cloud-kit aknowledge the satisfaction of the request;
4. Idle: Once the node is ready to run tasks;
5. Busy: Once the node is running one task;
6. ShuttingDown: Once the termination of the node is aked to the cloud-kit;
7. Terminated: Once the node is terminated.

The observed times are:

- $uptime(n) = terminated_n - booting_n$
- $boottime(n) = idle_n - booting_n$

During the execution, the task are in the following states, each corresponding to one date:

1. Pending: Once they are subitted to the system;
2. Scheduled: Once the system decided on which node the taks should be executed;
3. Submitted: Once the task is sent to the worker node;
4. Inputting: Once the task begin to download its data;
5. Running: Once the task begin to computed;
6. Outputting: Once the task begin to upload its result;
7. Finished: Once the task is finished;
8. Complete: Once the system aknowledge the completion of the task.;

The observed times are:

- $walltime(t) = complete_t - submitted_t$
- $inputtime(t) = running_t - inputting_t$
- $runtime(t) = outputting_t - running_t$
- $outputtime(t) = finished_t - outputting_t$
- $managementtime(t) = walltime_t - (inputtime_t + runtime_t + outputtime_t)$
- or $managementtime(t) = (inputting_t - submitted_t) + (complete_t - finished_t)$

3 Results

4 metrics $m \in M$ for each execution $e \in E$:

- uptime: amount of rented resources, cost

$$uptime(e) = \sum_{n \in N_e} uptime_n$$

- makespan: duration of the xp from the submission of the first task to the end of the last task, user experience

$$makespan(e) = max_{t \in T_e} complete_t$$

- usage: runtime / uptime, efficiency of the provisioning

$$usage(e) = \frac{\sum_{t \in T_e} walltime_t}{\sum_{n \in N_e} uptime_n}$$

- schederror: number of tasks that are not assigned to the same node in the simulation compared to the reality, accuracy of the scheduling decisions

$$schederror(e) = |t \forall t \in T / t_n^R \neq t_n^S|$$

Absolute errors are computed for each metric $m \in M$:

$$m.ae(e) = \frac{|m^S(e) - m^R(e)|}{m^R(e)}$$

Results are shown as frequencies and statistics (stat = min, mean, median, max) of absolute errors occurrences. Frequencies are weighted so that the two applications weigth the same, and the two platforms weigth alos the same (i.e. each couple application $\times$ platform represents 1/4th of the frequencies).

To compare absolute errors between set of simulations S and S' S being the reference:

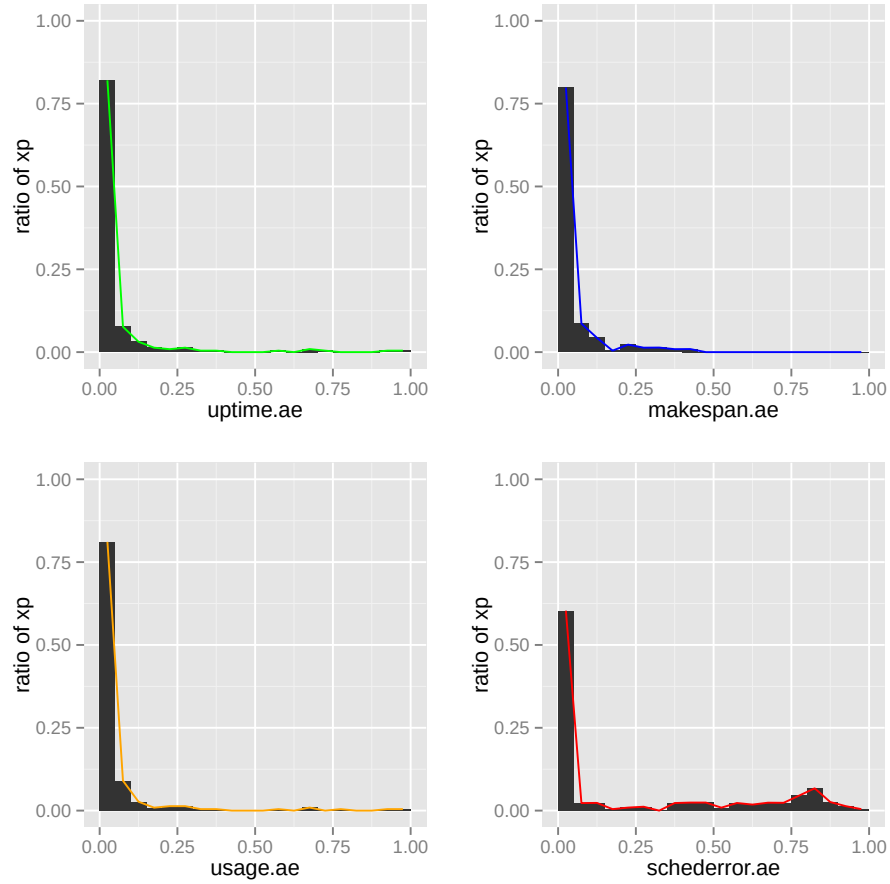
- $\delta stat(m.ae(E)) = stat_{e \in E}(m.ae^{S'}(e)) - stat_{e \in E}(m.ae^S(e))$
- $\Delta stat(m.ae(E)) = stat_{e \in E}(m.ae^{S'}(e) - m.ae^S(e))$

3.1 Simulator accuracy

Best simulation we can do.

Assess the raw simulator accuracy, injecting all real-life hazards that can be captured : boottimes, walltimes and scheduling dates.

Scheduling dates allow to simulate some internal threaded mechanisms of Schlouder. Schlouder uses two threads: the node manager and the task manager. At settled intervals, the node manager interrupts the task manager to start and stop new nodes. This changes the state of nodes, which influence provisioning and scheduling decisions. However, simulating the exact moment of this interruption is utterly difficult, leading to differences between simulation and reality.



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.025	0.019	0.027	0.131
median	0.001	0.000	0.001	0.000
max	0.995	0.406	0.999	0.961

Figure 1: Frequencies and statistics about absolute error of best simulations (274 xp)

- uptime: 86% show less than 0.05 of absolute error, 92% less than 0.10, 2 simulations exceed 0.30, ranging from 0.00 to 0.50, for a mean of 0.025 and a median of 0.001
- makespan: 76% show less than 0.05 of absolute error, 90% less than 0.10, 0 simulations exceed 0.30, ranging from 0.00 to 0.62, for a mean of 0.042 and a median of 0.018
- usage: 59% show less than 0.05 of absolute error, 91% less than 0.10, 2 simulations exceed 0.30, ranging from 0.00 to 0.60, for a mean of 0.043 and a median of 0.002
- schederror: 70% show less than 0.05 of absolute error, 72% less than 0.10, 59 simulations exceed 0.30, ranging from 0.00 to 0.965, for a mean of 0.155 and a median of 0.000

If global metrics are quite accurately assessed by the simulator, the scheduling decisions can be very different between simulation and reality. One part of the explanation is that scheduling decisions are interdependent: any error leads to several others.

3.2 Simulator accuracy according to platforms and applications

- openstack-icps / omssa (107 xp):

	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.001	0.001	0.001	0.024
median	0.000	0.000	0.001	0.000
max	0.029	0.079	0.028	0.749

All metrics are almost perfectly assessed (mean AR from 0.001 to 0.002) except scheduling error (mean 0.04 and max 0.75, 13% of xp show at least one error), leading to small makespan and usage errors.

We looked at each single case of scheduling error and all those errors comes from ambiguities in the scheduling algorithms.

This is a first limitation of simulation: Whenever heuristics lead to several equivalent solutions, the decision is made by the implementation and relies on data structures (e.g. selection of the first encountered suitable solution) or clocks (e.g. the solution differs from a second to the next, which depends on threads activations and timers). While we made sure to use the same structures and timers, some clocks-related events can not be captured nor simulated: Processing the nodes and tasks queues for scheduling and provisioning decisions take time. Consequently, if those decisions rely on clock, they change during the decision process in reality, as clocks advance by itself, but not in simulation, as clocks advance only explicitly.

Thus, the simulation is not mistaken, but only different from reality. Actually, the decisions made by the simulator are exactly those that one can expect, while the decisions made by the real scheduler are sometimes difficult to understand.

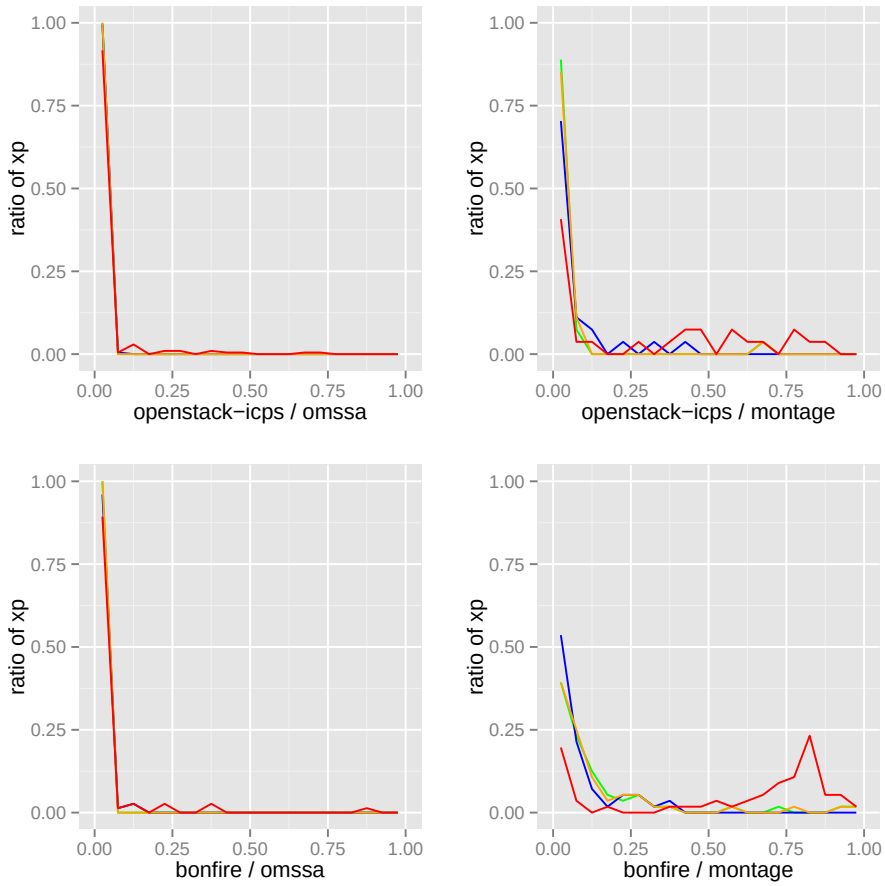
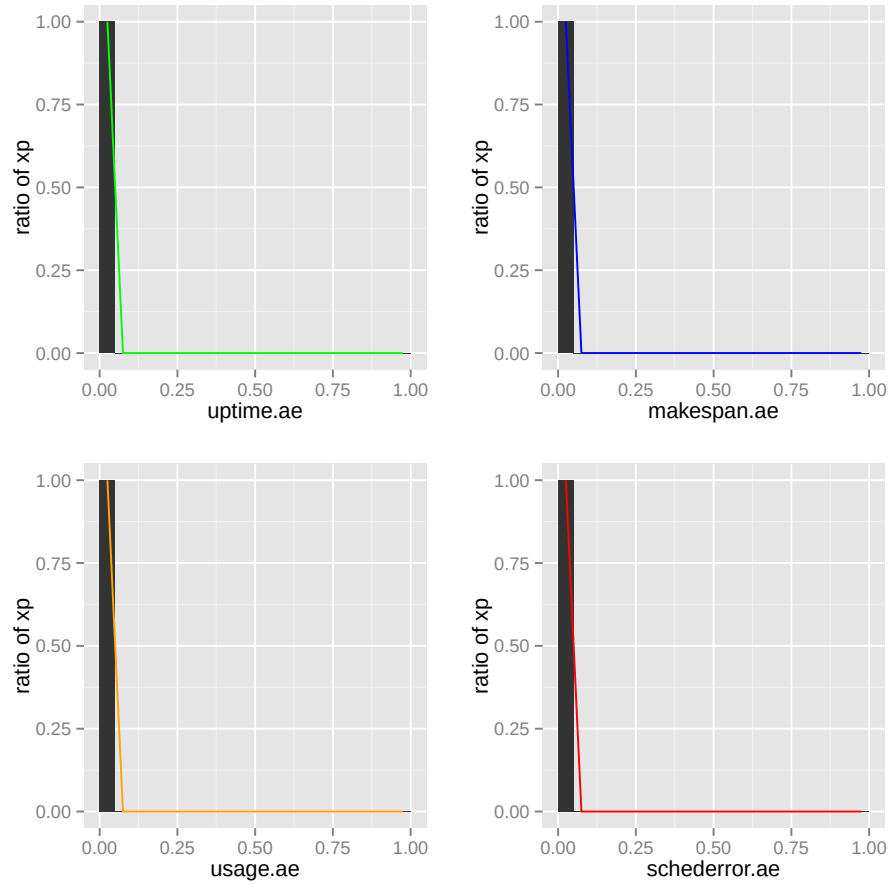


Figure 2: Absolute error frequencies of best simulations according to platforms and applications



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.001	0.000	0.001	0.000
median	0.000	0.000	0.001	0.000
max	0.003	0.010	0.011	0.000

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	0.000	+0.000	0.000	0.000	0.000	+0.000	0.000	0.000
mean	0.001	−0.000	0.000	−0.001	0.001	−0.000	0.000	−0.024
median	0.000	+0.000	0.000	+0.000	0.001	+0.000	0.000	0.000
max	0.003	−0.027	0.010	−0.069	0.011	−0.017	0.000	−0.749

Figure 3: Frequencies and statistics about absolute error of best simulations for openstack-icps / ommssa, without scheduling error cases (91 xps)

Filtering the xps showing clocks-related issues (16 xps), the results are perfect: all metrics present a mean ae of at most 0.001.

The less accurate simulation shows a makespan absolute error of 0.010. Actually, the makespan of the simulation is $94s$, whereas it is $95s$ in reality. This small difference is due to one lag between two consecutive tasks in the middle of the simulation. Such lags are not injected in our simulations.

This shows that, providing that one can inject the right information, the only limitation of our simulator are micro clock-related hazards.

- openstack-icps / montage (36 xps):

	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.034	0.058	0.043	0.307
median	0.000	0.002	0.011	0.258
max	0.665	0.406	0.664	0.851

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	0.000	+0.000	0.000	+0.000	0.000	+0.000	0.000	0.000
mean	0.034	+0.033	0.058	+0.057	0.043	+0.042	0.307	+0.283
median	0.000	+0.000	0.002	+0.002	0.011	+0.010	0.258	+0.258
max	0.665	+0.635	0.406	+0.327	0.664	+0.636	0.851	+0.102

With a work-flow, scheduling errors are more numerous (ae mean of 0.24 for a max of 0.79), leading to less accurate assessments of uptime, makespan and usage (mean ae of 0.01, 0.05, and 0.01), that is ten times more than with a BoT.

First, montage has much more tasks (from 43 to 1672) than omssa (from 33 to 223). Consequently, queues are much longer, which increases the clock-related issues.

Second, BoT scheduling are actually made offline (i.e. scheduling decisions are taken before any actual execution), while WF scheduling implies decisions during the execution, every time dependencies are satisfied. Those decisions rely on the system state (predicted end date of nodes for instance). Consequently, divergences between simulation and reality have more important impacts with WF than with BoTs.

For instance, the worst case shows a very large amount of scheduling errors (0.954). A close examination of this case shown that the simulation behave as expected : After the first dependencies were satisfied, three newly ready tasks $t1$, $t2$, and $t3$ were scheduled on the node n. However in reality, scheduling takes time. During this time, the last task scheduled to node n was completed between the scheduling of $t2$ and $t3$, but before $t1$ were actually submitted to n. This lead to mistakingly set the state of node n to idle, impacting the scheduling decision of $t3$.

Those kind of complex and unforeseeable events are actually frequent when confronted to reality. However, they are utterly difficult to detect (1672

jobs were scheduled for the presented case). Comparing real execution with simulation allow the detection of such case, without having to look at each scheduling decision.

the last task assigned to node n was completed during the scheduling of the tasks which dependencies were satisfied first. But those tasks were intended to This completion lead Schlouder to mistake the state of the

- bonfire / omssa (75 xp):

	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.002	0.009	0.005	0.032
median	0.001	0.004	0.004	0.000
max	0.044	0.134	0.047	0.857

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	0.000	+0.000	0.000	+0.000	0.000	+0.000	0.000	0.000
mean	0.002	+0.002	0.009	+0.008	0.005	+0.003	0.032	+0.008
median	0.001	+0.001	0.004	+0.004	0.004	+0.003	0.000	0.000
max	0.044	+0.015	0.134	+0.054	0.047	+0.019	0.857	+0.108

On a public shared heterogeneous cloud, scheduling errors are more numerous (AR mean of 0.03 for a max of 0.86), leading to less accurate assessments of uptime, makespan and usage (mean AR of 0.005, 0.045, and 0.053).

More interesting, usage are never perfectly assessed: 16% of xp show less than 0.05 of AR, while 86% show an AR between 0.05 and 0.10

This show the impacts of public heterogeneous platforms on simulation accuracy: It is not possible to precisely simulate the vm-to-pm scheduling algorithm of public cloud, as they are generally not public, and their decisions impacts performances, as one can not predict the power of the VM one get.

- bonfire / montage (56 xp):

	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.001	0.001	0.000
mean	0.138	0.082	0.139	0.572
median	0.081	0.048	0.078	0.742
max	0.995	0.375	0.999	0.961

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	0.000	+0.000	0.000	+0.000	0.000	+0.000	0.000	0.000
mean	0.002	+0.002	0.009	+0.008	0.005	+0.003	0.032	+0.008
median	0.001	+0.001	0.004	+0.004	0.004	+0.003	0.000	0.000
max	0.044	+0.015	0.134	+0.054	0.047	+0.019	0.857	+0.108

On a public shared heterogeneous cloud, scheduling errors are even more numerous (AR mean of 0.48 for a max of 0.96), leading to less accurate assessments of uptime, makespan and usage (mean AR of 0.10, 0.115, and 0.48).

This is simply explained by the cumulation of inaccuracies from both platform and applications.

3.3 Boottime impacts

Assessing the impact of efficient boottimes simulation.

Same simulations, without injecting the boottimes observations. Thus, boottimes are only predictions, based on linear regressions of previously observed boottimes.

The worst case show a makespan ae of 0.816 (3141s instead of 17076s). This is due to boottimes on BonFire that were completely of charts: 5 boots were normal (ranging from 232s to 311s), the 17 others ranged from 3281s to 11084s. Whereas BonFire were intended to deliver 22 simultaneous VMs, only 5 were available at the time of the experiment. Instead of refusing the following 17 VMs, the provisioning system of BonFire put them in pending state, waiting for the delivered ones to stop. The VMs being provisioned for one hour, following the 5 normal boots, 5 boots took approximatively 1 hour, then 5 other boots took 2 hours, and 5 another more took 3 hours. Finally, 2 boots took 1 hour after the last dependencies were satsified.

This illustrates that defective clouds can not be efficiently simulated without proper information capture. However, once captured, this kind of defection is perfectly simulated by SchIaaS. Consequently, it can be used to assess behavior and robustness of solutions facing these defections.

Some case are surprisingly improved without the real boot times injection: For instance, one xp shows a real makespan of 25788s, for 35106s with boot times injection and 24266s without.

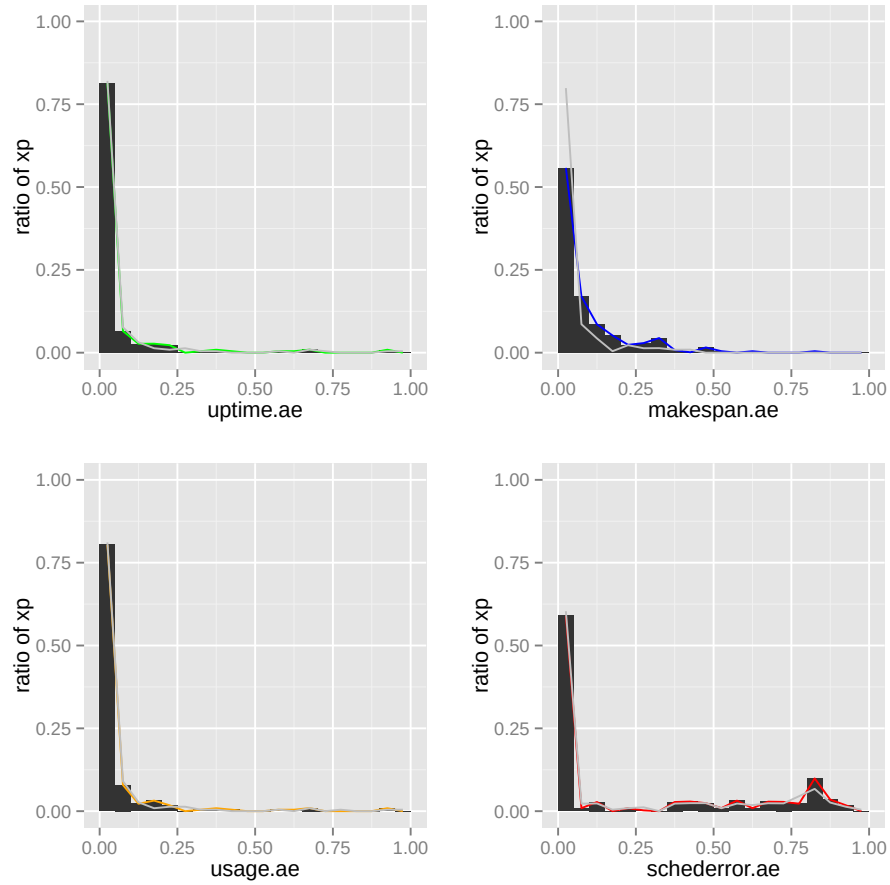
3.3.1 No-threads

Injection of: real boot times and some times due to Schlouder internal threads, such as lapses after a node become ready and the start of the first job.

Assess the impact of efficient internal threads simulation

3.3.2 Communications

Injection of: real boot times, some times due to Schlouder internal threads, such as lapses after a node become ready and the start of the first job, and, real

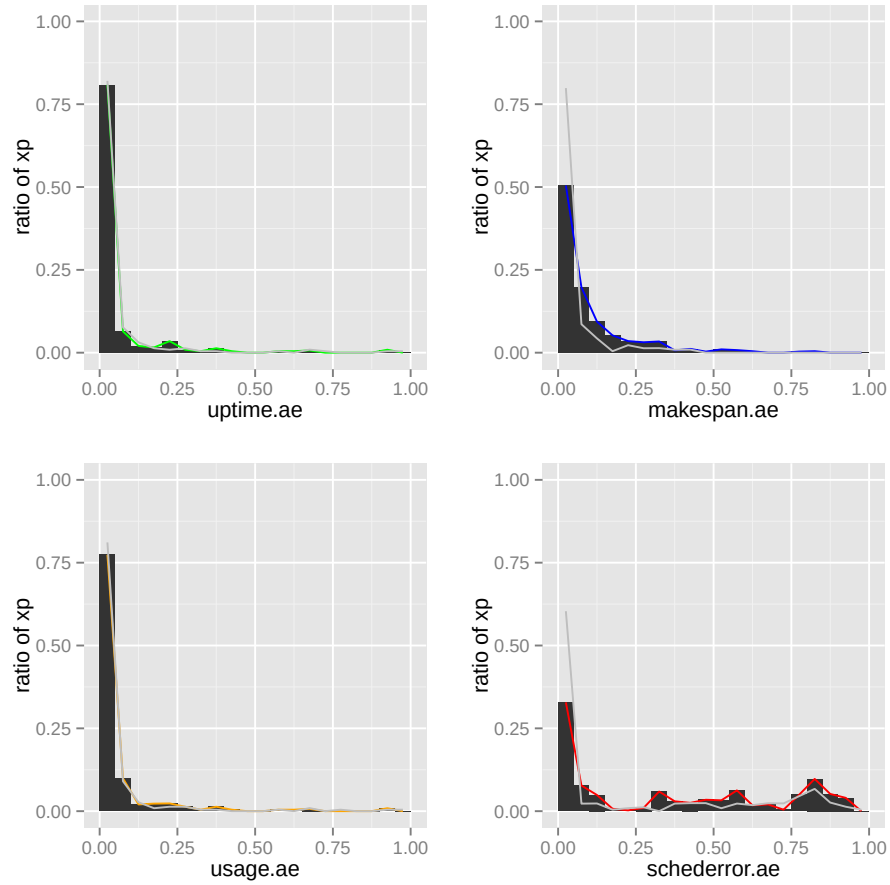


	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.028	0.061	0.029	0.142
median	0.001	0.008	0.002	0.000
max	0.918	0.815	0.923	0.949

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	0.000	+0.000	0.000	+0.000	0.000	+0.000	0.000	0.000
mean	0.028	+0.003	0.061	+0.042	0.029	+0.003	0.142	+0.011
median	0.001	+0.000	0.008	+0.008	0.002	+0.000	0.000	0.000
max	0.918	−0.077	0.815	+0.409	0.923	−0.077	0.949	−0.012

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.373	−0.164	−0.375	−0.288
mean	+0.003	+0.042	+0.003	+0.011
median	0.000	+0.004	0.000	0.000
max	+0.412	+0.807	+0.406	+0.583

Figure 4: Frequencies, statistics, and comparison with best of simulations with no real boot times injection



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.031	0.070	0.034	0.272
median	0.001	0.025	0.002	0.108
max	0.917	0.815	0.920	0.929

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	0.000	−0.000	0.000	+0.000	0.000	+0.000	0.000	0.000
mean	0.031	+0.007	0.070	+0.051	0.034	+0.007	0.272	+0.141
median	0.001	+0.001	0.025	+0.024	0.002	+0.000	0.108	+0.108
max	0.917	−0.077	0.815	+0.409	0.920	−0.079	0.929	−0.031

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.375	−0.199	−0.377	−0.601
mean	+0.007	+0.051	+0.007	+0.141
median	0.000	+0.011	−0.000	+0.046
max	+0.412	+0.807	+0.407	+0.870

Figure 5: Frequencies, statistics, and comparison with best of simulations with no real thread times injection

runtimes and real data size for jobs input and output communications.

Assess the impact of efficient communications

3.3.3 Prediction

Injection of nothing from the real xp, except the xp description as submitted to schlouder.

Assess the efficiency of using a simulator as a predictor of a cloud.

4 Open-science

```
git clone https://git.unistra.fr/gossa/schlouder-traces.git
git clone https://scm.gforge.inria.fr/anonscm/git/schiaas/schiaas.git
cd schiaas
cmake .
make
cd lab
./lap.py -p2 setup/simschlouder/validation.cfg
cd setup/simschlouder/validation-results
ls
```

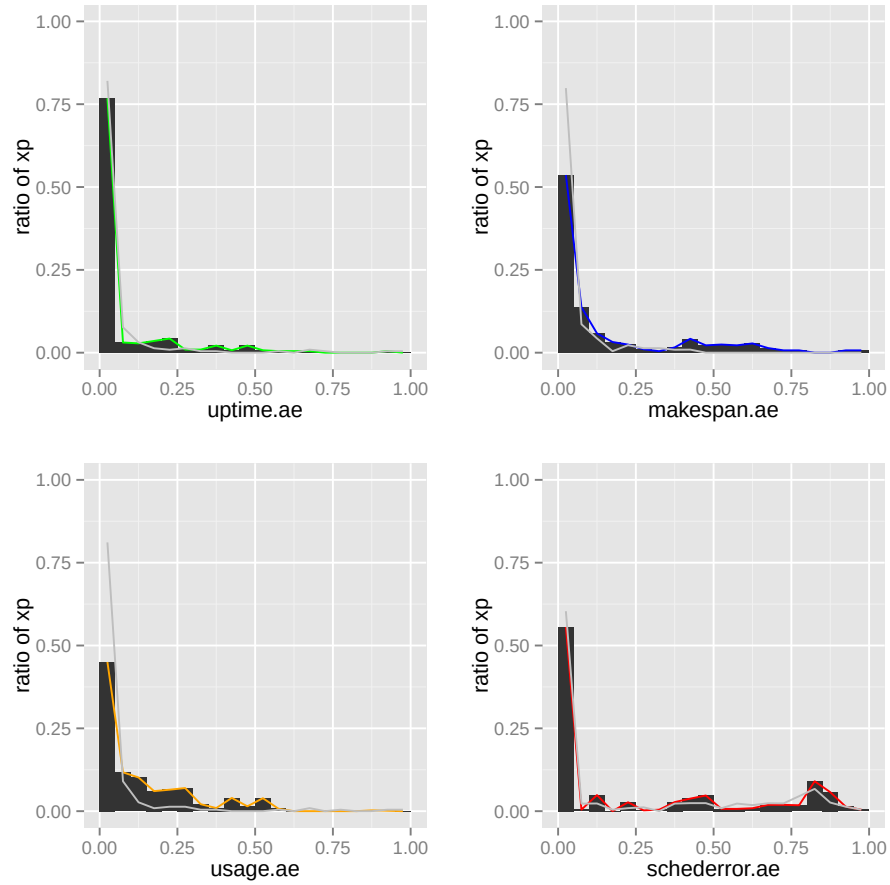
References

[1] Rodrigo N. Calheiros, Marco Aurélio Stelmar Netto, César A. F. De Rose, and Rajkumar Buyya. EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Softw., Pract. Exper.*, 43(5):595–612, 2013.

[2] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: a generic framework for large-scale distributed experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.

[3] In Kee Kim, Wei Wang, and Marty Humphrey. PICS: A public iaas cloud simulator. In *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, pages 211–220, 2015.

[4] Alexander Pucher, Emre Gul, Rich Wolski, and Chandra Krintz. Using trustworthy simulation to engineer cloud schedulers. In *2015 IEEE International Conference on Cloud Engineering, IC2E 2015, Tempe, AZ, USA, March 9-13, 2015*, pages 256–265, 2015.



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	-1.000	0.000	-1.000	0.000
mean	0.064	0.088	0.123	0.148
median	0.001	0.013	0.026	0.000
max	2.068	1.021	2.219	0.961

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	-1.000	−1.000	0.000	+0.000	-1.000	−1.000	0.000	0.000
mean	0.064	+0.039	0.088	+0.069	0.123	+0.096	0.148	+0.017
median	0.001	+0.000	0.013	+0.012	0.026	+0.024	0.000	0.000
max	2.068	+1.073	1.021	+0.614	2.219	+1.220	0.961	0.000

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.141	−0.211	−0.680	−0.426
mean	+0.047	+0.055	+0.096	+0.017
median	0.000	+0.004	+0.011	0.000
max	+2.015	+0.982	+2.219	+0.751

Figure 6: Frequencies, statistics, and comparison with best of simulations with simulation of communications

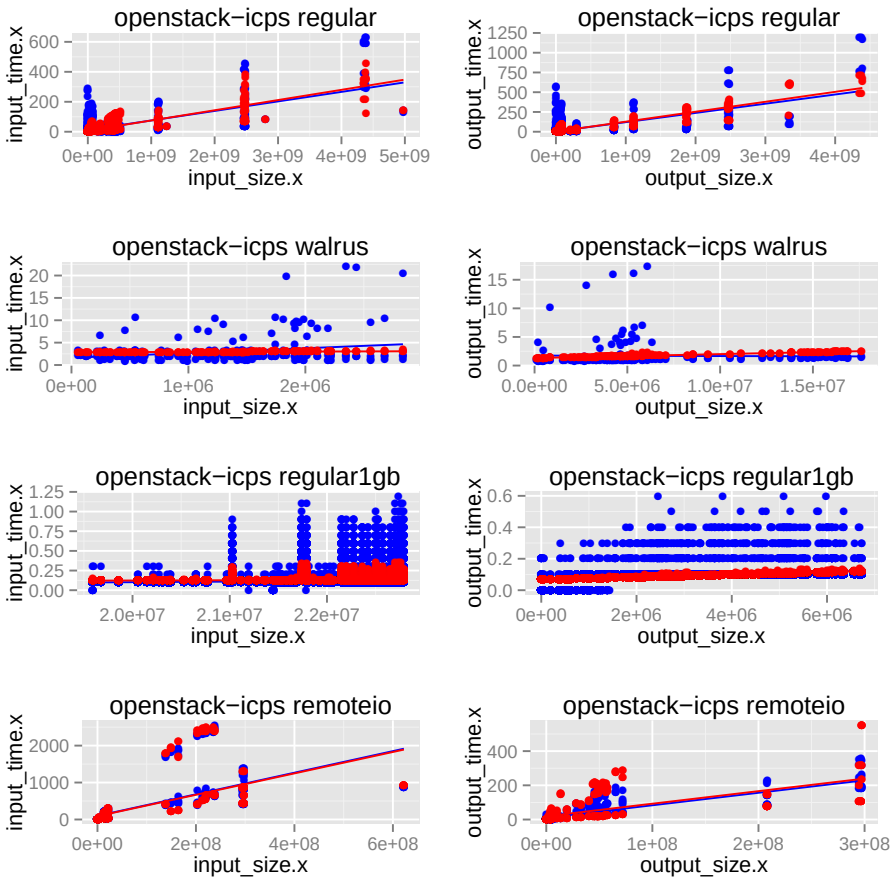


Figure 7: Linear regressions of communication times vs. data size, according to platform, storage, and communication direction on openstack-icps

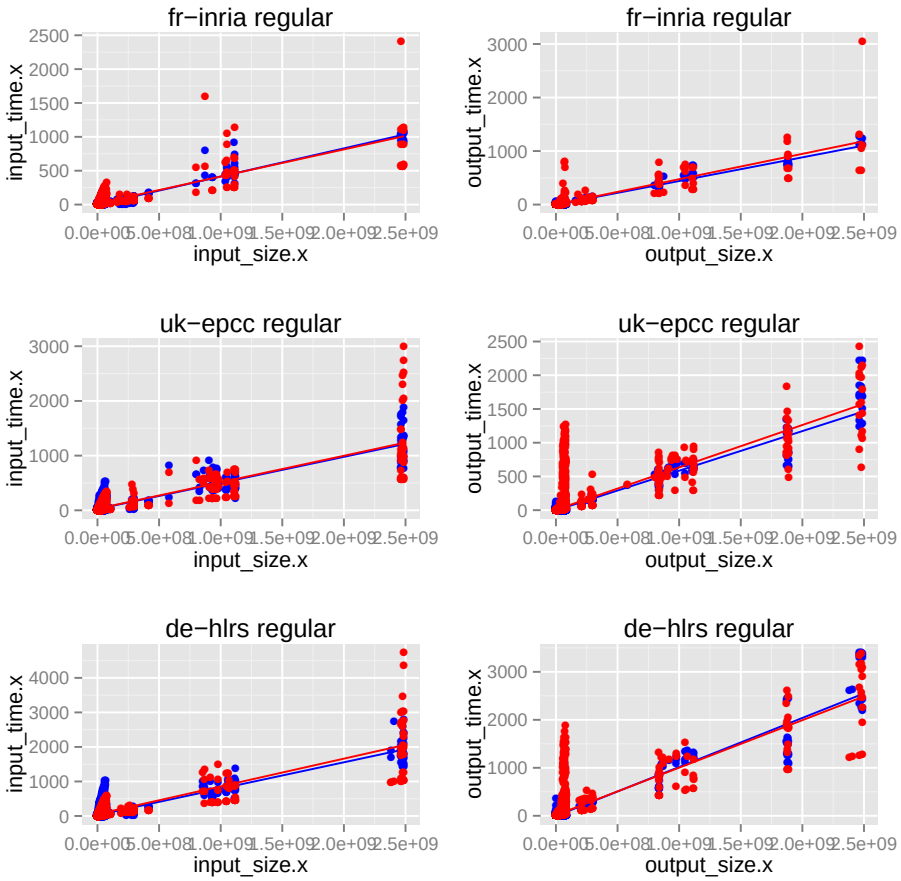
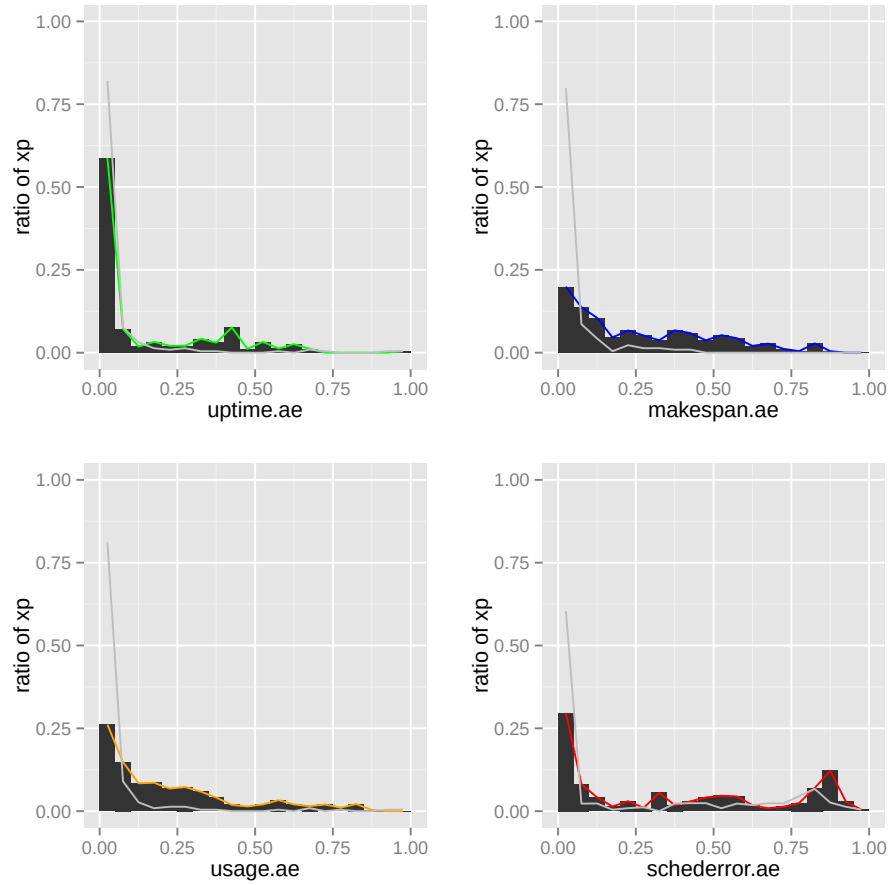


Figure 8: Linear regressions of communication times vs. data size, according to platform, storage, and communication direction on BonFire



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	-1.000	0.006	-1.000	0.000
mean	0.124	0.212	0.316	0.279
median	0.003	0.115	0.135	0.108
max	2.548	1.772	17.227	0.965

	uptime.ae	$\delta_{uptime.ae}$	makespan.ae	$\delta_{makespan.ae}$	usage.ae	$\delta_{usage.ae}$	schederror.ae	$\delta_{schederror.ae}$
min	-1.000	-1.000	0.006	$+0.006$	-1.000	-1.000	0.000	0.000
mean	0.124	$+0.099$	0.212	$+0.192$	0.316	$+0.290$	0.279	$+0.148$
median	0.003	$+0.002$	0.115	$+0.115$	0.135	$+0.134$	0.108	$+0.108$
max	2.548	$+1.554$	1.772	$+1.365$	17.227	$+16.227$	0.965	$+0.004$

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	-0.141	-0.211	-0.680	-0.426
mean	$+0.047$	$+0.055$	$+0.096$	$+0.017$
median	0.000	$+0.004$	$+0.011$	0.000
max	$+2.015$	$+0.982$	$+2.219$	$+0.751$

Figure 9: Frequencies, statistics, and comparison with best of simulations with no injection