Cloud simulation

June 8, 2017

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 papers adopting a client-side view generally include an evaluation of the simulation 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 [**?**] that use emulation, PICS [**?**] which implemens a simplified discrete event simulator, and [**?**] who build a statistical model from observations.

In this paper, we study the simulation also from the application perspective, using the discrete event simulation toolkit SimGrid [**?**] to implement our model. 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. This paper's contribution is twofold:

- We propose a simulation tool
- We make an in-depth analysis of the factors that impact the simulation accuracy, and in this regard go further than the related works. We analyze the sensitivity of several parameters, among which the impact of the job submission management overheads, the effect of inaccuracies in the job execution times specified by the user, or the boot time of VMs. The study is carried out on several use cases which 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 advocate that a precise assessment of simulation should be carried out against real execution figures to better understand the limits of simulation applicability.

The paper is organized as follows : ....

2 Related Work

EMUSIM combines emulation and simulation to extract information automatically from the application behavior (via emulation) and uses this information to generate the corresponding simulation model. Such a simulation model is then used to build a simulated scenario that is closer to the actual target production environment in terms computing resources available for the application and request patterns. - EMUSIM operates uniquely with information that is available for customers of public IaaS providers - Automated Emulation Framework (AEF) [2] for emulation and CloudSim [3] for simulation - emulation : run the actual software on a subset of the hardware (=hardware model) - application BoT

3 Setup / Context

3.1 Real Environment

Evaluation of simulation is done by comparing simulation logs, against logs generated by running actual scientific workflows on multiple platforms. Though our work on cloud scheduling and cloud simulation we acquired archive of 274 experimental executions on two wildly differing environment.

3.1.1 Schlouder

Every experiment log in the archive was generated by running a scientific workflow through a client-side cloud resource broker named*Schlouder* [**?**]. *Schlouder* automates the provisioning of resources needed to execute a workflow, the scheduling of tasks to the provisioned resources and, monitors the proper execution of tasks.

Provisioning is done thought a cloud kit interface that allows for Schlouder to communicate with different clouds in order to start and stops the necessary **vm!**s (**vm!**s) to execute the workload. To match account for the pricing of commercial

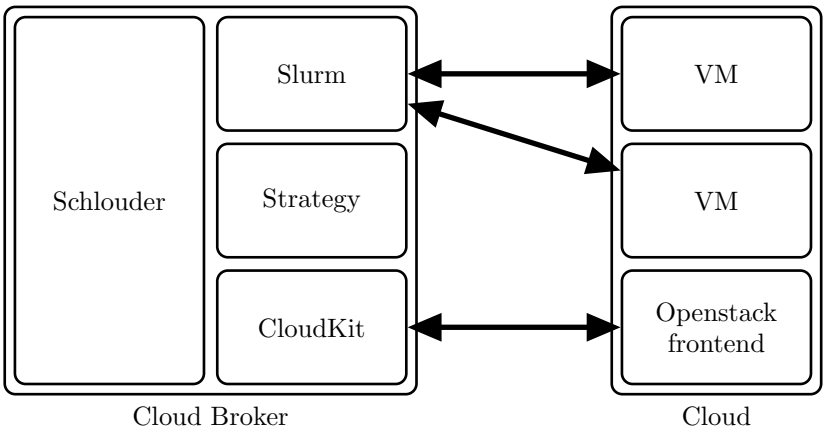


Figure 1: The Schlouder cloud broker and its components relating to an Openstack cloud.

IaaS, Schlouder views provisioning in fixed increments of time, referred to as **btu!** (**btu!**), machines idle when arriving at the end of their time will be automatically shut off.

Tasks execution is monitored using SLURM[**?**] which connects to the available **vm!**s, send the jobs, and monitors their executions.

A strategy controls provisioning and scheduling decisions, it determines when **vm!** provisioned and to which **vm!** each task is assigned. Schlouder provides a few basic strategies and new ones can be added trivially. Strategies can affect the workload's *makespan*, the **vm!**s usage rate, and the number of opened **btu!**s(i.e. the price of the experiment on a commercial cloud). The experiments in our archive mainly use two strategies, **asap!** (**asap!**) and **afap!** (**afap!**)

asap! will attempt to execute tasks as early as possible by booting a new **vm!** unless a **vm!** is already idle or one is predicted to become idle faster than a **vm!** can boot.

afap! will favor scheduling tasks on already opened **vm!** unless doing is predicted extend the **vm!** runtime by an additional **btu!**, in which case it will boot a new **vm!**.

Predicted tasks runtimes, as well as tasks dependencies, are user-provided. Scheduling is done dynamically, as soon as tasks are submitted and all their dependencies have been completed. Once assigned to a designated **vm!** jobs are not rescheduled. However, if a **vm!** fails to boot Schlouder will provision a new one to replace it.

3.1.2 Use cases: Applications characteristics

Two test-case applications where submitted to Schlouder in order to cover a variety of application profiles in terms of computation intensity, data load, and task dependency.

3.1.3 OMSSA

The **omssa!** (**omssa!**)[**?**] comes from the field of biology, it is used in tandem mass spectrometry analysis (also known as MS/MS analysis) to identify peptides from the mass and fragment ions obtained by a mass spectrometer. **omssa!** matches measurements from the mass spectrometer, called spectra, to a protein database.

The **omssa!** workload features fully independent tasks, making it **bot!** (**bot!**), since every spectra within a set can be submitted independently to **omssa!**. With a *communication-to-computation* ratio comprised between 20% and % **omssa!** is considered an CPU-intensive workload.

This application was run with 4 different workload covering 2 different mass spectrometer resolutions of two different protein solutions, denoted *brs,hrs,brt,hrt*.

3.1.4 Montage

The Montage Astronomical Image Mosaic Engine[**?**] is designed to gather astronomical images into a mosaic. This application is a workflow designed to reproject, normalize, and collate source images into a single output image. The montage workflow is presented figure **??**.

Working on images Montage is an extremely data intensive workflow with a *communication-to-computation* ratio superior to 90%.

This application was run on images of the *Pleiade* star cluster at 3 different output size, 1X1, 2X2, and 3X3.

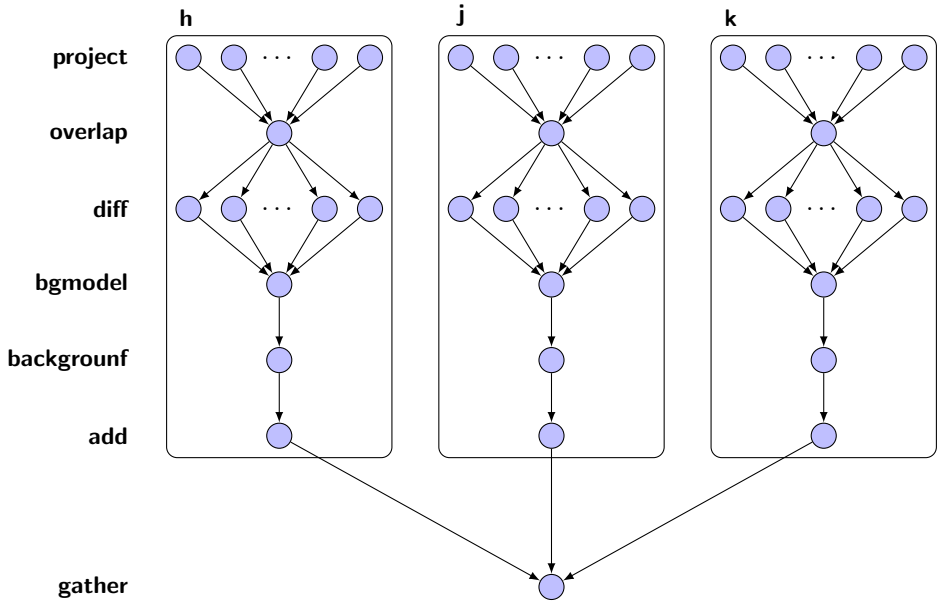


Figure 2: Illustration of the Montage workflow. Each node represents a tasks and each arc represents a data dependency between two tasks. Every task on a specific row run the Montage command indicated on the left hand side of the graph.

Cloud	#cores	Hypervisor	Network	version	#VM	Storage
openstack-icps	48	KVM	100mb	1-3	25	NFS
			1Gb	4-5	10	NFS
de-hlrs	344	Xen 3.1.2	n/a	v1-3	20	NFS
fr-inria	96	Xen 3.2	n/a	v1-3	20	NFS
uk-epcc	176	Xen 3.0.3	n/a	v1-2	20	NFS

Table 1: Characteristics of our cloud testbeds, version numbers also account for changes in measured boottimes not presented in this table.

3.1.5 Test beds

The application where executed on two different clouds : a private OpenStack based cloud, and BonFIRE[**?**], a public European cloud testbed.

3.1.6 Private Cloud

Our private cloud is based on two local nodes sporting dual $2.64GHz$ Intel Xeon processors (X5650), for a total of 96 cores. Nodes operated on Ubuntu 12.04 distributions and virtualisation is achieved using KVM. Openstack 2012.1.3 was used as a cloud interface. This cloud was build on perfectly homogeneous nodes and devoid of other users. For data storage these experiments rely on a **nfs!** (**nfs!**). Special attention was taken to not overbook **vm!**s by caping the number of single core **vm!**s to 25. Due to the experimental nature of this cloud configurations have change overtime. Table **??** regroups configurations of all cloud versions. We will referrer to this cloud as `openstack-icps.`*version*.

3.1.7 Public Cloud

BonFire[**?**] is a public multi-cloud distributed all over Europe. Our experiments were run over three sites, de-hlrs based in Stuttgart, uk-epcc in Edinburgh, and fr-inria in Rennes. BonFIRE clouds were accessed through an OCCI based API and the clouds were controlled throught software derived from OpenNebula 3.6 . Each site provided different hardware[1]. Resource quotas limited most experiment 20 **vm!**s, not far from the limits generally imposed on public clouds. Centralized storage was provided through a **nfs!** based on the be-ibbt site in Ghent. Due to network acces restriction the Schlouder server was brought in the BonFIRE WAN through a VPN. Due to the experimental nature of this cloud configurations have change overtime. Table **??** regroups configurations of all cloud versions. In this article we refer to experiment run on the BonFire clouds by the name of the cloud site followed by the version number.

3.2 Simulations

3.2.1 Simulator

SimSchlouder/SCHIaaS/Simgrid

SimSchlouder reproducing Schlouder runs. Same input/output file.

[1] Comprehensive information available at `http://www.bonfire-project.eu/infrastructure/testbeds`

3.2.2 Lab

3.2.3 Procedural Analysis

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.

3.2.4 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 $managementtime(t) = (inputting_t - submitted_t) + (complete_t - finished_t)$

4 Results

4.1 Definitions

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:

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

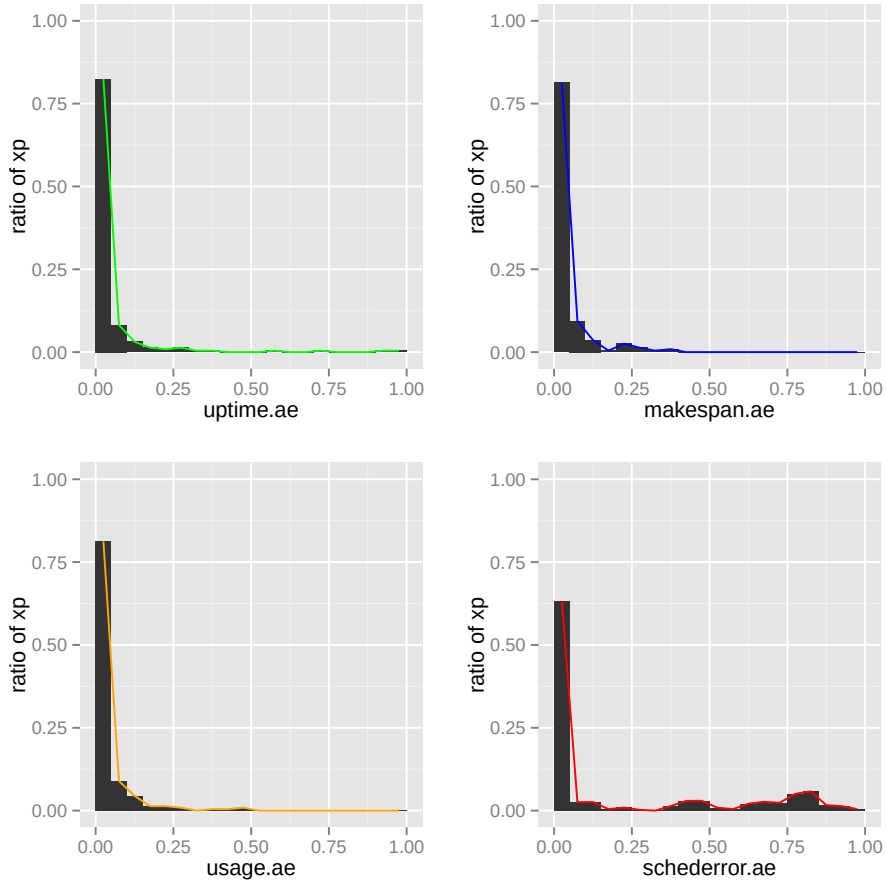
4.2 Simulator accuracy

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: 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



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.023	0.017	0.019	0.124
median	0.001	0.000	0.001	0.000
max	0.995	0.375	0.500	0.961

Figure 3: Frequencies and statistics about absolute error of best simulations (274 xp)

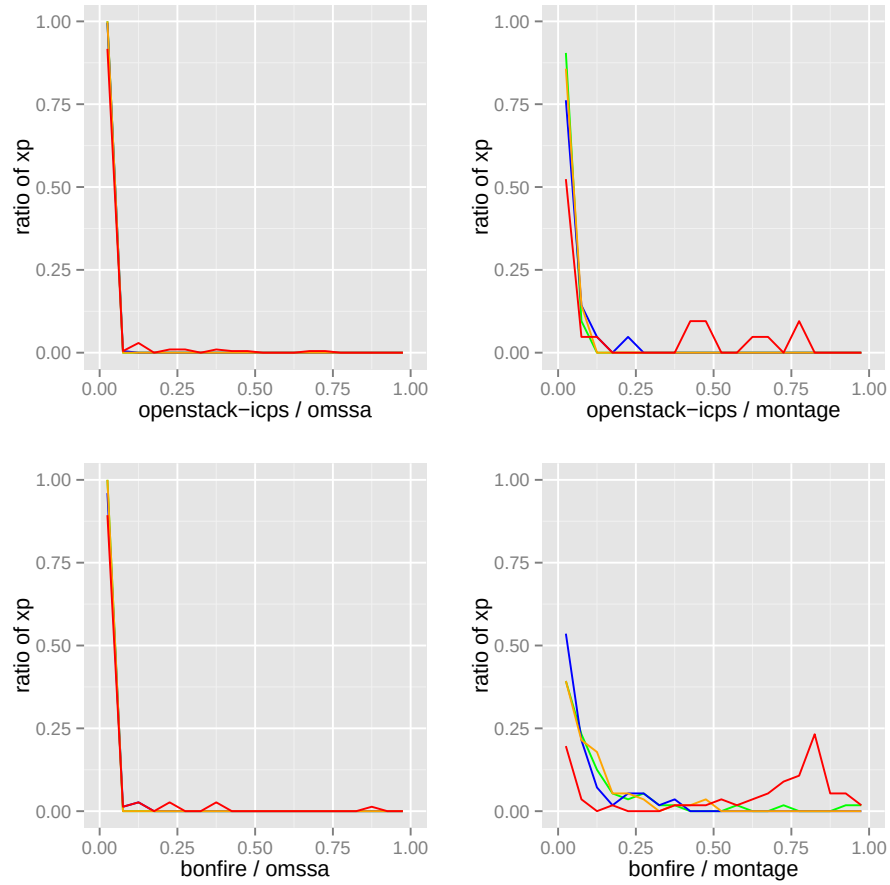


Figure 4: Absolute error frequencies of best simulations according to platforms and applications

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.

4.3 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	0.029	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.

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.

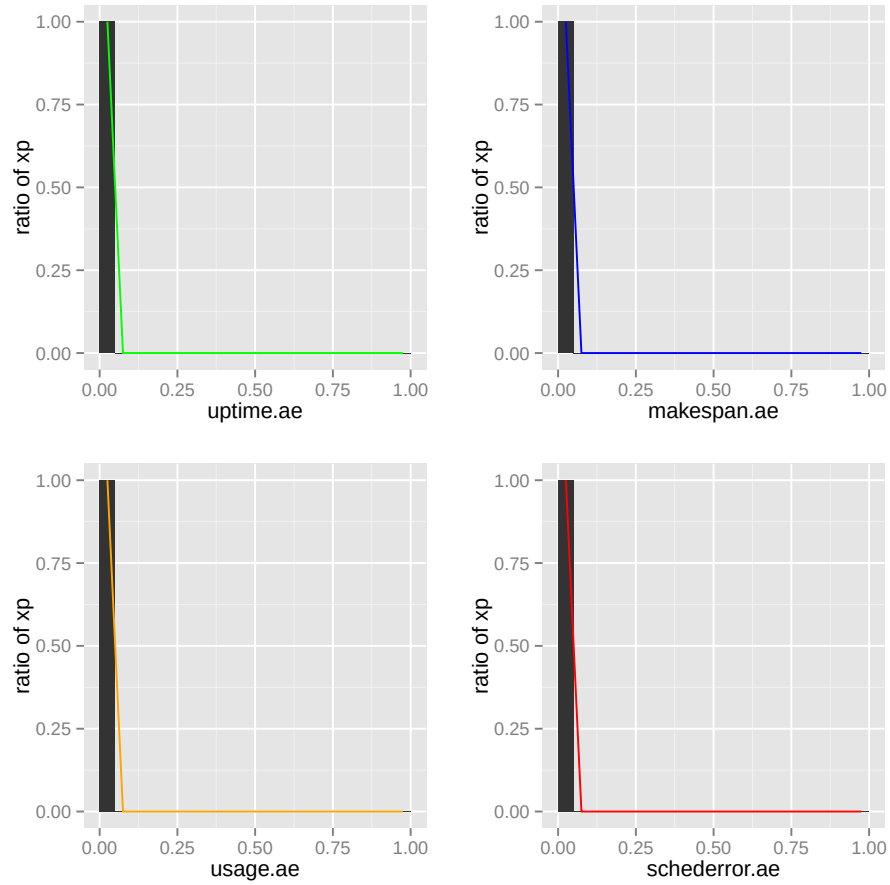
This shows that, providing that one can inject the right information, the only limitation of our simulator are micro clock-related hazards.

- openstack-icps / montage (36 xps):

	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.010	0.033	0.018	0.231
median	0.000	0.001	0.002	0.000
max	0.086	0.221	0.079	0.790

	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	0.010	+0.009	0.033	+0.032	0.018	+0.017	0.231	+0.207
median	0.000	+0.000	0.001	+0.001	0.002	+0.001	0.000	0.000
max	0.086	+0.056	0.221	+0.141	0.079	+0.050	0.790	+0.041

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.



	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	−0.018	0.000	−0.749

Figure 5: Frequencies and statistics about absolute error of best simulations for openstack-icps / ommssa, without scheduling error cases (91 xps)

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	0.045	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	0.045	+0.016	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	0.104	0.572
median	0.081	0.048	0.082	0.742
max	0.995	0.375	0.500	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	0.045	+0.016	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.

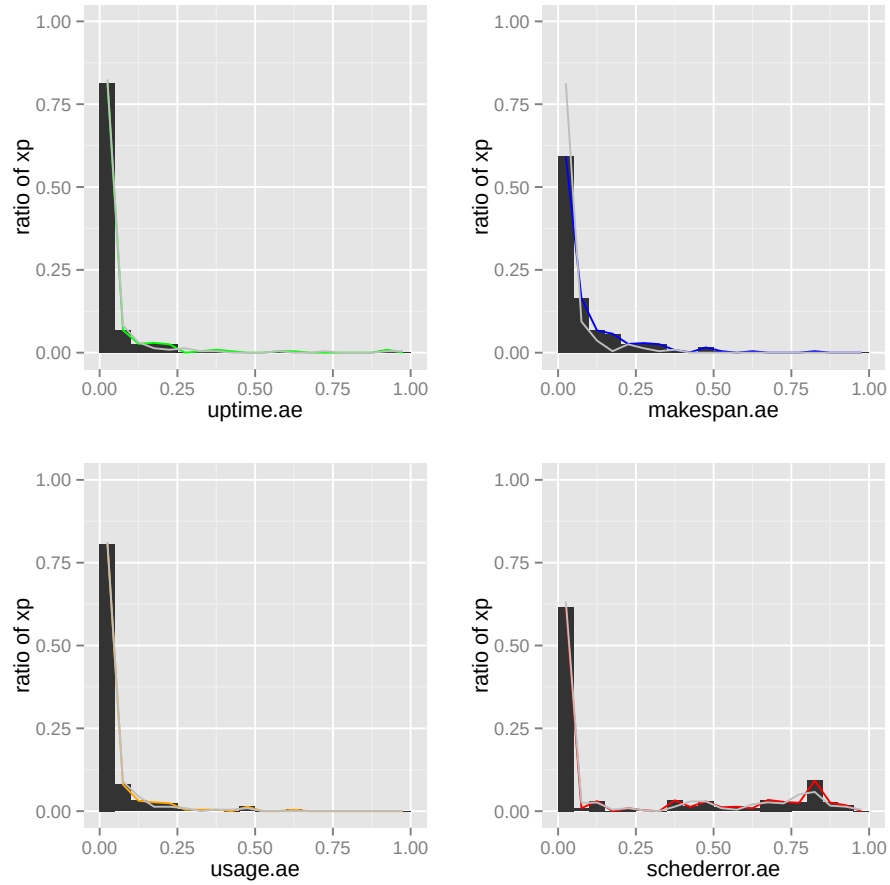
4.4 Boottime impacts

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.



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.026	0.059	0.028	0.134
median	0.001	0.008	0.002	0.000
max	0.918	0.815	1.711	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	0.026	+0.003	0.059	+0.042	0.028	+0.009	0.134	+0.010
median	0.001	+0.000	0.008	+0.007	0.002	+0.000	0.000	0.000
max	0.918	−0.077	0.815	+0.440	1.711	+1.211	0.949	−0.012

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.373	−0.164	−0.249	−0.288
mean	+0.003	+0.042	+0.009	+0.010
median	0.000	+0.004	0.000	0.000
max	+0.412	+0.807	+1.528	+0.583

Figure 6: Frequencies, statistics, and comparison with best of simulations with no real boot times injection

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.

4.4.1 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

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

Assess the impact of efficient communications

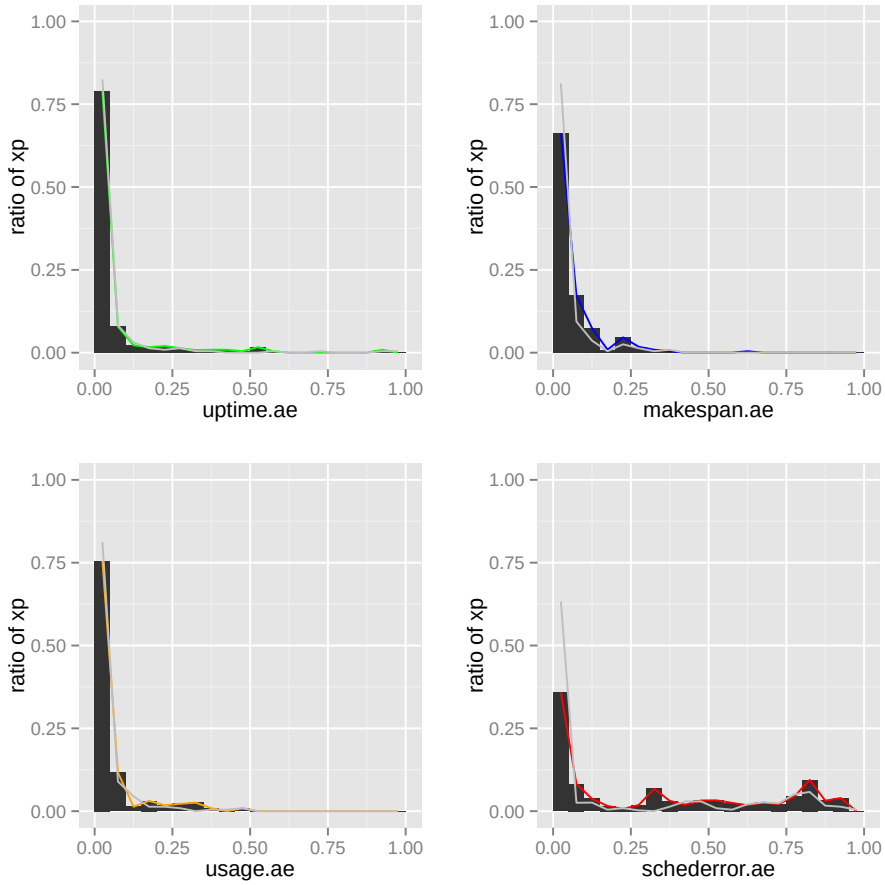
4.4.3 Prediction

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.

5 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
```

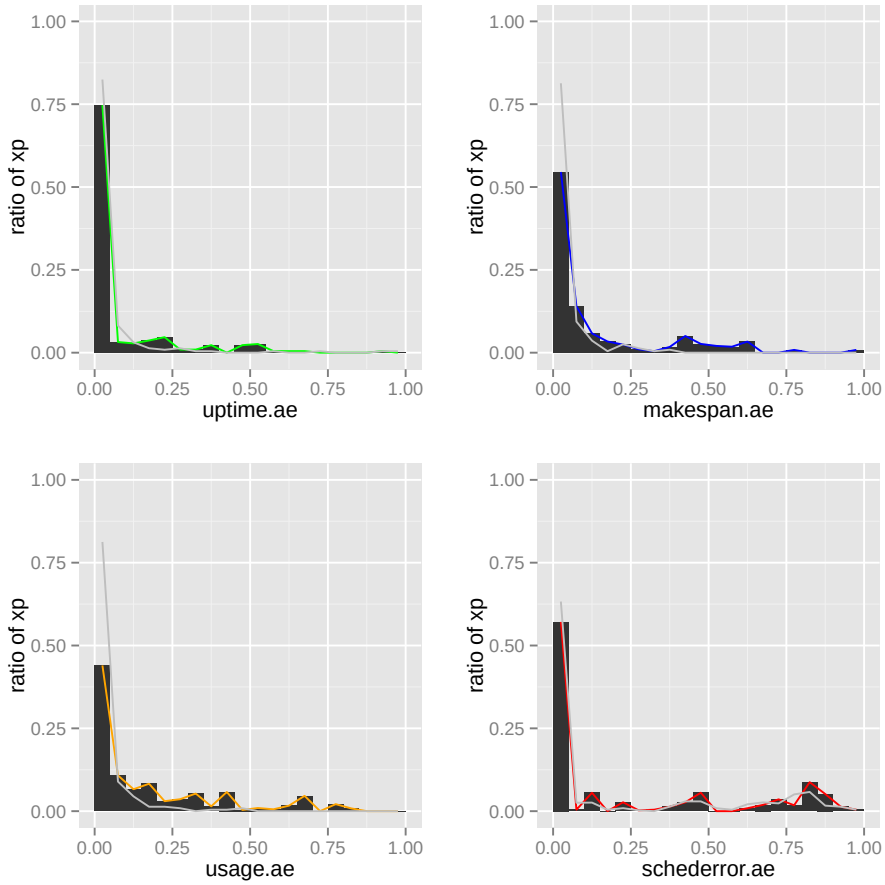


	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	0.000	0.000	0.000	0.000
mean	0.033	0.038	0.029	0.261
median	0.001	0.013	0.002	0.092
max	0.918	0.607	0.479	0.944

	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	0.033	+0.009	0.038	+0.021	0.029	+0.010	0.261	+0.137
median	0.001	+0.000	0.013	+0.013	0.002	+0.000	0.092	+0.092
max	0.918	−0.077	0.607	+0.231	0.479	−0.020	0.944	−0.017

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.251	−0.181	−0.143	−0.601
mean	+0.007	+0.020	+0.007	+0.138
median	0.000	+0.003	−0.000	+0.046
max	+0.403	+0.554	+0.288	+0.870

Figure 7: Frequencies, statistics, and comparison with best of simulations with no real thread times injection



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	-1.000	0.000	-26175.045	0.000
mean	0.063	0.078	-104.675	0.143
median	0.001	0.012	0.026	0.000
max	2.068	1.109	0.824	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	-26175.045	−26175.045	0.000	0.000
mean	0.063	+0.039	0.078	+0.061	-104.675	−104.694	0.143	+0.019
median	0.001	+0.000	0.012	+0.012	0.026	+0.025	0.000	0.000
max	2.068	+1.073	1.109	+0.734	0.824	+0.325	0.961	0.000

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.141	−0.211	−0.294	−0.426
mean	+0.044	+0.047	+0.073	+0.017
median	0.000	+0.003	+0.010	0.000
max	+2.015	+0.642	+0.761	+0.751

Figure 8: Frequencies, statistics, and comparison with best of simulations with simulation of communications

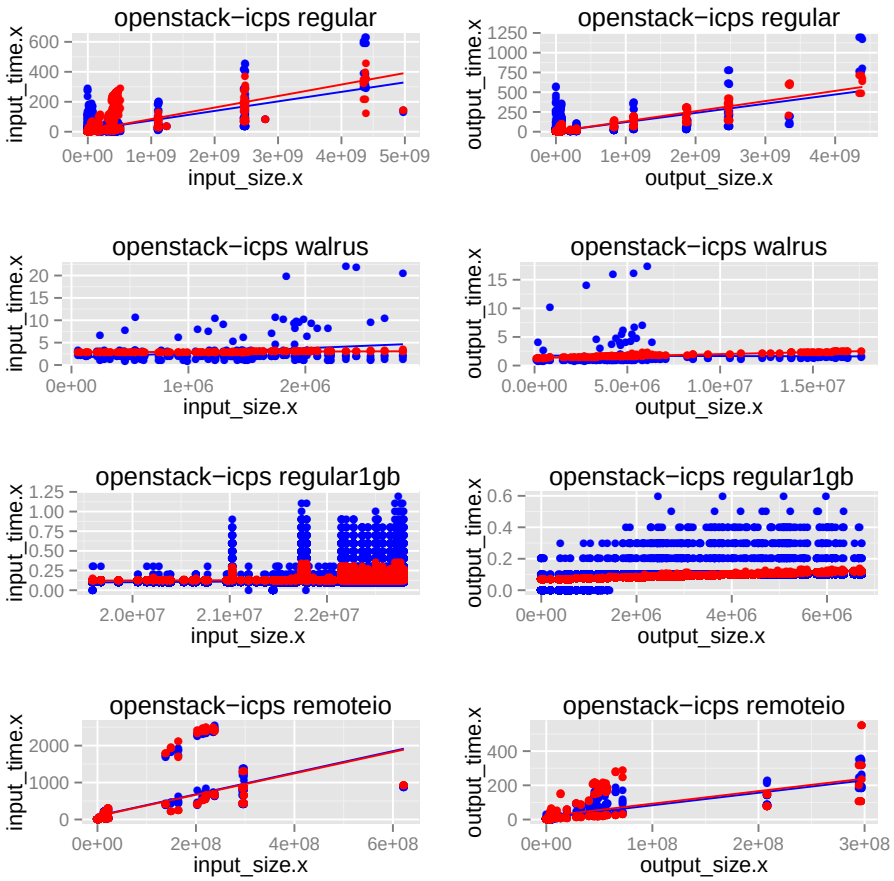


Figure 9: Linear regressions of communication times vs. data size, according to platform, storage, and communication direction on openstack-icps

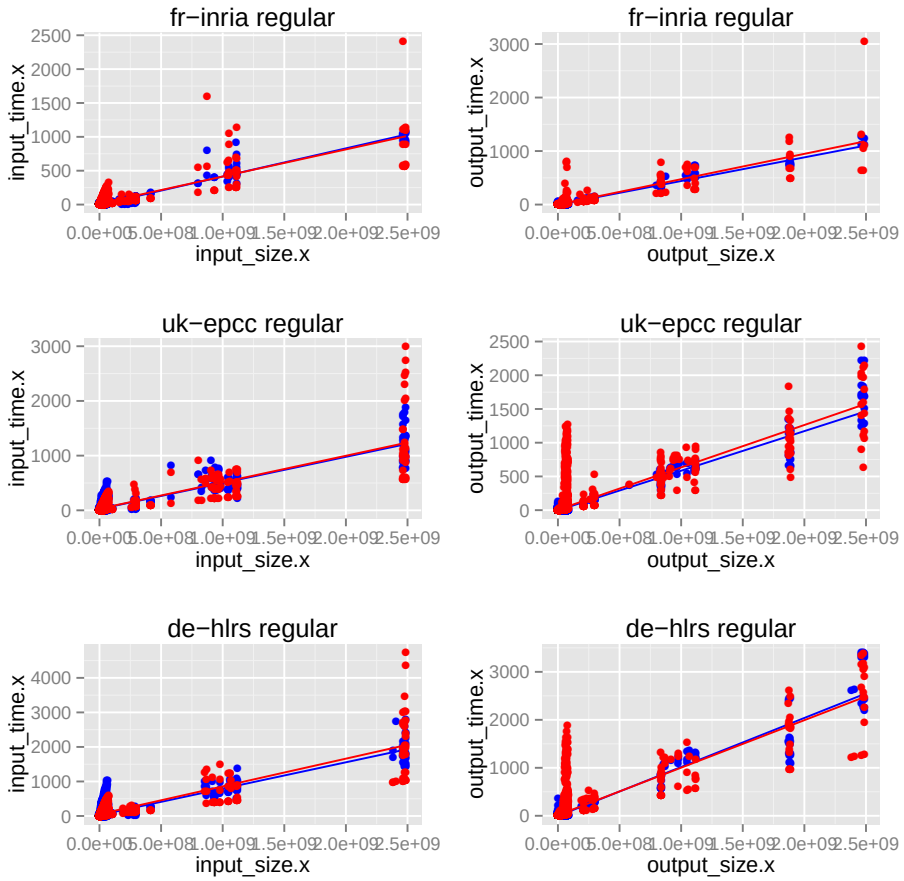
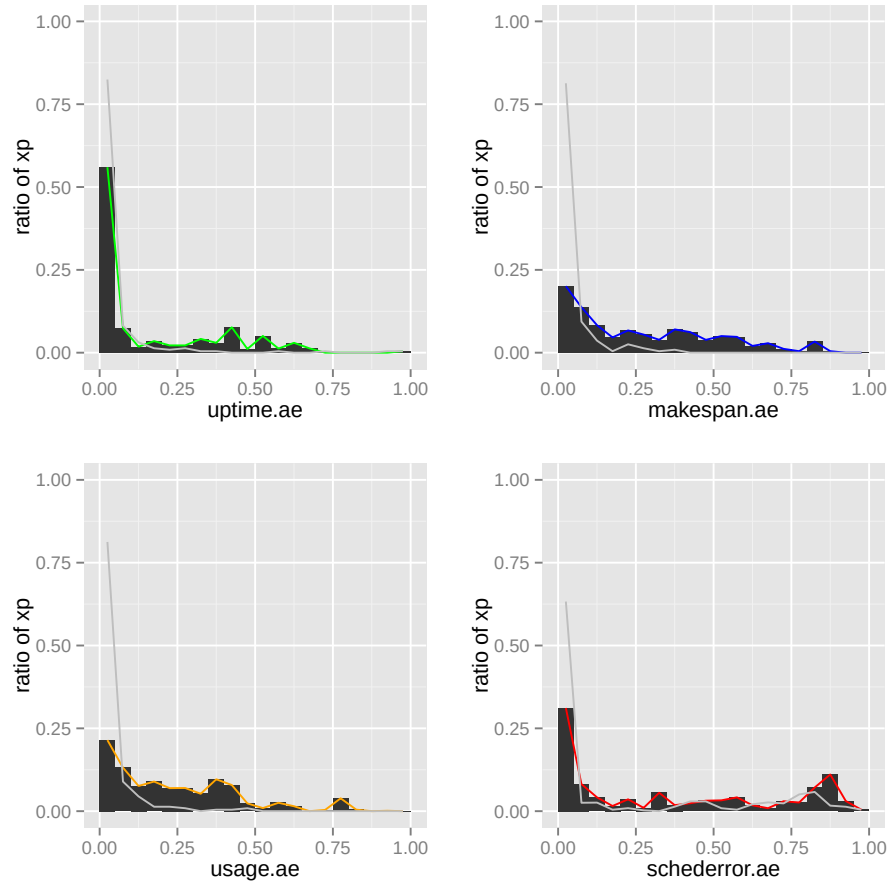


Figure 10: Linear regressions of communication times vs. data size, according to platform, storage, and communication direction on BonFire



	uptime.ae	makespan.ae	usage.ae	schederror.ae
min	-1.000	0.006	-27953.710	0.000
mean	0.119	0.212	-110.118	0.274
median	0.003	0.115	0.142	0.108
max	1.601	1.772	2.911	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	-27953.710	−27953.710	0.000	0.000
mean	0.119	+0.096	0.212	+0.195	-110.118	−110.137	0.274	+0.150
median	0.003	+0.002	0.115	+0.115	0.142	+0.141	0.108	+0.108
max	1.601	+0.606	1.772	+1.396	2.911	+2.411	0.965	+0.004

	$\Delta_{uptime.ae}$	$\Delta_{makespan.ae}$	$\Delta_{usage.ae}$	$\Delta_{schederror.ae}$
min	−0.141	−0.211	−0.294	−0.426
mean	+0.044	+0.047	+0.073	+0.017
median	0.000	+0.003	+0.010	0.000
max	+2.015	+0.642	+0.761	+0.751

Figure 11: Frequencies, statistics, and comparison with best of simulations with no injection