# Replay with Feedback: How does the performance of HPC system impact user submission behavior?

Maël Madon, Georges da Costa, Jean-Marc Pierson

Future Generation Computer Systems 155 (2024) 66–79

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Replay with Feedback: How does the performance of HPC system impact user submission behavior?

Maël Madon *, Georges Da Costa, Jean-Marc Pierson

*IRIT, University of Toulouse, CNRS, Toulouse INP, UT3, 118 route de Narbonne, Toulouse, 31062, France*

ARTICLE INFO

*Keywords:*
HPC simulation
User behavior
Parallel workload
Scheduling
Performance evaluation
Reproducibility

ABSTRACT

High Performance Computing (HPC) is a key infrastructure to solve large scale scientific problems, from weather to quantum simulations. Scheduling jobs in HPC infrastructures is complex due to their scale, the different behaviors of their users, and the multiple objectives, from performance to ecological impact. Schedulers are evaluated on data center simulations, due to the complexity and cost of evaluating them in-situ. One key element for this evaluation is the behavioral model of users. Most studies are limited to replaying past workload of existing data centers. This reduces the realism of performance evaluation in cases where the scheduler and the hardware infrastructure are not exactly the same. Any such change would potentially impact the behavior of the users.

In this article we introduce a novel model "Replay with Feedback" accounting for the impact of HPC system performances on user submission behavior in simulations. Instead of keeping the original timestamps of job submissions, we exhibit and use the relationships between each user jobs. We propose an open-source implementation of this model along with an extensive and reproducible set of experiments to assess the impact of the scheduler and infrastructure changes. We also provide new metrics adapted to the flexibility of user submission behaviors. Results show that using this model, we advance towards more realistic simulations of schedulers in HPC systems.

## 1. Introduction

When there is a need for intensive computations (e.g. machine learning training or fluid mechanics calculations), companies and researchers use big computer farms called High Performance Computing (HPC) infrastructures. Even if there exist many types, the principle remain the same: users submit computing jobs to the infrastructure through an interface and wait for the results. On the infrastructure side, a scheduler takes care of collecting the requests and selects when and where (on which machines) they will be executed. HPC infrastructures have greatly evolved in the past decades, thanks to progress in hardware and software. Simulations of these infrastructures have developed in parallel, e.g. SimGrid [1] or Gridsim [2], with the aim to reproduce their behavior as precisely as possible and provide tools to compare different schedulers or infrastructure designs.

An HPC simulation needs at least two kinds of inputs: a description of the simulated infrastructure and a workload, i.e. which jobs are submitted and when. Its purpose is to simulate how the workload would be scheduled in the infrastructure. By experimenting with different infrastructures, workloads or scheduling strategies, the researcher gain knowledge on which one is the best to optimize for certain objectives like throughput (quantity of jobs per unit of time), waiting time for the users or energy consumed.

This paper focuses on the workload. How do we accurately model the arrival of jobs in the infrastructure? Traditionally, we encounter two main approaches: (i) using records from real infrastructures to replay the jobs submissions (a collection of such records is available for example in the Parallel Workload Archive) or (ii) generating synthetic workloads (e.g., following probability laws [3]). In both methods, the workload is fully determined before the simulation. It means in particular that *the jobs submission times are known in advance*. Only rarely do the simulations include a feedback loop, allowing the job arrivals to adapt to what happens in the simulated system.

While this makes the workload model simple and the results easy to compare, we argue that this is too strong a hypothesis leading to unreliable results. In fact, in reality, HPC users adapt their submission behavior to the feedback they get from the infrastructure. For example, imagine a real infrastructure that suddenly becomes twice slower due to a breakdown. The jobs will start accumulating in the queues and the users, seeing that their previous submissions are still pending, will slow down their rate of submission. On the contrary, if the infrastructure gets

* Corresponding author.
*E-mail addresses:* mael.madon@irit.fr (M. Madon), georges.da-dosta@irit.fr (G. Da Costa), jean-marc.pierson@irit.fr (J.-M. Pierson).

https://doi.org/10.1016/j.future.2024.01.024
Received 22 September 2023; Received in revised form 10 January 2024; Accepted 20 January 2024
Available online 29 January 2024

faster due to a more efficient scheduling or the addition of new nodes, the users will tend to submit more and bigger jobs, a phenomenon known as "rebound effect". The pattern of job arrival is in fact tightly linked to the specific infrastructure in which it is observed and is the fruit of interaction between its users and the scheduling algorithm. Consequently, the results obtained by replaying a historical record in a simulation in which a change in the infrastructure or scheduler is introduced will not reflect the reality.

This problem was identified by Dror Feitelson in the first half of the 2010s. He proposed with Netanel Zakay some methods of "replay with feedback" that account for user reaction to system performance [4]. This paper builds upon their work and addresses the following question:

*Is replay with feedback satisfying to simulate a change in the infrastructure or scheduler?*

The main contributions are:

- a definition of "replay with feedback", a way of accounting for the impact of HPC system performances on user submission behavior in simulations,
- a novel model of replay with feedback based on dependencies and "think times" between jobs,
- two open-source and easily customizable software to implement replay with feedback: **batmen**, a C++ plugin for the data center simulator Batsim, and **swf2userSessions**, a Python script to handle input logs,
- a reproducible experimental campaign comparing traditional replay and replay with feedback, and
- three original metrics for the analysis of the results: *mean lateness*, *relative lateness* and *additional lateness*.

The remaining of this paper is organized as follows. First, we discuss the related works in Section 2. Then, we introduce in detail our model for replay with feedback in Sections 3 and 4. The experimental campaign is described in Section 5, along with preliminary results. After that, we introduce new metrics in Section 6 that we use for further analysis. Finally, we provide a discussion of our approach in Section 7 before concluding and providing ideas for future work.

## 2. Related works

The method of traditional replay, where the job arrivals are fixed in advance, is still the most widely used simulation method for performance evaluation in HPC-like system. It is used for example in diverse recent works like Vasconcelos et al. using a synthetic workload to study distributed cloud federation [5], Wiesner et al. using a recorded workload from a production system to evaluate their renewable-aware scheduler [6] or de Nardin et al. using logs from an academic data center [7].

However, this model has been criticized in the literature [4,8], because it does not take into account how users react to the performances of the system. They recommend doing *closed-loop* simulation instead [9], also called 'dynamic' or 'feedback-aware' simulations. For this, some modeling of user submission behavior is needed.

### 2.1. Modeling HPC user behavior

Interesting insights on the ways to model user submission behavior can be found in the rich literature on workload analysis and forecasting [10,11]. For example, Dinh et al. propose a method of load prediction based on passed submissions and inter-arrival time distribution [12].

Some works provide extensive analysis of HPC logs in order to get a deep understanding of user characteristics, such as waiting time, number of cores requested, inter-arrival time, walltime accuracy etc. [13]. An interesting article [14] also mixes these analyses with methods from social sciences: an analysis of help tickets, a survey and interviews with users.

However, in these works, the angle is fundamentally different: what one tries to model or predict is the *load submitted*, and not the way users react to the termination (or non-termination) of their jobs.

In the remaining of this section, we will focus on literature on closed-loop simulation for HPC systems.

### 2.2. Replay with feedback

As mentioned in the Introduction, introducing a feedback loop inside data center simulations was first suggested by Zakay and Feitelson. They follow a two-step approach: (i) extracting the relevant patterns of user submission behavior from recorded workloads [15], and (ii) using this information to replay users reaction to feedback inside the simulations [4]. Step (i) provides them with a list of *sessions of submission* for each user, with precedence relations between them. They compare several methods, based on think time (time elapsed between the termination of a job and the submission of another) or inter-arrival time between jobs (see 3.1). They propose for step (ii) three different ways to use the sessions and their precedence relations for replay: 'adjusted', 'distribution-based' or 'fluid' user model. We will come back in detail on these methods in Section 7.5. Regrettably, we could find no code available to reproduce their results, nor detail on the simulation software used. This paper is based on inter-arrival for step (i) and the 'adjusted' approach for step (ii). Differences with their model will be pointed out along the way.

To the best of our knowledge, replay with feedback is only used once more in the literature, by Klusáček et al. They implemented Zakay and Feitelson's 'adjusted' model inside the open source simulator Alea [16]. The feature is called "dynamic workload adaptation" and was notably used to test different schedulers before their deployment in a production system [17]. However, the model in their work is only used as a tool, and they provide no evaluation nor in-depth discussion on its effect on the simulation outcome.

### 2.3. Generative simulation

Going one step further from replay with feedback, we can find a few works proposing "generative simulation" as a method for closed-loop simulation. Generative simulation consists in a statistical analysis of the input workload in order to retrieve some patterns and probabilistic laws. This knowledge is then used dynamically during the simulation, to generate incoming workload.

Zakay and Feitelson propose a method of "resampling with feedback" [18], featuring a model of population of users, whose arrivals and departures depend on user satisfaction. Workload of individual users are sampled from the original workload, and replayed with feedback, as described before [4]. Similarly, Schlagkamp proposes a parametric user model, with weekly and daily activity windows for users, fitted to the observations from the input log [8]. The replay is based on think times, that are calculated as a linear function of the response time of the previously terminated job.

These models can be powerful, but they are complex and largely understudied. Once again, we are not aware of any available implementation allowing for comparison and incremental improvement. We try to fill this gap with the open-source tools released with this paper.

## 3. Workload model

This section and the next one provide a formal definition of our model.

### 3.1. Recorded workload trace and session partitioning

A recorded workload trace on an infrastructure is the log of all the jobs that were submitted by users and executed in this infrastructure during a certain time window. To simplify the notations, all the definitions below are given for a certain user $u$.

The smallest record in the trace is the *job*. It is completely defined by

- an execution time $d$ (for 'duration'),
- a number $r \in \mathbb{N}^{+*}$ of requested resources,
- a submission time $a$ (for 'arrival'), which is the time at which the user submitted the job in the infrastructure,
- a finish time $f$, and
- a walltime $w$, which is an upper bound on execution time given by the user.

The time at which the job started in the infrastructure can be obtained by $f - d$.

**Definition 1.** A **recorded job** $j_i$ is defined by the tuple $(d_i, r_i, a_i, f_i, w_i)$.

**Definition 2.** A **recorded trace** is a list $(j_1, \dots, j_n)$ of recorded jobs, ordered by submission time.

In previous work, Zakay and Feitelson argue that meaningful components of a recorded trace are *user sessions*, which they define conceptually as "periods of continuous work by a user" [15]. In other words, a user has sessions of work in which she interacts with the infrastructure – mainly by submitting new jobs – and periods of absence. With our notation:

**Definition 3.** A **user session** in the recorded trace $\mathcal{T}$ is a list $(j_n, j_{n+1} \dots, j_m) \subset \mathcal{T}$ of consecutive recorded jobs, with $(n, m) \in \mathbb{N}^2$ and $n \leq m$.

Unfortunately, we rarely have metadata on user activity associated to the recorded trace to help us to detect the session boundaries. Hence, we will infer them from the information contained in the recorded trace. We call this operation *session partitioning*:

**Definition 4.** A **session partition** of the recorded trace $\mathcal{T}$ is a set $\{s_1, \dots, s_m\}$ of sessions such that $\forall (i,k), s_i \cap s_k = \emptyset$ and $s_1 \cup \dots \cup s_m = \mathcal{T}$.

There are many possible ways to do this partitioning based on the information at hand in the recorded trace. The simplest is to consider that each job is a session. Another way is to cut into sessions based on thresholds on inter-arrival time between jobs, like in Fig. 1. We refer to the original paper of Zakay and Feitelson [15] for the proposition and comparison of three such methods, and our open source tool *swf2userSessions*[1] in which they are implemented.

### 3.2. Session graph

We suppose that we have a session partitioning of a recorded trace $\mathcal{T}$. We define between sessions the partial order *depends on*:

**Definition 5.** For two sessions $s = (j_n, \dots, j_m) \subset \mathcal{T}$ and $s' = (j_{n'}, \dots, j_{m'}) \subset \mathcal{T}$, we say that $s'$ **depends on** $s$ if all the recorded jobs of $s$ finished their execution before the first job of $s'$ was submitted, i.e., if $max_{n \leq i \leq m}(f_i) \leq a_{n'}$

[1] Python script performing the session partitioning of a recorded trace in the Standard Workload Archive format, available at gitlab.irit.fr/sepia-pub/mael/swf2userSessions. (the specific version tagged `replay_feedback2023` is used in this article).

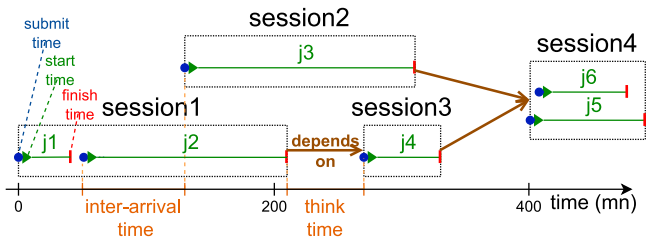



**Fig. 1.** Illustration of a session graph with six jobs and four sessions. Here, an inter-arrival time greater than 60 min between two jobs delimits a new session: *j1 and j2 are in the same session since* $a_2 - a_1 < 60$ mn, *while j3 is in a different session since* $a_3 - a_2 \geq 60$ mn.

From this definition follows our definition of *think time* between sessions. Conceptually, it corresponds to the time that a user had to think between the termination of all the jobs in a session and the submission of the first job of another:

**Definition 6.** For two sessions $s = (j_n, \dots, j_m) \subset \mathcal{T}$ and $s' = (j_{n'}, \dots, j_{m'}) \subset \mathcal{T}$, we call **think time** the quantity $a_{n'} - max_{n \leq i \leq m}(f_i)$

**Observation 1.** *By definition of the relation depends on, the think time between two depending sessions is always* $\geq 0$

As a result, the set of sessions for a user forms a *weighted directed acyclic graph*, where the nodes are the sessions and the edges represent the relation *depends on*, weighted by the corresponding think time. See Fig. 1 for illustration.

In this representation, some sessions have no predecessor: sometimes only the first session and sometimes more (like session1 and session2 in the illustration). We make the graph connected by adding a fictive session at the root of the graph and making all the sessions that do not have a predecessor dependent on it. For each edge added that way, we choose the recorded starting time of the session as *think time*, i.e., the submission time of the first recorded job of this session. The resulting session graph contains all the information needed for the replay.

## 4. Replay with feedback

In this section, we provide a definition of replay with feedback, then describe our method of replay using the session graphs previously described.

### 4.1. Feedback and rigid: two paradigms of replay

Replay with feedback is a new paradigm of using recorded workload data in simulations:

**Definition 7.** **Replay with feedback** is a way of using a recorded workload in simulations to mimic the platform activity while accounting for user reactions to simulated system performance.

In practice, users adapt to feedback in many ways. They change their dates of submission, submit bigger or smaller jobs, modify their software to fit the infrastructure or even leave the infrastructure to submit somewhere else. Taking into account all these behaviors in the replay is a very challenging task, and can potentially modify the workload significantly. For this reason, *we consider in this work only one type of user response, namely changes in submission times.*

To make a distinction between the *recorded* jobs and their simulated copy, we call the latter *replay* jobs and denote them $\hat{j}_i = (\hat{d}_i, \hat{r}_i, \hat{a}_i, \hat{f}_i, \hat{w}_i)$. The type of replay with feedback performed here preserves the jobs

characteristics: mass of computation to perform, number of resources and walltime. With the notation:

$$\forall 1 \leq i \leq n, \begin{cases} \hat{d}_i * \hat{r}_i * \hat{P} &= d_i * r_i * P \\ \hat{r}_i &= r_i \\ \hat{w}_i &= w_i \end{cases}$$

where $P$ (resp. $\hat{P}$) is the performance of the node in the original (resp. simulated) infrastructure, in floating-point operations per second.

The traditional way of replaying jobs in simulations would also preserve the submission times, i.e., $\hat{a}_i = a_i$. We will denote it "rigid replay":

**Definition 8.** **Rigid replay** simulates the arrival of jobs with the same characteristics and same submission time as in the recorded workload.

### 4.2. Replay based on think times

The main idea behind our replay method is that it preserves the *think time between sessions* rather than the exact submission times of the jobs, thus reacting to the feedback provided by the (simulated) infrastructure. For example, if a job inside a session takes longer to finish in the simulation compared to the recorded trace, the following sessions in the session graph will be delayed accordingly. Jobs *within* a session, however, are neither delayed nor brought forward in reaction to feedback in our algorithm.[2] Consequently, all the information needed for the replay with feedback are embedded in the session graph of each user.

We say that a session **starts** when its first (replay) job is submitted. A session **finishes** when the last of its jobs finishes its execution. Without loss of information, we represent the submission times $\hat{a}_i$ of the replay jobs *relatively* to the start time of their session, i.e.,

$$\forall s = (j_n, \ldots, j_m) \subset \mathcal{T}, \forall i \in \{n, n+1, \ldots, m\}, \hat{a}_i = a_i - a_n$$

Before going on with the description of the replay method, we need to introduce two additional definitions:

**Definition 9.** A session $\hat{s} = (\hat{j_n}, \ldots, \hat{j_m})$ is **active**, at time $t$, if $\hat{s}$ has started and $t_{\hat{s}} \leq t < t_{\hat{s}} + \hat{a_m}$, with $t_{\hat{s}}$ the starting time of $\hat{s}$.

Conceptually, $\hat{s}$ is active when the user is currently submitting from it.

**Definition 10.** A session $\hat{s}$ is **free**, at time $t$, if it has not started and all the sessions it depends on have finished, i.e.,

$$\begin{cases} t < t_{\hat{s}} \\ \forall \hat{s'} = (\hat{j_n}, \ldots, \hat{j_m}), \hat{s} depends on \hat{s'} \implies max_{n \leq i \leq m}(\hat{f}_i) \leq t \end{cases}$$

If $\hat{s}$ has not started but at least one session it depends on has not finished, we say that $\hat{s}$ is **dependent**.

The usual lifecycle of a session is then to be first dependent, then free, then active.

### 4.3. Replay method

We can now proceed to the explanation of the replay method. The method is combined with a discrete event simulation, in charge of simulating the platform and job scheduler. In the course of the simulation, we will traverse the session graph for each user, by keeping track of

1. the list $\mathcal{A}$ of active sessions
2. the list $\mathcal{F}$ of free sessions

At the beginning of the simulation ($t = 0$), $\mathcal{A}$ is empty and $\mathcal{F}$ contains the successors of the fictive root session. The replay method consists of two functions called in reaction to two different events: `wake_on_feedback`, when a job terminates, and `job_to_submit`, when it's time to submit a job. These functions are given as pseudocode in Algorithm 1.

Note that our replay method does *not* necessarily preserve the original submission order of jobs. For example in Fig. 1, if j2 finishes earlier in the replay, j4 might be submitted before j3.

## 5. Experimental comparison of feedback and rigid replay

In this section, we compare experimentally the results obtained with replay with feedback (Definition 7) and rigid replay (Definition 8).

### 5.1. Simulation inputs

As inputs for the simulations, we use two historical logs retrieved from the Parallel Workload Archive:

- KTH-SP2 (file `KTH-SP2-1996-2.1-cln.swf`): 11-month log from a 100-node IBM SP2
- SDSCSP2 (file `SDSC-SP2-1998-4.swf`): 24-month log from a 128-node IBM SP2

KTH and SDSC logs contain respectively 28 475 and 67 667 jobs, for 214 (resp. 428) users. The submission log for each user was converted to session graphs, as explained in Section 3. We made the *session partitioning* based on a threshold on inter-arrival time. We tried two values for this threshold: 0 min ('arrival 0', in short '**a0**') and 60 min ('**a60**'). a0 gives sessions of only one job. Doing a replay with this delimitation is equivalent to *preserve the think time between jobs only*. We chose the other threshold of 60 min because it is the value used in the original paper [15]. The influence of this parameter will be discussed in Section 7.2.

### 5.2. Experimental setup

The simulations are run with Batsim,[3] an open-source infrastructure and resource management system simulator based on SimGrid[4] [19].

The replay with feedback model described in Section 4 is implemented in our C++ plugin Batmen[5] (classes `FeedbackUser` and `FBUserThinkTimeOnly`) and available for download and evaluation. The scheduler is also implemented in Batmen. From the information we could find online [20], IBM SP2 systems seem to be using some version of EASY-backfilling algorithm[6] for scheduling. For this reason and unless specified otherwise, we use such a scheduler in our experiments (class `EasyBackfillingFast`), called 'EASY' in the remaining of this paper.

We design an experimental campaign to answer the question raised in Introduction (Section 1): *is replay with feedback satisfying to simulate a change in the infrastructure or scheduler?* A change in the infrastructure can be a change in the number of nodes, node performance, interconnection, bandwidth etc. Whichever the change, the outcome will be that

[2] This is different from Zakay and Feitelson, who introduce the notion of 'batches' within a session, which are groups of overlapping jobs. The relation *depends on* and the shifts during replay are defined for the batches.

[3] Batsim v4.2: batsim.org.

[4] SimGrid v3.32: simgrid.org, with `ptask_L07` model.

[5] Batmen repository: gitlab.irit.fr/sepia-pub/mael/batmen (the specific version tagged `replay_feedback2023` is used in this article).

[6] EASY-backfilling: sort the jobs by *submission time* in the queue of waiting jobs. When the first job in the queue cannot be immediately executed, backfill with jobs that have a walltime lower than the expected start time of that job.

**Algorithm 1** Replay method

```
function WAKE_ON_FEEDBACK                                   ▷ in reaction to the termination of a job
    for all j ∈ {jobs finished recently} do
        s ← session(j)
        if j was the last job of s to finish then
            for all s′ ∈ {successors of s} do
                dependencies(s′).pop(s)
                if dependencies(s′) is empty then
                    F.add(s′)                               ▷ definition of free session
function JOBS_TO_SUBMIT                                     ▷ when it's time to submit one job (or more)
    for all s ∈ F do
        if s starts now then
            A.add(s)                                        ▷ definition of active session
            F.pop(s)
    for all s ∈ A do
        submit all the jobs in job_list(s) with submission_time = now
        if job_list(s) is empty then
            A.pop(s)                                        ▷ definition of active session
```

**Table 1**
Scheduling metrics calculated on the recorded log and for all the experiments. Makespan and waiting times are expressed in days, with 2 decimal places. For KTH infra ∗ 2, we read mean waiting times of 0.00 day. This is because of rounding, and these values are actually between 162 and 229 s.

| Exp. name | Replay | *KTH* | | | *SDSC* | | |
|---|---|---|---|---|---|---|---|
| | | Makespan | Waiting time | | Makespan | Waiting time | |
| | | | Mean | Max | | Mean | Max |
| Recorded log | / | 332.93 | 0.18 | 11.34 | 736.12 | 0.26 | 62.48 |
| EASY | Rigid | 332.91 | 0.07 | 4.07 | 731.36 | 0.19 | 5.73 |
| | a0 | 366.14 | 0.06 | 5.06 | 808.88 | 0.14 | 5.90 |
| | a60 | 366.67 | 0.07 | 6.11 | 789.77 | 0.18 | 5.16 |
| FCFS | Rigid | 333.10 | 4.51 | 11.79 | 794.26 | 14.82 | 63.96 |
| | a0 | 457.89 | 0.29 | 4.95 | 1200.10 | 0.58 | 6.26 |
| | a60 | 454.41 | 0.47 | 4.47 | 1065.66 | 0.88 | 5.51 |
| perf ∗ 2 | Rigid | 332.91 | 0.01 | 1.34 | 731.32 | 0.01 | 1.84 |
| | a0 | 332.57 | 0.01 | 1.82 | 730.31 | 0.01 | 1.58 |
| | a60 | 332.61 | 0.01 | 1.44 | 729.82 | 0.02 | 1.13 |
| perf/2 | Rigid | 471.85 | 31.84 | 141.34 | 1239.37 | 64.62 | 508.38 |
| | a0 | 635.97 | 0.46 | 10.70 | 1506.26 | 0.92 | 15.54 |
| | a60 | 630.28 | 0.62 | 10.26 | 1492.67 | 1.61 | 14.17 |
| infra ∗ 2 | Rigid | 332.91 | 0.00 | 0.54 | 731.36 | 0.01 | 1.28 |
| | a0 | 332.63 | 0.00 | 0.81 | 729.81 | 0.01 | 1.04 |
| | a60 | 332.65 | 0.00 | 0.56 | 730.02 | 0.01 | 1.35 |
| infra/2 | Rigid | 386.70 | 4.15 | 58.87 | 1167.94 | 37.43 | 437.28 |
| | a0 | 472.93 | 0.27 | 7.43 | 1452.31 | 0.80 | 14.82 |
| | a60 | 472.45 | 0.35 | 7.31 | 1446.13 | 1.20 | 15.93 |

jobs execute faster or slower. Since the only feedback that matters to users in our replay method is the finish time of their jobs, we consider sufficient in this study to focus on two types of infrastructure change: number of nodes and node performance. Consequently, jobs in the simulation are represented as compute-only (parallel_homogeneous in Batsim), without communication.

*Experimental campaign.* We run the workload several times, varying the scheduler and hardware infrastructure (6 different cases):

- **easy**: the baseline experiment, with EASY scheduler and a simulated platform representing the original infrastructure
- **perf ∗ 2, perf/2**: multiplying or dividing by two the *performances* of the nodes, in terms of floating-point operations per second, i.e., the jobs are executed twice (resp. half) as fast
- **infra ∗ 2, infra/2**: multiplying or dividing by two the *number* of nodes, e.g., for KTH log we tried with 200 (resp. 50) nodes
- **FCFS**: changing the scheduling algorithm to First Come First Serve

Each instance is run *with* the feedback model (a0 and a60), and *without* (rigid). In total, $6 * 3 = 18$ simulations are run for each log.

*Reproducibility.* All the experimental details and material to reproduce the graphs presented in this paper are provided in forms of two notebooks.[7] Running the two notebooks on a recent laptop (Intel i5 11th gen) takes less than one hour, including downloading and processing the inputs, running the simulations and plotting the graphs.

### 5.3. Results

The results of the simulations consist in a complete record $(\hat{a}_i, \hat{f}_i - d_i, \hat{f}_i)$ of the timestamps of submission, start and finish time for each job $\hat{j}_i$. From these records, we compute several scheduling metrics like makespan or waiting times, as defined below.

**Definition 11.** The **makespan** is the time that elapses between the submission of the first job and the completion of the last:

$$\text{makespan} = max(\hat{f}_i) - min(\hat{a}_i) \tag{1}$$

**Definition 12.** The **waiting time** of a job is the time that elapsed between the submission of the job and the beginning of its execution in the infrastructure.

$$\text{waiting time}(\hat{j}_i) = \hat{f}_i - d_i - \hat{a}_i \tag{2}$$

Makespan, mean waiting time and max waiting time are given in Table 1. Other usual scheduling metrics like turnaround time or slowdown can be found in the notebooks.

These results confirm what was said in introduction: the traditional replay model is not satisfactory to simulate a change in the infrastructure or in the scheduler. In the recorded log, the waiting times were of 0.18 days on average, and 11.34 days maximum for KTH (resp. 0.26 and 62.48 for SDSC). With the feedback model and *whichever the change we make in the infrastructure*, the mean waiting times are under 0.62 day for KTH (**perf/2 a60**) and 1.61 for SDSC (**perf/2 a60**). All the max waiting times remain lower than the original max waiting times. In the case of rigid replay however, the picture looks different. We get waiting times of up to 141 days (**perf/2 rigid KTH**), resp. 508 days (**perf/2 rigid SDSC**). The mean waiting times are also significantly higher than the original in **infra/2** and **perf/2** experiments (up to two months for **perf/2 rigid SDSC**). It is unrealistic to think that the users would have waited on average all this time if the change actually occurred in the real infrastructure. Instead, they would have slowed down their pace of submission, which our model successfully accounts for.

[7] Experiment repository: gitlab.irit.fr/sepia-pub/open-science/expe-replay-feedback. The outputs are directly visible in the GitLab interface.

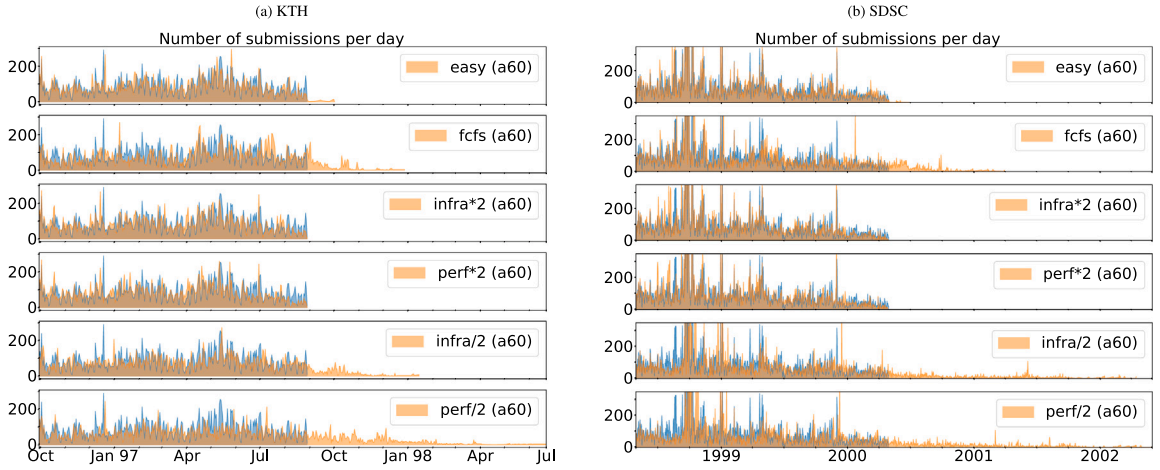



**Fig. 2.** Distribution of submission times with rigid (blue) and feedback (orange) replay models. Here we only plot feedback a60, but the picture with a0 is sensibly the same.

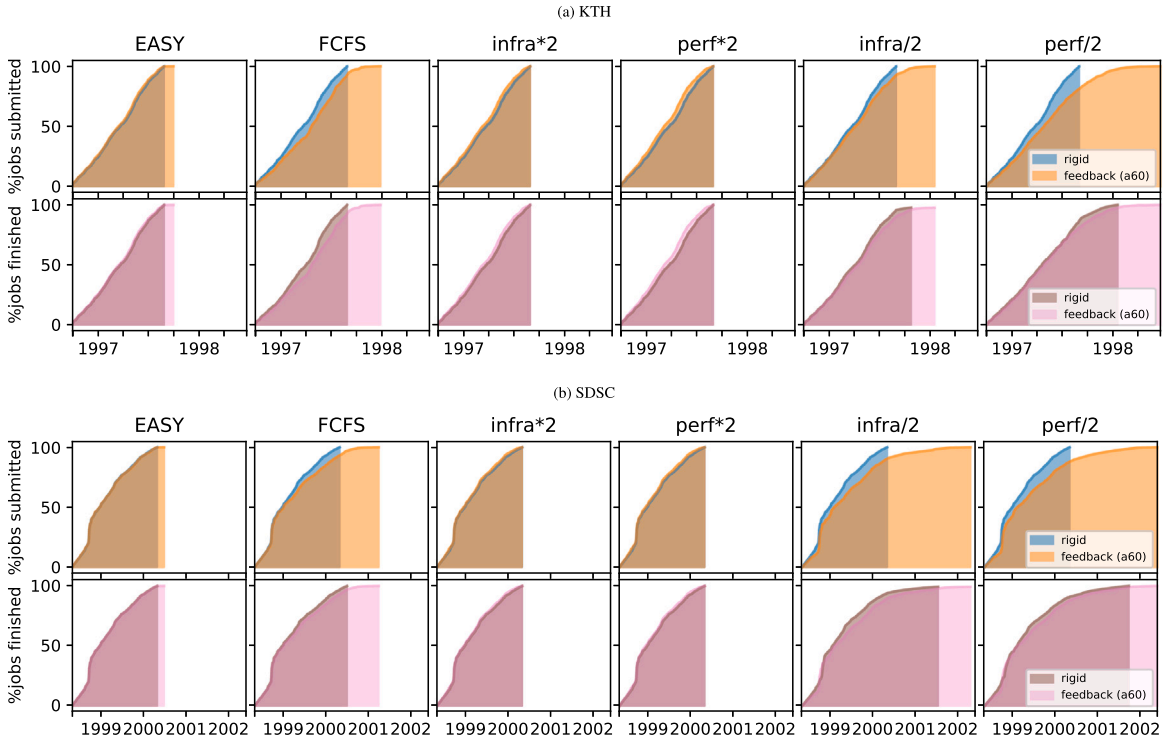



**Fig. 3.** Cumulative number of jobs submitted (top) and finished (bottom).

Since the pace of submission slows down in reaction to a slower infrastructure (**FCFS**, **perf/2** and **infra/2**) with the feedback model, it should take more time for the same workload to be fully executed. This effect is clearly visible in the results: the makespan in experiments **FCFS**, **perf/2** and **infra/2** increases significantly more with feedback replay than with rigid replay, compared to the original makespan. However, we would also expect to see the opposite effect when the infrastructure is faster (**perf** $*$ **2** and **infra** $*$ **2**), which is not the case here. The makespans in these experiments with rigid, a0 and a60 replay models are very similar, close to the original makespan. This is due to the relative rigidity remaining in the feedback model, which we discuss in Section 7.3.3 to give hints for improvement.

Note that the results also show that the scheduler in the real infrastructure and our implementation of EASY backfilling are not exactly the same. For example, the mean waiting times are significantly lower with our implementation (experiment **EASY rigid**). All the same, EASY seems closer to the original scheduler than FCFS, with which the mean waiting time explode (**FCFS rigid**). Interestingly, FCFS produces a *max waiting time* close to the original in both logs, suggesting that some jobs are probably submitted in pure FCFS order in the original scheduler. A detailed description of the scheduler originally used would be necessary to understand better, which we could not find for these logs.

In the end, the usual scheduling metrics discussed in this section give us useful insights, but they show their limits to fully explain the effect of changing the replay model. For example, they do not capture to what extent the submission times are shifted compared to rigid replay. To that end, we introduce in the following section a new way of making the analysis, including new metrics.

## 6. New metrics for analysis

This section starts by analyzing the distribution of submission times, before introducing our three new metrics *mean lateness*, *relative lateness* and *additional lateness*.

### 6.1. Submission time distribution

The replay with feedback model primarily impacts the *submission times* of jobs. Its effect is thus visible on the temporal distribution of job arrivals, plotted in Fig. 2 and in cumulative values in Fig. 3. The blue curve, corresponding to the rigid replay model, remains identical for all experiments: it corresponds to the original timestamps of submission in the recorded logs. With the feedback model, however, we observe that the submission distribution spreads with the specific infrastructure or scheduler used.

In experiments **perf/2**, **infra/2** and **FCFS**, we get confirmation that the simulated users submitted fewer jobs per day on average (orange curve under the blue curve in Fig. 3). In return, the length of the submission period has increased (horizontal span of the orange curve longer than the blue). On Fig. 2 we observe that, passed the original length of submission, the rate of submission decreases in trend. This is also visible in the cumulative graphs: the orange curve starts to slowly plateau where the blue graph ends. These are in fact end-of-simulation side effects: users finish submitting their backlog of jobs, without new jobs and users arriving. These effects are not relevant, and the analysis should instead focus on what happens in the simulations within the length of the original workload.

Experiments **perf** $*$ **2** and **infra** $*$ **2** need a closer look, as the effect of the infrastructure change is less visible in this case. We can observe that the orange curve is slightly above the blue in the cumulative graphs, meaning that the rate of submission increased slightly. In return, the length of submission is not shorter, but we can notice that users submit fewer jobs per day at the very end, for example in the last month of KTH log. They are reaching the end of their pool of jobs to submit.

Regarding the schedulers, the graphs confirm that our implementation of **EASY** is closer to the scheduler used in the real infrastructures than **FCFS** is. Indeed, the rate of submission with rigid and feedback looks fairly similar in experiment **EASY**. This is an indication that the jobs get executed and finished around the same time (we remind that rigid replay preserves the *original timestamps* while feedback replay preserves the *think times*). With the scheduler FCFS, the patterns of submission in Fig. 2 look more disrupted. This is due to the absence of backfilling: big jobs are blocking the queue, delaying the execution of small jobs, and the users have to wait for their termination before submitting the next jobs that *depend on* them.

Finally, Fig. 3 also displays in its bottom graphs information about rate of job *terminations*. This time, the distributions with rigid replay (in brown) vary between the experiments, because contrary to the submission times, the finish times of jobs do get affected by the infrastructure or scheduler used. The makespans given in Table 1 are reflected in the horizontal spans of these plots. We get to see that if the makespans in experiment **EASY** were larger with feedback compared to rigid, it is only due to a few jobs that got delayed. Indeed, the right part of the pink curve after the brown curve ends is essentially flat. Also, we note that the cumulative distributions of job terminations look fairly similar for all experiments, if we disregard the side effects in the end. This is an indication that throughput (defined as average number of jobs finished per week) is relatively independent of the replay model. We come back to that in the Discussion (Section 7.3.3)

If we were able to better characterize the effect of the replay model thanks to the distributions of submission times, we lack reliable metrics to measure it quantitatively. We attempt to fill this gap in the next section, by defining three new metrics.

### 6.2. Mean lateness, relative lateness and additional lateness

#### 6.2.1. Mean lateness

First, we define the *lateness* of a job, a fundamental quantity that will allow us to define the three metrics:

**Table 2**
New metrics calculated for all the experiments. Units: mean lateness in days, additional lateness in seconds and relative lateness without unit. Please note that lateness in a quantity that tends to accumulate for long chain of jobs. Taking the mean hides this distribution.

| Expe | Replay | *KTH* | | | *SDSC* | | |
|---|---|---|---|---|---|---|---|
| | | Mean lateness | Relative lateness | Additional lateness | Mean lateness | Relative lateness | Additional lateness |
| EASY | a0 | −3.36 | 0.99 | −20.39 | 2.35 | 1.00 | 6.00 |
| | a60 | −4.47 | 0.99 | −27.12 | 1.04 | 1.00 | 2.65 |
| FCFS | a0 | 32.66 | 1.10 | 198.18 | 76.90 | 1.11 | 196.38 |
| | a60 | 26.31 | 1.08 | 159.64 | 36.00 | 1.05 | 91.92 |
| perf $*$ 2 | a0 | −12.40 | 0.96 | −75.27 | −11.04 | 0.98 | −28.18 |
| | a60 | −13.31 | 0.96 | −80.79 | −11.55 | 0.98 | −29.49 |
| perf/2 | a0 | 46.10 | 1.14 | 279.75 | 106.58 | 1.15 | 272.17 |
| | a60 | 43.54 | 1.13 | 264.24 | 95.34 | 1.13 | 243.47 |
| infra $*$ 2 | a0 | −8.65 | 0.97 | −52.48 | −8.58 | 0.99 | −21.92 |
| | a60 | −9.32 | 0.97 | −56.57 | −9.23 | 0.99 | −23.56 |
| infra/2 | a0 | 16.48 | 1.05 | 99.99 | 89.82 | 1.12 | 229.37 |
| | a60 | 14.91 | 1.04 | 90.48 | 81.50 | 1.11 | 208.12 |

**Definition 13.** The **lateness** $\ell(i)$ of job $j_i$ is the difference between its submission time in the replay and in the original record: $\ell(i) = \hat{a}_i - a_i$.

Consequently, for a set of jobs $(j_0, \dots, j_{n-1})$, we can compute our first metric, the **mean lateness**, denoted $\bar{\ell}$:

$$\bar{\ell} = \frac{1}{n}\sum_{i=0}^{n-1} \ell(i) = \frac{1}{n}\sum_{i=0}^{n-1} (\hat{a}_i - a_i) \tag{3}$$

*Mean lateness* can be calculated per user or on the whole simulation. It measures *how many days difference there are on average between the original submission times and those in the simulation*.

Calculated on all the jobs in the simulation, *mean lateness* characterizes "the extent to which the orange curve is shifted to the right" in Fig. 2. Values of *mean lateness* for each experiment are given in Table 2. We can read for example that jobs in experiment **perf/2 a60** are submitted 44 days later on average with KTH log, and 13 days *earlier* in experiment **perf** $*$ **2 a60**. We also notice that values are positive for experiments **FCFS**, **perf/2** and **infra/2**, indicating that jobs are submitted *later* on average, and negative for experiments **perf** $*$ **2** and **infra** $*$ **2**, a sign that jobs are submitted *earlier*. *Mean lateness* for experiment **EASY** are close to zero. This confirms our previous observations.

*Mean lateness* per user is plotted in Fig. 4. First, we see that values are very scattered. Depending on the user, *mean lateness* can be several orders of magnitude different. We can nevertheless make the same distinction between the experiments that have a positive lateness, and experiments with a negative lateness. More interestingly, we observe a drift to high values as the number of jobs submitted by the user gets higher. In other words, users that submit more jobs tend to have a greater (positive or negative) *mean lateness*. This makes sense as the more the users submit jobs, the more they get to experience the feedback given by the infrastructure, hence the more they accumulate lateness. An implication of this drift is that *mean lateness does not scale with the size of the workload*. This makes the metric *mean lateness* unpractical to compare different workloads. That is why we introduce the two next metrics built to be independent on the number of jobs: *relative lateness* and *additional lateness*.

#### 6.2.2. Relative lateness

The metric **relative lateness** is the expression of mean lateness relatively to the length of the original workload. We want to see how significant the shifts in submission times in the replay are. Consequently, we define the length of the workload as the inter-arrival time between the first and the last job. It gives for relative lateness a dimensionless quantity:

$$\text{relative lateness} = 1 + \frac{\bar{\ell}}{a_{n-1} - a_0} \tag{4}$$

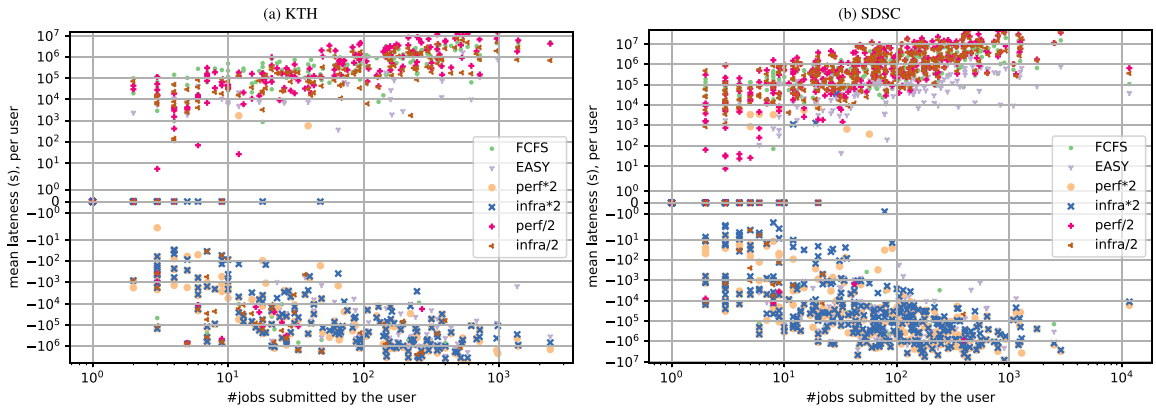



**Fig. 4.** Mean lateness per user, for all experiments, with replay a60 (logarithmic scale). Each dot corresponds to one simulated user. A positive (resp. negative) value indicates that the user submitted later (resp. earlier) on average in the replay with feedback compared to the recorded log.

Values for this metric are given in Table 2. A *relative lateness* $> 1$ corresponds to a *mean lateness* $> 0$, so a simulation where the submission times spread out over time. The maximum *relative lateness* in our experiments is reached by **perf/2 a0 SDSC**, with a value of 1.15. A way to interpret it is: "dividing the performances of the nodes by two lead the users to accumulate a delay in their submissions, corresponding to 15% of the length of the workload".

#### 6.2.3. Additional lateness

Another way to make the metric independent on the number of jobs is to look at the "additional lateness" $\delta_i$ that accumulates with each new replay job $\hat{j}_i$:

$$\ell(i) = \ell(i-1) + \delta_i \tag{5}$$

Once again, the $\delta_i$ can be defined per user (the successive $j_i$ would be the successive jobs submitted by one user) or on the whole simulation (the $j_i$ would be all the jobs of the simulation, ordered by original submission time). $\delta_i$ is a duration, that can take positive or negative values. Similarly to common scheduling metrics such as the waiting times, they fluctuate a lot with $i$. To understand the overall trend, one should look at their distribution. However, taking the mean is not meaningful as the $\ell(i)$ would cancel out when we take the sum in Eq. (5): $\sum \delta_i = \ell(n-1) - \ell(0)$.

Instead, to build a simple yet aggregated metric, we suppose that $\delta_i$ is constant equal to $\delta$. Since $\ell(0) = 0$ with our replay model, we have by recurrence on $i$: $\ell(i) = i\delta$.

Injecting this in the definition of mean lateness gives:

$$\bar{\ell} = \frac{1}{n}\sum_{i=0}^{n-1} \ell(i) = \frac{\delta}{n}\sum_{i=0}^{n-1} i = \frac{\delta}{n}\frac{(n-1)n}{2} = \frac{\delta(n-1)}{2}$$

Thus, we propose the metric **additional lateness**, denoted $\delta$, defined through the formula below:

$$\delta = \frac{2\bar{\ell}}{n-1} = \frac{2}{n(n-1)}\sum_{i=0}^{n-1}(\hat{a}_i - a_i) \tag{6}$$

We interpret this metric as the *additional delay that the users accumulate at each submission, in response to the feedback provided by the infrastructure*. Values of *additional lateness* on our simulations are given in Table 2. For example, halving the node performances makes the users accumulate 280 s of extra delay at each job submitted with KTH log, and 272 s with SDSC (replay a0). On the contrary, doubling the performances makes the users submit an extra 75 s earlier on average at each job for KTH, and 28 s for SDSC.

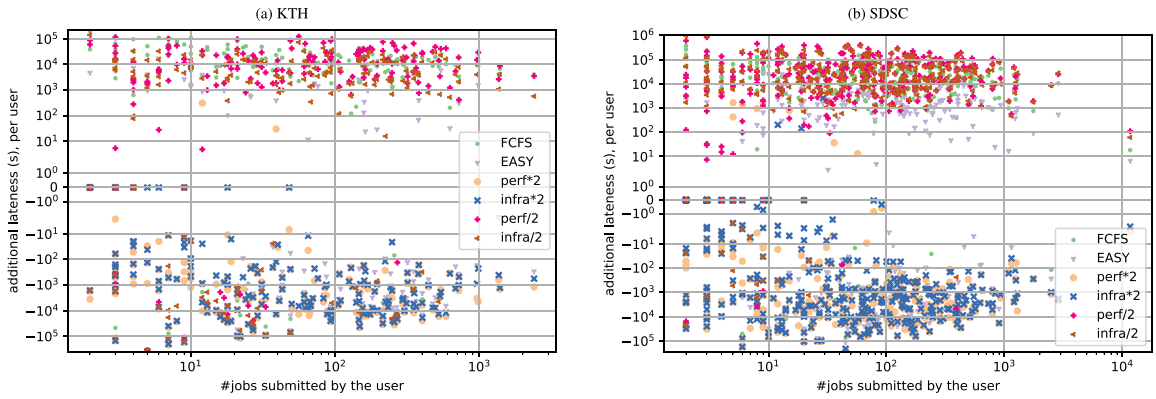
#### 6.2.4. Analysis of relative lateness and additional lateness results

**Preliminary remark:** Looking at the definitions of the two metrics in Eqs. (4) and (6), we note that $relative lateness - 1$ and $\delta$ are roughly proportional to $\bar{\ell}/n$, assuming that there is an affine relationship between the length of the simulation and the number of jobs. This implies that *relative lateness* and *additional lateness* are linearly correlated, which we were able to confirm experimentally with our data. Consequently, the analyses that can be made for one metric also apply to the other, and we will only present in the following the analyses for the metric *additional lateness*.

**Which parameter influences the *additional lateness*?** In our results (Table 2), the parameter influencing the most the *additional lateness* is the infrastructure/scheduler. For a fixed log and replay method, we get very different values of *additional lateness* depending on the performances or number of nodes or the type of scheduler. For instance in log **KTH** and replay method **a60**, *additional lateness* ranges from −80.79 to 264.24 days. In second comes the specific log used for the replay. In our case, except for experiments **perf/2 a0**, **perf/2 a60** and **FCFS a0** where they are relatively similar, we observe a significant variability in *additional lateness* between the logs. The overall trends remain the same in both logs. Finally, the change of replay method (a0 or a60) has the lowest influence on *additional lateness* in our results. A notable exception is experiment **SDSC FCFS**, where using a60 instead of a0 makes the *additional lateness* decrease significantly. A possible explanation is the presence of several flurries of very high activity by individual users in this log,[8] that get grouped in the same few sessions with a60, so submitted concomitantly. With a0, every job is a separate session that waits for its dependencies to finish, which might lead to the increased delay with FCFS.

***Additional lateness* per user.** *Additional lateness* per user are plotted in Fig. 5. Unlike for *mean lateness*, the values are independent on the number of jobs submitted by the user: there is no drift compared to Fig. 4. However, they still depend on the specific user, with great variability. In fact, there are differences in *additional lateness* of more than 10'000 s between the 10th and 90th percentiles in all the experiments (Fig. 5(c)). We also note that the median *additional lateness* per user are in the order of *hours* while they are in the order of *minutes* when aggregated at the level of the whole simulation (Table 2). This means that despite the overall *additional lateness* being relatively low, the additional lateness experienced by most users is much more significant.

[8] Up to 11 740 jobs submitted by the same user in less than 25 days, see "Usage Notes" in the page describing the log.




(c) Percentiles on the distributions of additional lateness plotted above (in seconds)

| | *KTH* | | | *SDSC* | | |
|---|---|---|---|---|---|---|
| | $10^{th}$ | $50^{th}$ | $90^{th}$ | $10^{th}$ | $50^{th}$ | $90^{th}$ |
| FCFS | -5 | 8775 | 44525 | 0 | 12141 | 67078 |
| EASY | -17980 | -699 | 918 | -8898 | 0 | 4180 |
| perf*2 | -26247 | -2186 | -49 | -26170 | -1803 | 0 |
| infra*2 | -25321 | -1553 | -18 | -21738 | -1288 | 0 |
| perf/2 | 0 | 10485 | 47168 | 0 | 15190 | 153976 |
| infra/2 | -8695 | 2586 | 19434 | 0 | 13424 | 119480 |

**Fig. 5.** Additional lateness per user, for all experiments, with replay a60.

## 7. Discussion

In this section, we start by discussing the results. We see how our approach can enlighten us on the influence of a change in the infrastructure 7.1 and how the session delimitation method impacts the results 7.2. Then, we come back on our feedback model and highlight some of its limits 7.3, as well as the limits of our experimental campaign 7.4. We also point out the differences of our model with related approaches 7.5. Finally, we focus on the generalization of the new metrics by studying their scalability with regard to workload size 7.6.

### 7.1. Influence of the change in infrastructure

Thanks to the replay model and new metrics, we are able to characterize the effect that a change in the infrastructure might have on user submission behavior. It will impact the submission times, shifting them forward or backward. Below is a ranking of the impact of the studied infrastructure change, from the earliest to the latest submission times in relation to the original times, based on Table 2:

1. **perf** $*$ **2** (earlier than original)
2. **infra** $*$ **2** (earlier)
3. no change (**EASY**)
4. **infra/2** (later)
5. **perf/2** (later)

This ranking is verified by both logs, no matter the replay method (a0 or a60). For KTH log, the change to scheduler **FCFS** would rank between items 4 and 5 while for SDSC it would come between 3 and 4.

Importantly, we see that doubling/halving the node performance has a more significant effect in shifting the submission times than doubling/halving the *number* of nodes (in absolute value). This effect is particularly visible with KTH log. In fact, **changing the performance directly affects the execution time of every job**. Changing the number of nodes, however, has no effect on the execution times, but only **impacts indirectly the waiting times**. If the original infrastructure was already oversized, this change will have little effect. Note also that decreasing the number of nodes below the maximum number of requested resources will cause some jobs to be rejected.

### 7.2. Influence of the delimitation method

At the root of the feedback model is the partitioning of jobs into sessions (see Definition 4). In the experiments presented in this paper, we used two methods: a delimitation on inter-arrival of 0 min and 60 min. The characteristics of the resulting session graphs are shown in Fig. 6.

**Session graph structure.** We observe a large diversity in the size of the session graphs for the different users: some contain only one session, while others have thousands of sessions with the longest path inside the graph of several hundred sessions. These reflect the intrinsic differences between HPC users that use the platform for various motives and with different level of activity. Unsurprisingly, the use of delimitation a60 reduces significantly the number of sessions in the graphs, hence the longest paths. More notably, we observe that a60 reduces greatly the arity of the graphs: there is no graph with an arity greater than 11 with this delimitation method. In other words, there are less "sessions in parallel" with a60. Also, an analysis on the think times reveals that 75% of edges have a think time <10 h and 35% are <1 h for delimitation a0, while 50% of think times are <10 h and around 13% are <1 h with a60. In short, **the session graphs produced by the two delimitations methods have very different structure.**

**In the experiments.** All the same, and as we already mentioned in the previous sections, the two delimitations studied have little influence on the results. Scheduling metrics and our new metrics are roughly the same (Tables 1 and 2) and submission time distributions look very similar. However, if we look more carefully, we note that *mean lateness* (and hence *relative lateness* and *additional lateness*) are always lower with a60. To see that more in detail, we plotted the distribution of the difference in submission times between the different methods in Fig. 7.

As we can see, the difference between the submission time in a0 and a60 is positive for almost all jobs. **Delimitation a0 lead to slightly (a few days) later submission times than a60**. This effect is explained by the greater complexity of session graphs obtained with a0 that we explained above. Having more sessions and more dependencies results in less flexibility during the replay. If one job gets delayed in its execution, it will have more impact because it has more successors.

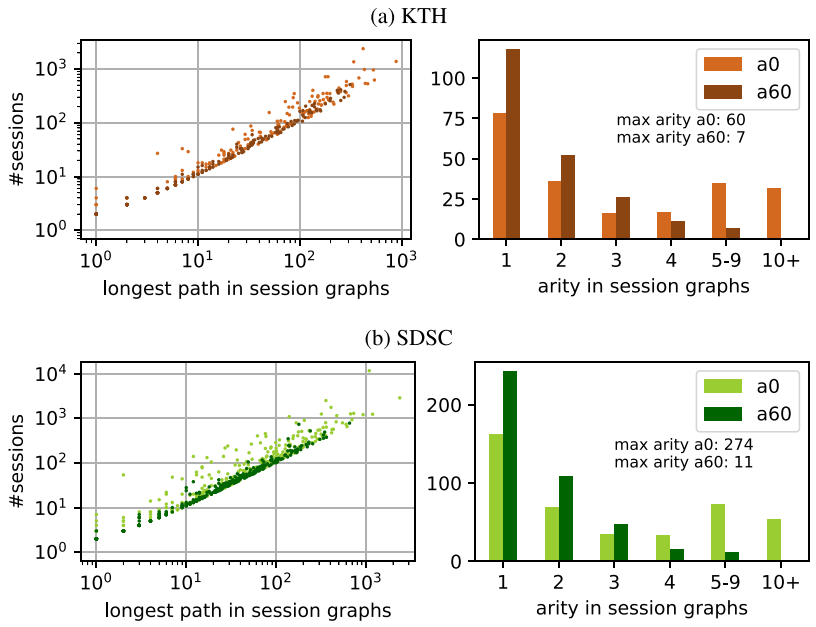



**Fig. 6.** Longest path and arity distribution of user session graphs.

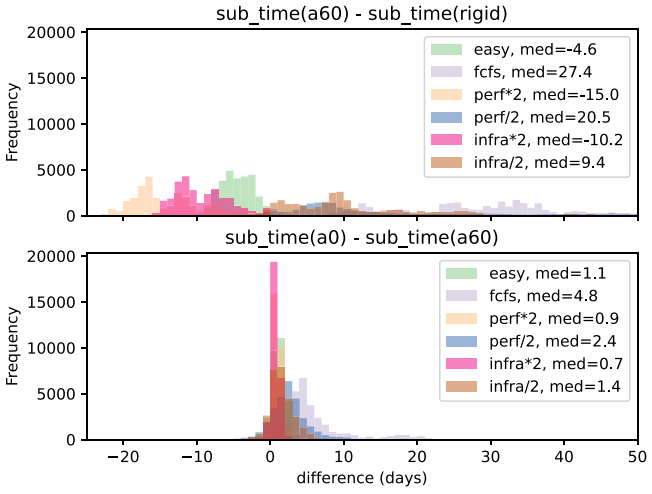



**Fig. 7.** Distribution of the difference in submission timestamps between rigid and a60 replay methods (top) and a60 and a0 (bottom), KTH log. Note: the top graph corresponds to the definition of lateness, and it confirms the ranking made in 7.1.

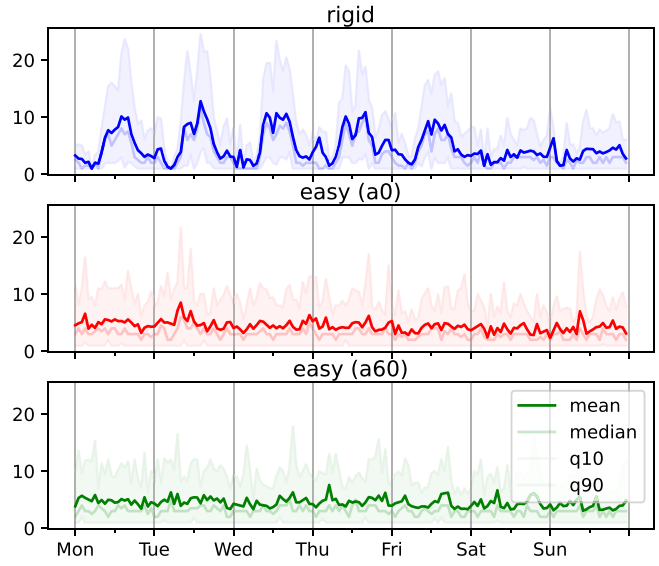



**Fig. 8.** Number of submissions per hour, aggregated by week, KTH log.

### 7.3. Limits of our feedback model

The model of replay with feedback used in this article allows accounting for effects that are invisible in traditional simulations. However, we point out in this section several limitations that would need to be addressed to reach further realism.

#### 7.3.1. Only one type of user response

First, let us recall that our method of replay with feedback focuses only on submission times. In reality, user response to feedback goes well beyond (see Section 4.1). However, we chose to stick to it for the three following reasons:

1. Even for feedback on submission time, we cruelly lack related literature and methods of validation (see 7.3.4). We found no literature on the other types of user response.
2. Multiplying the parameters that we change compared to rigid replay makes it harder to deeply analyze the effect of the proposed feature. We preferred to proceed by incremental steps.
3. Allowing for more types of user response might alter the input workload even further, to the point where it is not easy to know if it kept its fundamental structure.

#### 7.3.2. Day/night variability

In real infrastructures, we observe a day/night and weekday/weekend variability in the user submissions. This variability is also present in our input data (see top graph Fig. 8). Unfortunately, with the replay method used in this article, the submission times get shifted around, and this variability is lost. This could be fixed for example by adding assumptions on activity times for users (like in the 'distribution-based' user model [4]).

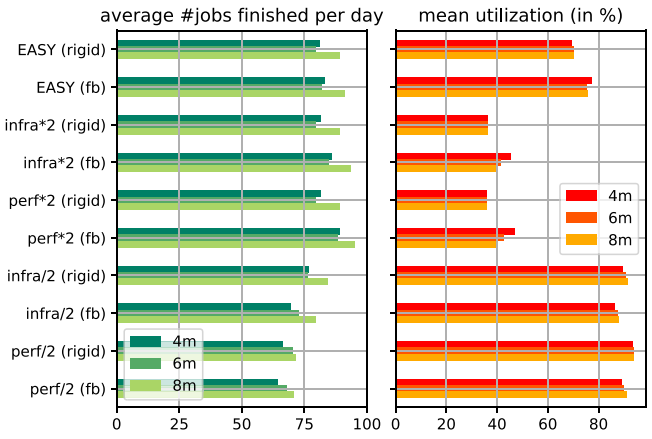



**Fig. 9.** Throughput (average number of job terminations per day) and mean utilization (average number of computing nodes), KTH log. The metrics are calculated on a time window starting two weeks after the beginning of the simulation and with length 4, 6 and 8 months, to leave away beginning- and end-of-simulation edge effects.

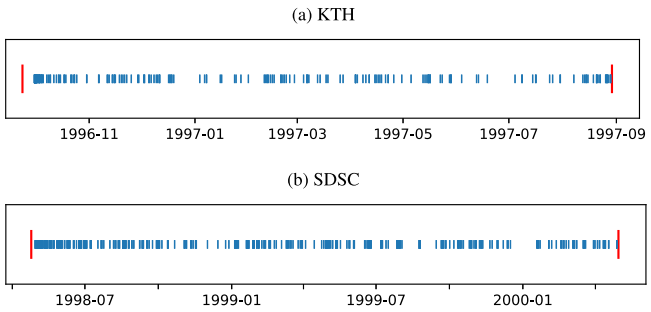



**Fig. 10.** User arrivals in the platform. Each vertical blue bar represents the first time one user submits in the platform. The red vertical bars are the start and end time of the original log.

#### 7.3.3. Remaining rigidity in the feedback model

Our method, although better than rigid replay in this regard, is unsatisfying to fully capture user response to feedback from the infrastructure. In fact, as already mentioned, doubling the performances of nodes only stretches the length of the submission period by 0.99 or 0.98 (see *relative lateness* in Table 2). A more significant rebound effect would have been expected as a consequence of such a performance gain. An analysis of throughput and utilization in the different experiments, plotted below in Fig. 9, enlightens us on the reasons behind this limited rebound.

Doubling the performances or number of nodes has no effect on throughput with the rigid replay model. The throughput in this case of oversized infrastructure (mean utilization <40%) is dictated by the fixed job arrivals. With the feedback model, the infrastructure change does lead to a higher throughput, but only 5 to 11% greater compared to rigid. The rebound in mean utilization is only from 36%–37% (**rigid infra** $*$ **2** and **perf** $*$ **2**) to 40%–47% (**feedback infra** $*$ **2** and **perf** $*$ **2**), far from approaching its original level of above 70%. Similar effects can be observed with the experiments **perf/2** and **infra/2**, with, this time, a saturation of the platform (utilization >90%).

The limited ability of our model to fully respond to feedback is due to its remaining "rigidity", coming from at least two factors. On the one hand, the think times are constant in the model. Even with the best performances from the infrastructure, they can never be reduced (for illustration, see Fig. 1: better performance can reduce the turnaround time of jobs, i.e. the length of the session boxes, but the brown arrow will keep the same length). On the other hand, the first job submitted by a user is always replayed at its original timestamp. In both KTH and SDSC logs, new users arrive in the platform up to a few days before the end of the record (Fig. 10). This explains why the length of the submission period is never significantly reduced when the performances are better. Finally, feedback only affects the submission times in our model. In reality, the performances of the system influence a wider variety of parameters: number and size of submitted jobs, number of user arrival or departure, etc., which our model do not account for.

#### 7.3.4. On the validation of the model

The model is perfectible, and this paper to not pretend to give a scientific validation of it. In fact, validating such a model is a challenging task, as already mentioned by Zakay and Feitelson [4], and we are not aware of any work attempting to do so.

We see at least two ways such a validation could be carried out:

- making a survey with users of grid/HPC infrastructures to understand what their behavior is in reaction to feedback (see Wolter et al. for an example of HPC user survey [14]),
- collecting data on a real infrastructure that underwent a major change and check if the model is able to predict the way the users adapted to this change (see Klusáček et al. for the analysis of Metacentrum log that underwent a major reconfiguration [17]).

We leave these avenues of research for future work.

### 7.4. Limits of the experimental campaign

**Generalizability.** The experimental campaign proposed in this paper only includes two workloads, which are both quite old (recorded before the year 2000). They were carefully selected because they disclosed information on their scheduling policy and featured simple platforms (homogeneous with monocore machines). Similarly, we studied only two monocore schedulers (EASY and FCFS). These were chosen as they are the most commonly used in the literature and correspond to the workloads. Since our work focus on the model of *replay* and not specific scheduling results, we argue that our campaign is sufficient to reach our conclusions, which would extend other workloads and other schedulers. Furthermore, we remind that we took particular attention to make the experiments reproducible. Hence, it should be easy to rerun the campaign with any workload available in the Parallel Workload Archive.

### 7.5. Comparison with related works

In this part, we come back to the differences between our model and Zakay and Feitelson's [4]. Like them, we do a replay with feedback on submission times only. The method is based on think times, and on a session partitioning of the original workload. Compared to them, we only preserve the think times between *sessions*. Zakay and Feitelson introduce an additional notion of "batch", which are groups of overlapping jobs within a session. From there, they propose three methods:

1. 'adjusted': preserves the think time between *batches*
2. 'distribution-based': when a batch becomes free, submit it if the current time is in a *period of activity* of the user (working day, working hours). Otherwise, shift it to the next period of activity. This method requires assumptions on periods of activity, that they manage through a probabilistic model.
3. 'fluid': preserve the session start and end times for users, they will be the "periods of activity". The batches are submitted only during these periods, if they are free.

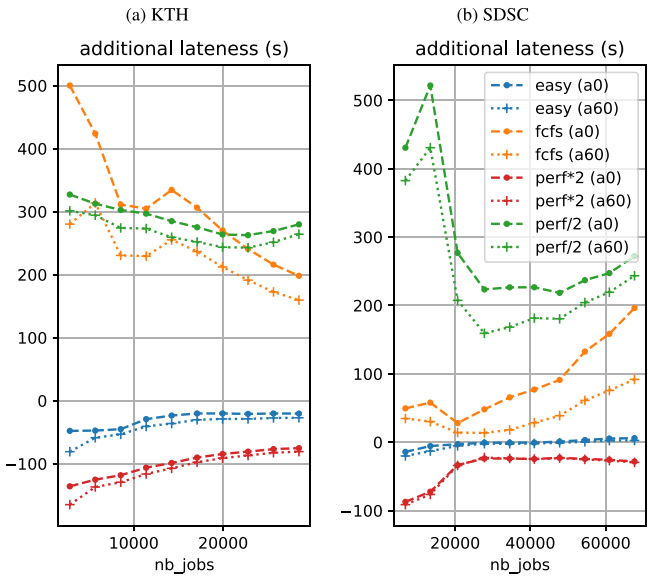



**Fig. 11.** Scalability of the *additional lateness* metric. For each point, a new simulation has been run with a subset of the workload as input: the subset contains only the $n$ first jobs (ordered by submission time) of the original workload. Then, *additional lateness* is calculated on the output using Eq. (6).

In this paper, we did not reproduce their methods to compare our results to theirs. The reasons are twofold:

Firstly, in absence of a validation method, such a comparison would be inconclusive. For example, we expect their 'adjusted' method to show similar results than ours but we would have no way to conclude which one is the most realistic. Similarly, 'distribution-based' artificially restores the seasonality (see 7.3.2) to the cost of an additional set of assumptions on periods of activity for users, making it difficult to know if it kept the fundamental features of the original log.

Secondly, we disagree with the assumptions behind the 'fluid' model. We think that the *sessions* that are deducted from the recorded workload and the *periods of activity* are two separate notions. If a user does not submit any job one day, it does not necessarily mean that she was not working that day, but rather that she did not have anything to submit. If the performances of the platform were different, she might have had something ready to submit that day.

Instead of proposing a comparison based on hypotheses and beliefs, we preferred to implement the simplest model of replay with feedback, and provide a solid theoretical and software base for future contributions in the domain.

### 7.6. Scalability of relative lateness and additional lateness

In Section 6, we introduced new metrics for analysis. The metrics *relative lateness* and *additional lateness* depend on the simulated platform, the scheduling algorithm, the workload and the user sessions delimitation method. Ideally, and unlike the metric mean lateness (remember the drift in Fig. 4), **we would like them to be independent on the length of the workload**. Fig. 5 is quite convincing in that regard as it does not show the drift mentioned above. To be sure, we tested in Fig. 11 that *additional lateness* remains the same if we run the simulation on any (sufficiently large) subset of the original workload.

The results are mixed. When only the infrastructure is modified, *additional lateness* seems to stabilize with the number of jobs as input. For experiments **easy** (no change in infrastructure) and **perf** $*$ **2** for example, *additional lateness* increases at first with the number of jobs, but seems to plateau after 20 000 jobs with both logs. The case of experiment **perf/2** is more problematic as *additional lateness* starts by decreasing until 20 000 jobs but increases again thereafter, especially with SDSC log. This means that the delay caused by halving the performances does not only add up with time ($\ell(i)$ increases), but the additional delay for each new submission also increases ($\delta_i = \ell(i) - \ell(i-1)$ increases). However, *additional lateness* is not scalable when the scheduler is modified (experiment **fcfs**). In this case, the metric does not seem to stabilize and behaves in the opposite way in both logs (decreasing for KTH and increasing for SDSC). It is hard to say what is intrinsic to the metric and what is due to heterogeneity in the input workloads.

To conclude, in our experiments, *additional lateness* scales rather well with the size of the workload for a change in infrastructure, but not for a change in scheduler. We recommend anyone using this metric to do this simple sensitivity analysis.

## 8. Conclusion

In this paper, we challenge the traditional way to simulate distributed systems by introducing a feedback loop in the workload model. Compared to using a pre-determined workload (historical record or generated) in the simulation, we let the workload adapt to the simulated performance of the system, in the same way that real users would adapt their pace of submission to the response they get from the infrastructure. This novel way of doing simulation, that we call "replay with feedback", was first proposed by Zakay and Feitelson [4]. It consists in using a historical workload as input and partitioning it into "sessions of work" for each user. During the simulation, we no longer preserve the original timestamps of submission, but rather the *think time between sessions*, i.e. the time that elapsed between the termination of all jobs in a session and the submission of the next one. Zakay and Feitelson introduce a notion of batch within session that are the sets of jobs whose execution overlap.

We complement their approach by providing a slightly different replay model, leaving aside the notion of batch to keep the model as simple as possible. Our model is implemented with a state-of-the-art simulator of distributed systems. The software developed are open source and customizable, to be easily reusable for implementing and studying other replay models. We apply our model and implementation by running a reproducible experimental campaign with two historical logs, with which we obtain similar results. The experiments show how replay with feedback can be used to predict the impact of a change in the infrastructure (computing performances, number of nodes, scheduling algorithm) on user submission behavior and scheduling performance. In our case, decreasing the performances or number of nodes make the users accumulate a delay at each submission compared to the baseline. On the other hand, increasing them instead made the users submit earlier on average. A change in performances has a more significant effect than a change in number of nodes. Lastly, we introduce three novel metrics, independent of the specific replay model, to describe the effect of feedback on user submission. *Mean lateness* measures the average time difference between the original submission times and those in the simulation. *Relative lateness* gives an expression of this time difference, relative to the length of the simulation. Finally, *additional lateness* expresses the additional delay that the users accumulate at each submission, in response to the feedback provided by the infrastructure.

This work contributes to what is in our opinion a fundamental yet much under-researched topic within the field of distributed system simulation. It requires to rethink the way we do simulation and interpret the results. Performance of computer systems are not only about bandwidth or number of operations per second, but rather the utility that they bring to the humans using them. Not taking the human factor into account leads to large overestimates of potential gains, as it neglects the rebound effect inherent to efficiency.

## 9. Future works

We are aware that this work opens more doors than it provides answers to the question of how best to simulate users of distributed

systems. We hope that it will spark interest for future research in this area. Relevant directions for future work could be:

- proposing a scientific validation protocol for the feedback model (see ideas in Section 7.3.4) and making it a standard for performance evaluation using simulation;
- studying other replay models that would account for day/night variability of submission or arrival/departure of user, and comparing them to existing models;
- studying other user response to feedback, like change in requested resources;
- extending the user model to also capture other types of feedback, like carbon intensity of electricity (as in [21]);

For all these directions, the open source tools and metrics provided in this article can prove useful.

## CRediT authorship contribution statement

**Maël Madon:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing. **Georges Da Costa:** Conceptualization, Supervision, Validation, Writing – review & editing. **Jean-Marc Pierson:** Conceptualization, Funding acquisition, Supervision, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Links to all code/data repositories are in the manuscript.

## Acknowledgments

This article would not have been possible without previous and high quality work of a few people that we want to mention here. First and foremost, we express our gratitude to Dror Feitelson for inspiring this research and maintaining the Parallel Workload Archive from which we could download our input data. Thanks also to Lars Malinowsky for providing the KTH workload log and Victor Hazlewood for SDSC. We also want to thank Millian Poquet, main developer and maintainer of Batsim, for his willingness to help with his simulator and Nix.

## References

[1] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter, Versatile, scalable, and accurate simulation of distributed applications and platforms, J. Parallel Distrib. Comput. 74 (10) (2014) 2899, http://dx.doi.org/10.1016/j.jpdc.2014.06.008.

[2] R. Buyya, M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurr. Comput.: Pract. Exp. 14 (13–15) (2002) 1175–1220, http://dx.doi.org/10.1002/cpe.710.

[3] G. Da Costa, L. Grange, I. de Courchelle, Modeling, classifying and generating large-scale Google-like workload, Sustain. Comput.: Inform. Syst. 19 (2018) 305–314, http://dx.doi.org/10.1016/j.suscom.2017.12.004.

[4] N. Zakay, D.G. Feitelson, Preserving user behavior characteristics in trace-based simulation of parallel job scheduling, in: Proceedings of the 8th ACM International Systems and Storage Conference, ACM, Haifa Israel, 2015, p. 1, http://dx.doi.org/10.1145/2757667.2778191.

[5] M. Vasconcelos, D. Cordeiro, G. Da Costa, F. Dufossé, J.-M. Nicod, V. Rehn-Sonigo, Optimal sizing of a globally distributed low carbon cloud federation, in: 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, 2023, pp. 203–215, http://dx.doi.org/10.1109/CCGrid57682.2023.00028.

[6] P. Wiesner, D. Scheinert, T. Wittkopp, L. Thamsen, O. Kao, Cucumber: Renewable-aware admission control for delay-tolerant cloud and edge workloads, in: J. Cano, P. Trinder (Eds.), Euro-Par 2022: Parallel Processing, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 218–232, http://dx.doi.org/10.1007/978-3-031-12597-3_14.

[7] I.F. de Nardin, P. Stolf, S. Caux, Analyzing power decisions in data center powered by renewable sources, in: 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD, 2022, pp. 305–314, http://dx.doi.org/10.1109/SBAC-PAD55451.2022.00041.

[8] S. Schlagkamp, Influence of dynamic think times on parallel job scheduler performances in generative simulations, in: N. Desai, W. Cirne (Eds.), Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 123–137, http://dx.doi.org/10.1007/978-3-319-61756-5_7.

[9] B. Schroeder, A. Wierman, M. Harchol-Balter, Open versus closed: A cautionary tale, in: Symposium on Networked Systems Design and Implementation, Carnegie Mellon University, San Jose, CA, USA, 2006, URL https://www.usenix.org/legacy/event/nsdi06/tech/full_papers/schroeder/schroeder_html/.

[10] J. Panneerselvam, L. Liu, N. Antonopoulos, Y. Bo, Workload analysis for the scope of user demand prediction model evaluations in cloud environments, in: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014, pp. 883–889, http://dx.doi.org/10.1109/UCC.2014.144.

[11] P. Kar, Workload Prediction in Cloud Datacenters Based on User Behavior Modeling (Undergraduate Thesis), Birla Institute of Technology and Science, Pilani, 2016, URL http://pratyushkar.com/files/WPRED.pdf.

[12] T.V. Dinh, L.L.H. Andrew, P. Branch, Exploiting per user information for supercomputing workload prediction requires care, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE, Delft, 2013, pp. 2–9, http://dx.doi.org/10.1109/CCGrid.2013.68.

[13] G.P. Rodrigo, P.-O. Östberg, E. Elmroth, K. Antypas, R. Gerber, L. Ramakrishnan, Towards understanding HPC users and systems: A NERSC case study, J. Parallel Distrib. Comput. 111 (2018) 206–221, http://dx.doi.org/10.1016/j.jpdc.2017.09.002, URL https://linkinghub.elsevier.com/retrieve/pii/S0743731517302563.

[14] N. Wolter, M.O. McCracken, A. Snavely, L. Hochstein, T. Nakamura, V. Basili, What's working in HPC: Investigating HPC user behavior and productivity, 2006, p. 14.

[15] N. Zakay, D.G. Feitelson, On identifying user session boundaries in parallel workload logs, in: W. Cirne, N. Desai, E. Frachtenberg, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, pp. 216–234, http://dx.doi.org/10.1007/978-3-642-35867-8_12.

[16] D. Klusáček, Š. Tóth, G. Podolníková, Complex job scheduling simulations with alea 4, in: Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques, in: SIMUTOOLS'16, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 2016, pp. 124–129, http://dx.doi.org/10.5555/3021426.3021446.

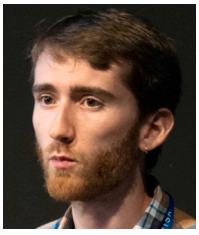
[17] D. Klusáček, Š. Tóth, G. Podolníková, Real-life experience with major reconfiguration of job scheduling system, in: N. Desai, W. Cirne (Eds.), Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 83–101, http://dx.doi.org/10.1007/978-3-319-61756-5_5.

[18] N. Zakay, D.G. Feitelson, Semi-open trace based simulation for reliable evaluation of job throughput and user productivity, in: Proceedings of the 9th ACM International on Systems and Storage Conference, ACM, Haifa Israel, 2016, p. 1, http://dx.doi.org/10.1145/2928275.2933280.

[19] P.-F. Dutot, M. Mercier, M. Poquet, O. Richard, Batsim: A realistic language-independent resources and jobs management systems simulator, in: 20th Workshop on Job Scheduling Strategies for Parallel Processing, Chicago, United States, 2016, http://dx.doi.org/10.1007/978-3-319-61756-5_10.

[20] D.A. Lifka, The ANL/IBM SP scheduling system, in: G. Goos, J. Hartmanis, J. Leeuwen, D.G. Feitelson, L. Rudolph (Eds.), in: Job Scheduling Strategies for Parallel Processing, vol. 949, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 295–303, http://dx.doi.org/10.1007/3-540-60153-8_35.

[21] M. Madon, G. Da Costa, J.-M. Pierson, Characterization of different user behaviors for demand response in data centers, in: J. Cano, P. Trinder (Eds.), Euro-Par 2022: Parallel Processing, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 53–68, http://dx.doi.org/10.1007/978-3-031-12597-3_4.



**Maël Madon** received in 2021 a double engineering degree from Ecole Polytechnique, France and KTH Royal Institute of Technology, Sweden. He is currently a Ph.D. student in Computer Science at IRIT, University of Toulouse, France. His personal concerns about the climate crisis pushed him to seek a strong "sustainability" coloration in his studies and research. His research interests include green IT, digital sufficiency, modeling and simulation of distributed systems and energy- and user-aware scheduling.

**Georges Da Costa** is Professor in Computer Science at the University of Toulouse. He received his Ph.D. from LIG (Grenoble, France) in 2005 and his Habilitation from University Paul Sabatier (Toulouse, France) in 2015. He is a member of the IRIT Laboratory. His research currently focus on energy aware distributed systems. His research highlights are HPC & cloud computing, large scale energy aware distributed systems, performance evaluation, ambient systems.

**Jean-Marc Pierson** serves as a Full Professor in Computer Science at the University of Toulouse (France) since 2006. He received his Ph.D. from the ENS-Lyon, France in 1996. He was an Associate Professor at the University Littoral Cote-d'Opale (1997–2001) in Calais, then at INSA-Lyon (2001–2006).

He is a member of the IRIT Laboratory and member of the SEPIA Team on distributed systems. His main interests are related to large-scale distributed systems, Cloud, HPC, IoT. He served on several PCs and editorial boards in the Cloud and Energy-aware computing area. His researches focus on energy aware distributed systems, in particular monitoring, job placement and scheduling, virtualization, networking, autonomic computing, mathematical modeling, renewable energies in datacenters, and the usage of machine learning in these environments. He was chairing the EU funded COST IC804 Action on "Energy Efficiency in Large Scale Distributed Systems" and participates in several national and european projects on energy efficiency in large scale distributed systems. He is currently chairing the French National ANR DATAZERO 2 Project.