Real-life Experience with Major Reconfiguration of Job Scheduling System

Dalibor Klusáček[1,2], Šimon Tóth[2], and Gabriela Podolníková[2]

[1] CESNET a.l.e., Zikova 4, Prague, Czech Republic
[2] Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
{xklusac,toth,xpodoln}@fi.muni.cz

Abstract. This work describes the goals and impacts of a large reconfiguration of the job scheduling system, used in the Czech National Grid and Cloud infrastructure MetaCentrum, which was implemented in early 2014. MetaCentrum, as a "long-tail" oriented provider, serves a varied user-base consisting of both individual users and research groups. This imposes strict requirements on the robustness of job scheduling algorithms being employed, as the system must be capable of assigning a highly heterogeneous set of workloads to a similarly heterogeneous set of computational resources. Primary goals for MetaCentrum were always to provide efficient and fair resource utilization with respect to different users in the system. During the last few years, MetaCentrum has gone through a period of rapid growth (1,500 CPU cores in 2009 vs. 10,600 CPU cores in 2014) forcing us to re-evaluate our scheduling approaches, as the "old" configuration no longer satisfied our utilization and fairness demands. This re-evaluation was supported by a significant body of research, which included the proposal of new scheduling approaches as well as detailed simulations based on real-life complex workload traces. First of all, a new multi-resource aware fair-sharing algorithm (based on our recent research) was deployed, with the goal of improving fairness with respect to the growing heterogeneity of resources and users' workloads. Second, the queue configuration of the entire system was completely reworked in order to decrease resource fragmentation and improve the utilization and the impact of fairness policies. This paper summarizes the effects of these changes using real-life data from the production system. Moreover, we publish complex workload traces from MetaCentrum that were used in this paper, since they represent a valuable source of data concerning a highly heterogeneous production system. Last but not least, we also present our advanced job scheduling simulator which is routinely used for testing of new scheduling strategies prior their deployment in the real system.

Keywords: Scheduling, Fairness, Queue, Workload, Heterogeneity

1 Introduction

MetaCentrum serves various users and research groups. During the last 5 years, MetaCentrum has grown from approximately 1,500 CPU cores (2009) to al-

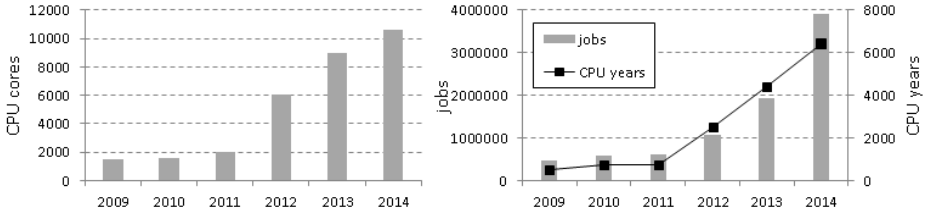

Fig. 1. Available CPU cores (left) and the number of jobs and used CPU years (right).

most 11,000 CPU cores (2014), with the number of processed jobs matching this growth curve (see Fig. 1). The system is divided into two separate pools of resources, each managed by a different job scheduler. The smaller pool ($\sim 4,900$ CPUs) is managed by a custom-developed scheduler which uses planning (instead of queues) [7] while the larger pool ($\sim 6,100$ CPUs) is managed by a queue-based scheduler based on TORQUE resource manager. While the plan-based scheduler has been heavily optimized in the past, the original "historic" scheduling approaches used in the queue-based scheduler — which remained mostly the same for a decade — were becoming clearly inefficient and had to be revised to better reflect growing heterogeneity of both hardware resources and users' workloads. In this work, we focus on the queue-based scheduler which manages the major part of MetaCentrum computing resources.

The goal of this work is to share our real-life experience with a major reconfiguration of a production system as it was a unique opportunity to apply "theoretical" results in practice. Therefore, we summarize our previous efforts and describe how the newly proposed modifications were evaluated and applied in practice, i.e., we provide new results showing the improvement of system performance achieved through a newly defined scheduling setup.

In case of MetaCentrum, there were two main issues with the historical setup: an obsolete (unfair) fair-sharing mechanism and a rather inefficient queue configuration. When solving these issues, we are building upon our earlier "theoretical" works where new multi-resource aware fair-sharing mechanisms were proposed [9] and the impact and interactions of various system-specific policies were described [10]. It is worth noticing that the improvement was solely achieved by the newly configured queues and new fair-sharing mechanism, while the actual scheduling algorithm remained unchanged. Furthermore, we also provide detailed information concerning MetaCentrum *infrastructure and users' workloads* that were used both for the development of the new system configuration as well as for later analysis of the suitability of the new solution [13]. They represent valuable source of data, especially in terms of heterogeneity of system resources and users' workloads. Last but not least, we have prepared a largely extended version of our jobs scheduling simulator *Alea* [2], which was heavily used when developing the new system setup and provides advanced simulation capabilities compared to its previous releases [8].

This paper is organized as follows. Applied modifications of the scheduling system are presented in the following section. Section 2.1 describes the queue reconfiguration, discussing differences between previous and current queue setup. Section 2.2 presents the new mechanism used to guarantee user-to-user fairness subject to heterogeneous users' requests. Next, the impact of queue reconfiguration on the overall performance is analyzed and the influence and suitability of the new fair-sharing mechanism is discussed in Section 3. Complex MetaCentrum workloads and the advanced job scheduling simulator Alea are presented in Sections 4 and 5 respectively. Section 6 concludes the paper and discusses the future work.

2 Reconfiguration of MetaCentrum Scheduling System

Production resource management systems need to satisfy the constraints imposed by resource providers, the expectations of users and must be robust enough to deal with short term fluctuations in the user base, its workloads and resource outages. In our previous work [10], we have discussed the complexities of configuring a production resource management system, such as PBS Pro and Torque, to satisfy all three of these requirements. The issue at the root of this problem is that production software is generally configured in a bottom-up fashion, meaning that the desired behavior is achieved through a combination of various policies.

The search for a new efficient setup of a resource management system is then particularly problematic as relatively straightforward configuration changes can have highly unexpected side effects arising from the interactions between individual policies. The choice of queue configuration can have significant effect on the way the scheduling algorithm selects jobs for execution [11], fair-sharing mechanisms that establish fair job order may be seriously diluted by both the scheduling algorithm [5] and the queue configuration [10], and too generous or too restrictive queue limits may either cause resource fragmentation or excessive resource saturation [16].

In this section we describe how we established the new queue configuration to enable higher job throughput and fairer scheduling (Section 2.1). Also, we briefly describe the newly applied multi-resource aware fair-sharing mechanism that reflects heterogeneity of resources and users' requests (Section 2.2).

2.1 Queue Reconfiguration

Detailed description of the queue reconfiguration, including the analysis of the historical setup, design and verification of a new configuration was already presented in our previous work [10]. Therefore, in this section we will only provide a summary of the core ideas that are required for the remainder of this paper and we kindly invite the reader to seek out our previous work for more details.

Mainstream resource management systems generally utilize the concept of queues to allow fine control over the systems behavior. Queue specific policies

include per-user, per-group and per-queue limits [1, 17] concerning the maximum number of running jobs and/or utilized amount of a particular resource type (e.g., CPU cores). Queues can also be configured to have access to only a subset of available resources, e.g., limiting a queue to a particular cluster of machines. This allows the establishment of resource pools, in which several queues compete for a limited set of resources, thus preventing a (potentially dangerous) saturation of the entire system. Of course, queues and their configuration can increase resource fragmentation [5] as each job is limited to a single pool of resources. This may however be necessary to deal with different classes of users and/or jobs accessing the system. We need to be very careful and avoid saturating the system with single class of jobs, as for example, saturating the system with long running jobs (i.e., jobs with expected runtime of several weeks) will lead to great deterioration in performance characteristics of the system, e.g., huge wait times for shorter jobs will be inevitable since they would have to wait until those long jobs would complete and free some resources.

Historical Queue Configuration For nearly a decade, MetaCentrum used one configuration, that only underwent small tweaks through the years. This configuration was originally designed manually by experts to fit the users' workloads at the time. The configuration was designed in a self-balancing manner, using overlapping resource pools with different sizes that were balanced out by queue priorities, with the highest priority queue having access to only the smallest resource pool. To achieve this, the system utilized three major queues (`long`, `normal`, `short`) each with a different maximum walltime limitation (30 days, 24 hours, 2 hours), different priorities (70, 50, 60) and different limits defining the maximum allowed number of concurrently running jobs of one user (70, 300, 250). Later (2010), a low priority (20) queue called `backfill` has been introduced, that only accepted single node jobs (max limit per user is 1000) that run up to 24 hours. It was designed for undemanding jobs and increases system utilization during off-peak hours. To provide a fair access to the system, jobs in these queues were dynamically ordered using priorities based on fair-share [5]. Next, queues were traversed one-by-one by the scheduling policy, starting with the highest priority queue (`long`). The scheme of the historical setup is shown in Fig. 2 (left).

After analyzing the behavior of this setup under the current users' workloads, we have determined that the major problem with this setup is the congestion of the `long` queue. To understand the reason we first must understand the self-balancing nature of the original setup. The `long` queue had to be limited to a relatively small pool of resources (1440 CPU cores) as increasing this pool would immediately lead to complete saturation of these resources with `long` jobs due to the high priority of the long queue. It was the combination of the small resource pool and the fact that the `long` queue was the only one accepting jobs longer than 1 day, that lead to the new inefficiency observed in the system. The users' workloads have shifted enough that the majority of the CPU time was now consumed by the `long` queue, despite the resource pool limitation. Shorter

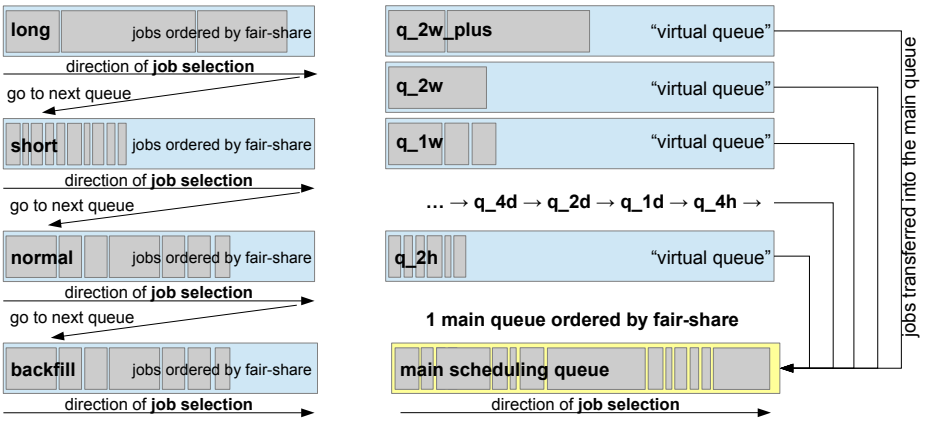

Fig. 2. Historical setup of queues (left) and the newly applied configuration (right).

jobs are much more frequent, as can be seen in Fig. 3, which shows job arrivals (top) and CPU time distribution (bottom) with respect to queues and time (on a weekly basis)[1]. As was observed, `long` queue only contained 2.75% of all jobs but produced 51.5% of overall CPU utilization. It was then very clear, that it should not have the smallest pool of available CPUs, but at the same time we could not simply increase the resource pool (for reasons mentioned above).

Applied Queue Reconfiguration Several proposals of new queue configuration have been considered and experimentally evaluated in a simulator [10]. The main goal was to increase the pool of available CPUs for longer jobs in a safe fashion. In the first step, `long` queue has been refined into 5 queues. The one with the longest maximum job walltime limit is called `q_2w_plus` (up to 30 days) and has the maximum priority. Next, there are `q_2w`, `q_1w`, `q_4d`, `q_2d` with decreasing priorities and walltime limits (2 weeks, 1 week, 4 days and 2 days, respectively). `Normal` and `short` queues are now called `q_1d` and `q_2h` while `q_4h` is a new queue with walltime limit being 4 hours. Once the `long` queue has been replaced with several new queues it was possible (and safe) to increase the number of available CPUs for selected newly created queues.

When setting up the new per-queue limits, several rules were applied that were based either on simulation results or our empirical knowledge. The first rule was that the number of available CPUs for a given queue should be — in general — inversely proportional to the maximum walltime limit of a given queue. In another words, it is safe to assign a large pool of resources to a queue that only executes short jobs, since those CPUs — if necessary — will be free soon (short jobs completes early). Also, the actual workload indicates that short jobs having their walltime $\leq$ 1 day are in fact the most common jobs in MetaCentrum (see

[1] Only major queues in the main system pool are considered. Auxiliary and specialized queues are omitted as well as all results coming from the second scheduler.

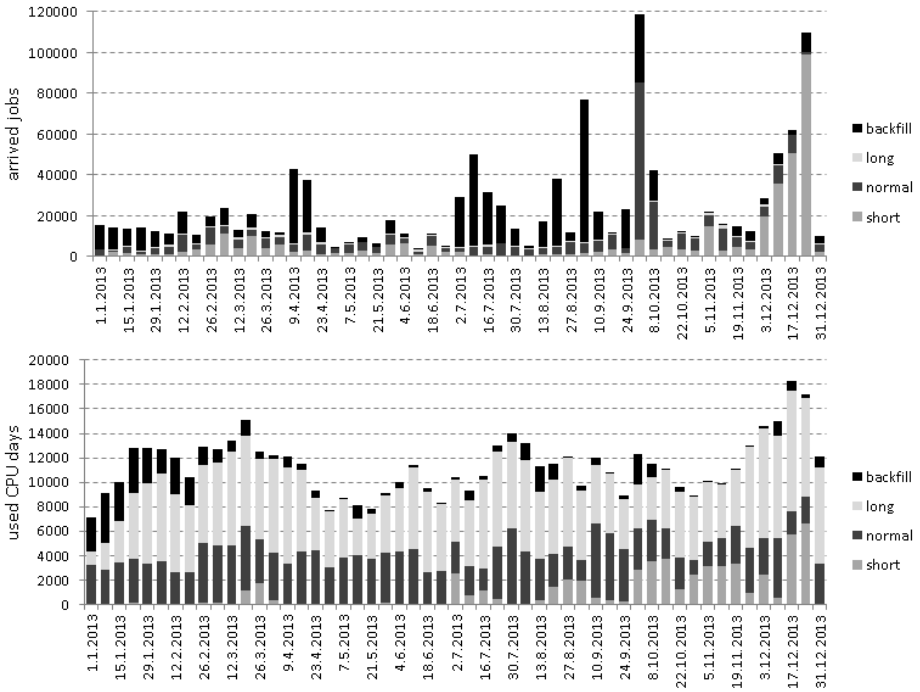

Fig. 3. Job arrivals (top) and used CPU time (bottom) per week and queue.

Fig. 4 (top)). On the other hand, it is very important to choose a rather conservative limit for long jobs as those may execute for weeks or even months, thus blocking resources over a long time period. Still, this "conservative" limit should be as high as acceptable, since long jobs are responsible for the majority of system utilization, at least this is the case in MetaCentrum (see Fig. 4 (bottom)). Last but not least, it is known that excessive number of queues with dedicated resources may cause resource fragmentation [5], leading to a low system utilization and large wait times. Therefore, whenever it was possible, resources were not dedicated exclusively to a given queue. Instead, several queues were allowed to compete for the same set of resources as their pools were overlapping. In such cases, it was observed that per-queue limits and fair-share are sufficient to balance "queue-sharing" of resources.

At the same time, the *effect of newly added queues on fairness* was considered as well. Using our complex workloads, we have performed detailed simulations which revealed that multiple queues with fixed ordering are very unfair and practically eliminate the impact of the fair-sharing algorithm. For example, if a job has a low priority (due to the fair-share) but ends up in a high priority queue (due to its expected walltime) it will often start much earlier than a high priority job residing in a low priority queue. Clearly, this is highly unfair. Therefore, the applied solution uses a little trick, where the queues are only used

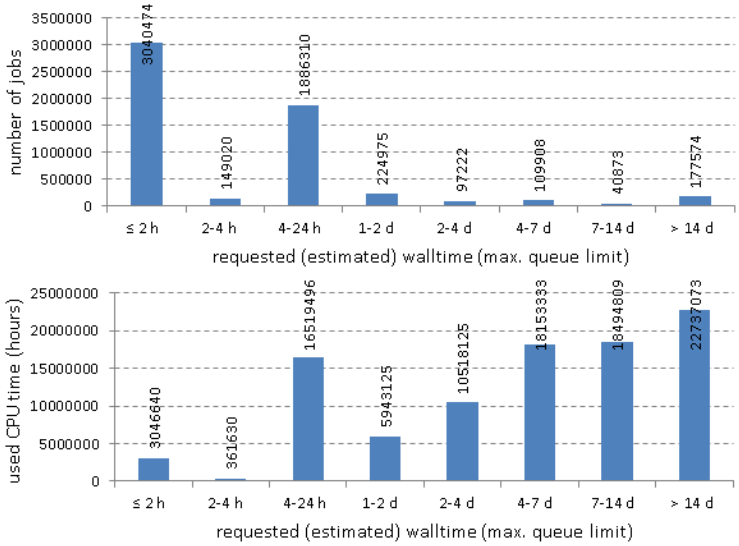

Fig. 4. Number of jobs according to their requested walltime (top) and corresponding used CPU time (bottom) wrt. to requested walltime during 2013 – 2014 period.

to (1) *maintain resource limits* and (2) *provide information on job's maximum walltime* (if not specified directly by a user). Otherwise, all (major) queues have the same priority, i.e., the ordering in which a job is being selected for execution is now solely based on the priority of a given user which is established by fair-share. Therefore, those queues are now only "virtual" and the actual scheduling process is performed over *one single queue* ordered by fair-share, which contains all jobs from those "virtual" queues. Fig. 2 (right) depicts the new setup of queues.

2.2 Multi-Resource Aware Fair-Sharing

MetaCentrum serves the scientific community and provides its resources for free. Therefore, *money cannot be used* to define the order in which users and their (pending) workload will use the resources [19]. Instead, user-to-user fairness is maintained by the well known *fair-sharing* [5] approach, which dynamically establishes fair user ordering.

Historical Fair-Sharing Algorithm Originally, the fair-sharing algorithm considered only single resource (allocated CPU time) and then calculated users priorities using the popular *max-min* approach [4], i.e., it assigned high priority to a user with low CPU time utilization and vice versa. As discussed in the literature, single-resource based fair-sharing is (highly) unfair for (highly) heterogeneous systems and workloads [4, 6, 9], which is the case of MetaCentrum. The most critical problem regarding the original CPU time-based fair-sharing

was that users with memory demanding workloads (and small CPU demands) were constantly favored by the scheduler, causing serious blocking of memory-heavy machines, poor fairness and low CPU utilization of large machines [9]. Therefore, the original single-resource aware fair-sharing algorithm was replaced by a newly developed multi-resource aware solution that also reflects the consumption of RAM memory.

Applied Multi-Resource Aware Fair-Sharing Algorithm Technical details as well as a detailed comparison with other existing techniques has been already published in our recent work [9]. Therefore, we will only briefly mention the main features of the newly applied solution. The new mechanism determines a user priority F_u based on CPU and RAM requirements of that user's (previously completed) jobs. F_u is computed by aggregating weighted walltimes of all jobs (J_u) of given user u (see Formula 1). Each job $walltime_j$ is weighted by so called job penalty P_j and machine speed factor S_j, which is used to reflect the influence of machine speed on resulting walltime of a completed job. S_j normalizes a job walltime such that job executed on a slow machine is not additionally penalized by its longer walltime (execution time), and vice versa. Job penalty P_j expresses the amount of allocated CPU and RAM resources (see Formula 2).

$$F_u = \sum_{j \in J_u} P_j \cdot walltime_j \cdot S_j \tag{1}$$

$$P_j = \min_{m \in M_j} \left(\max \left(\frac{cpu_j}{cpu_m}, \frac{ram_j}{ram_m} \right) \cdot cpu_m \right) \tag{2}$$

P_j penalty extends the well known Processor Equivalent (PE) metric [5], eliminating some serious problems related to job and resource heterogeneity. The major difference is that instead of calculating the penalty according to machines assigned to a given job (actual "price"), it calculates what is the minimal possible price (ideal price) according to the set of all suitable machines (M_j) and job requests (cpu_j, ram_j). Simply put, for each job the set of all suitable machines is constructed (M_j) and the "price" of executing that job is calculated for each machine in M_j. Finally, P_j is set to the minimal found price. Then P_j is independent of scheduler decisions and users have no reason to complain or cheat as they are guaranteed to obtain the best price. Once F_u priorities are calculated for all users, they are then ordered in the *lowest F_u first* order[2]. The actual implementation also *reflects aging* [5] by periodically decreasing all recorded consumption using the so called decay factor [1]. Using it we put higher emphasis on a more recent resource consumption.

[2] To be more precise, not users but their jobs in a queue are then ordered according to corresponding F_u values.

3 Results

This section analyzes the impact of queue reconfiguration on the overall performance of the MetaCentrum system and the influence and suitability of the new multi-resource aware fair-sharing mechanism.

3.1 Impact of Queue Reconfiguration

The new queue setup has been evaluated by comparing several statistical indicators using historical workload data from two consecutive time periods. The first period (October – December 2013) represented the old queue configuration while January – March 2014 period represented the new configuration. Both time periods lasted 92 days and the underlying infrastructure was identical during that time. We could not have used longer time periods, since those would contain several occasions when either old clusters were removed from the system or new ones were included. Obviously, such resource fluctuation would make the analysis less reliable. On the other hand, we do acknowledge that by comparing two setups of a production system in two distinct time periods, we inherently include differences in the underlying workloads, which could skew the presented results. Yet, we believe that the presented results are representative, as not only the metrics have shown improvements but also user feedback was positive.

We start with a comparison of the number of processed jobs which has increased significantly. During the October – December 2013 period, 513,976 jobs have been completed in MetaCentrum while in the January – March 2014 period (new queue configuration) 854,972 jobs were completed, representing an increase of 66.3%. At the same time, the overall CPU utilization has increased significantly (43.2%) as can be seen by the naked eye in Fig. 5, which shows the utilized CPU hours before and after queue reconfiguration. For simplicity, the average CPU time usage in those two periods is highlighted in the figure using dashed lines.

Fig. 6 (top) presents a closer look on the distribution of utilized CPU time. It reveals that the largest increase in utilized CPU time is visible for jobs having their walltime in the interval of 4 hours - 14 days. It confirms that the newly introduced queues are being used regularly and users of the system are able to recognize their benefits, e.g., larger pools of resources associated with these shorter queues[3].

Beside the overall utilization we have also analyzed job wait times which are an important factor, especially for the users of the system. It would not be surprising if the higher throughput and utilization caused that jobs are actually waiting longer. This is a real-life phenomenon originating from the fact that the system is more saturated, while users submit more jobs as they see the improved performance. However, as we have observed, even with a significantly larger throughput and utilization, job wait times remained decent. In fact, they

[3] As was explained in Section 2.1, `q_2w`, `q_1w`, `q_4d`, `q_2d`, etc. queues now have larger pools of available resources compared to the original `long` queue.

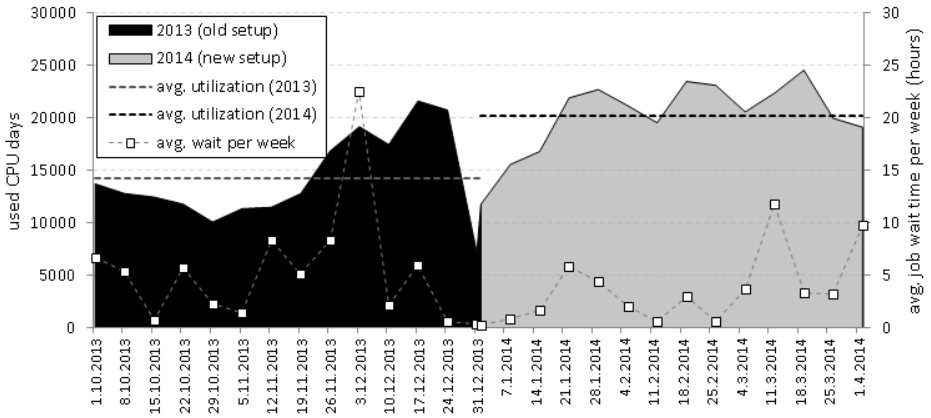

Fig. 5. Comparison of used CPU days (in a given week) before and after queue reconfiguration (left axis) and the average wait time per week (right axis).

were — on average — decreased by 17.9% (4.4 vs. 3.6 hours). A more detailed view is available in Fig. 5, where the average job wait time per week is shown (with the scale on the right side of the chart) along with the previously discussed average used CPU time. As we can see, the average wait time is bellow 5 hours on 11 occasions during January – March 2014. At the same time, there were only six weeks during October – December 2013 when the average wait time was bellow 5 hours. Certainly, this is a important finding which shows that the new configuration allows for higher throughput and utilization while keeping the wait times in an acceptable level. Another detailed view is presented in Fig. 6 (bottom). It shows how jobs are distributed with respect to their wait times. As we can see, the new queue configuration leads to a shorter wait times for majority of the jobs.

3.2 Impact of Multi-Resource Aware Fair-sharing

In the next step, we have analyzed the influence of the new fair-sharing mechanism. Again, we have used historical workload traces from the October – December 2013 (old, single-resource fair-sharing) and January – March 2014 (new, multi-resource aware fair-sharing) periods.

First of all, we have plotted all jobs coming from the January - March 2014 period according to their CPU and memory requirements, as shows Fig. 7 (top). For better visibility, both the x-axis and the y-axis are in log. scale. As can be seen, the workload from MetaCentrum is truly heterogeneous. For example, a job requesting 1 CPU may have its memory requirements anywhere between 1 GB and 2 terabytes of RAM[4]. Fig. 7 (top) demonstrates the huge heterogeneity

[4] Jobs requesting less than 1 GB of RAM are not shown in Fig. 7 as they would end up "bellow" the baseline of the log. scale-shaped graph.

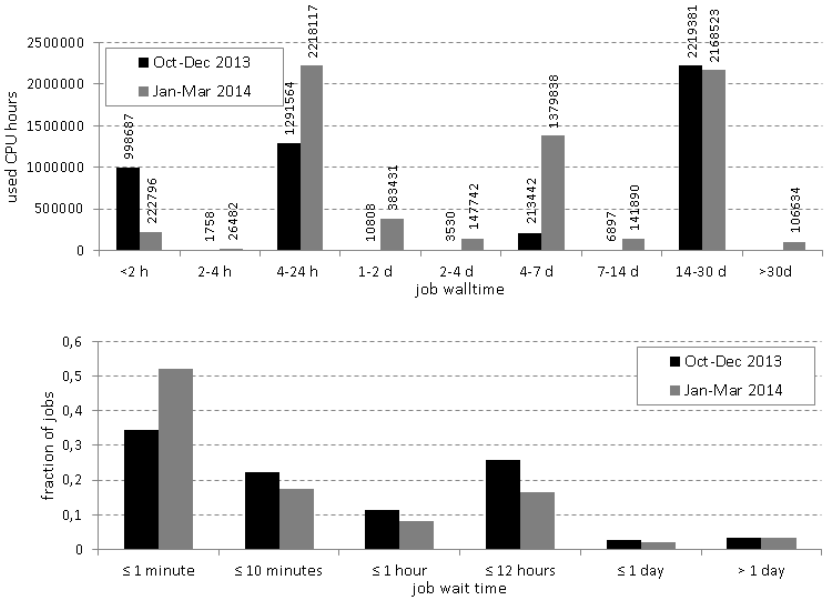

Fig. 6. Comparison utilized CPU hours wrt. to job walltime (top) and the distribution of job wait times before/after queues were reconfigured (bottom).

of job requirements, which was the reason why the historic single-resource based fair-sharing was impractical, i.e., extremely unfair.

In the second step, we have analyzed the workload from January – March 2014 period and selected *all jobs that were affected* by the new fair-sharing algorithm, i.e., their "fair-sharing penalty" P_j was different when computed according to the new multi-resource aware scheme. Those *affected jobs* are shown in Fig. 7 (bottom). As we can see, the new multi-resource aware penalty function works as ones intuition would suggest, i.e., higher penalties are assigned to those jobs that request large amounts of RAM compared to their CPU requirements. With a few exceptions[5], the new fair-sharing algorithm targets jobs lying "above the main diagonal", i.e., those that have high RAM to CPU ratio, which is the expected behavior.

We have also analyzed the increase of penalty values. For this purpose we took jobs affected by the new penalty $P(j)$ and measured the resulting percentage increase of $P(j)$ value with respect to the old, CPU-based version. Fig. 8 shows the resulting distribution using the cumulative distribution function (the x-axis is in log. scale). In this case, the CDF is a $f(x)$-like function showing the probability that the percentage increase of $P(j)$ for a given job j is less than or equal to

[5] Those exceptions are jobs lying under the main "diagonal", i.e., in the lower central/right part of the plot. Such exceptions were expected as the new fair-sharing scheme may also (rarely) assign smaller penalties compared to the original single-resource aware mechanism.

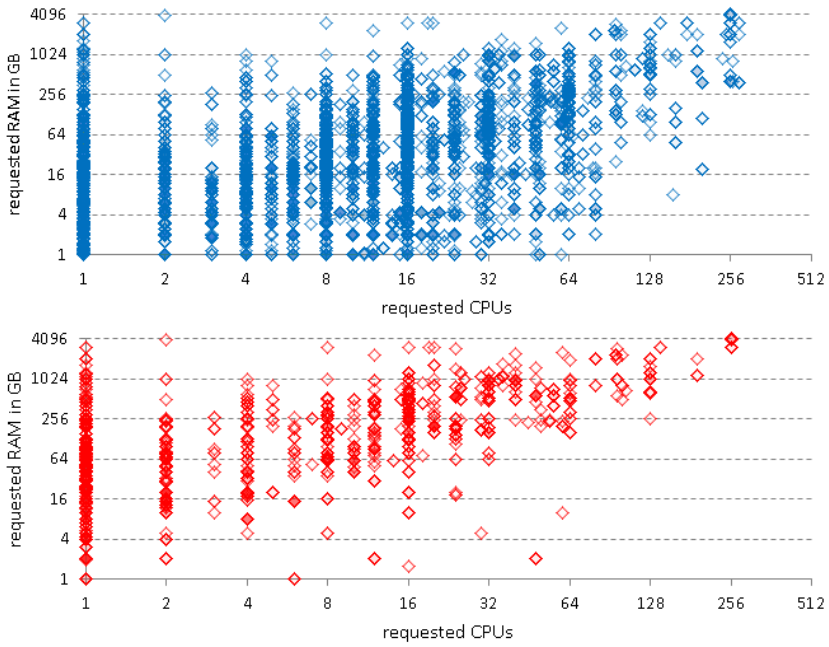

Fig. 7. The distribution of jobs requirements during January – March 2014 (top) and the corresponding jobs affected by the new fair-sharing mechanism (bottom).

x. In another words, the CDF represents the fraction of jobs having their $P(j)$ increased by at most x percents. As can be seen, the improvement is often significant. For example, nearly 60% of affected jobs have their $P(j)$ at least two times higher (penalty increase $\geq 100\%$).

In the next step, we have analyzed the impact of the new multi-resource aware fair-sharing mechanism on the performance of affected jobs. This time, we have compared those two time periods: October – December 2013 (old fair-sharing) and January – March 2014 (new fair-sharing). Again, we have selected those jobs that were affected by the new multi-resource aware fair-sharing scheme[6]. Then we have computed the average wait time (and its standard deviation) of such affected jobs for both periods, i.e, before and after the new fair-sharing was deployed.

Fig. 9 shows the results of such a comparison. Apparently, with the new multi-resource aware fair-sharing algorithm the average wait time of affected jobs is significantly larger (18.3 vs. 11.4 hours). At the same time, standard deviations of wait times are similar in both situations which indicates that the overall increase of wait times is not accidental (a result of few extremes), instead

[6] In case of the earlier period (October – December 2013) — which did not use the new fair-sharing mechanism — these affected jobs were detected using the Alea job scheduling simulator which is capable of emulating the new fair-sharing method.

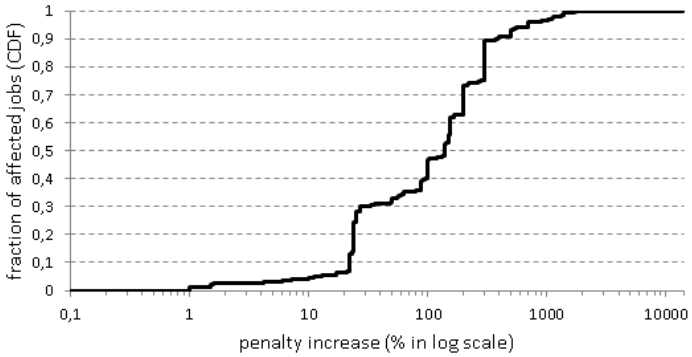

Fig. 8. The CDF showing the increase of penalty value (P_j) for affected jobs according to the new fair-sharing mechanism.

Fig. 9. Comparison of avg. wait times wrt. the old and the new fair-sharing mechanism.

it is a common tendency. It means that the new fair-sharing is working as intended, appropriately assigning higher penalties to (memory) demanding jobs, thus prolonging their wait times.

3.3 Summary

The evaluation presented above was based on real-life data coming from MetaCentrum. So far, the results indicate that the two applied modifications, i.e., queue reconfiguration and new fair-sharing algorithm work as intended. First of all, thanks to the newly configured queues the overall throughput and utilization have increased significantly. At the same time, the average wait times were decreased. Also, the new multi-resource aware fair-sharing mechanism works better than the original mechanism, since RAM demanding jobs now obtain appropriate penalties as the consumption of RAM memory is considered when computing a user priority. Unlike in the old fair-share, RAM intensive jobs are now penalized similarly to CPU demanding jobs, resulting in a more fair behavior of the sys-

tem. So far, no significant comments concerning the new fair-sharing approach were recorder from either the users or the management team of MetaCentrum.

4 MetaCentrum Workload Traces

One of the contributions of this paper is that we are offering the scientific community a complex workload trace from the MetaCentrum system. This workload starts in January 2013 and represents 2 years of job execution in MetaCentrum, containing 5.8 millions jobs.

We believe that this workload may be valuable for several reasons. First of all, MetaCentrum is a very heterogeneous environment. It contains variety of resources, starting with small nodes (8 cores with 16 GB of RAM per node) and going up through moderate nodes (16-64 cores with 64-256 GB of RAM per node) to large and RAM-heavy machines (80-384 cores with 0.5-6.0 TB of RAM). Beside common clusters, MetaCentrum also provides 3 GPU-enabled clusters (*konos*, *doom* and *gram*) for CUDA-like computations. We have prepared a detailed resource description file, which contains information about each cluster. Here we specify the number of nodes, number of CPU cores per node, the amount of RAM per node and the results of the Standard Performance Evaluation Corporation's SPEC CPU2006 benchmark (CFP2006 suite/fp_rate_base2006). Furthermore, the availability of MetaCentrum's clusters is provided too.

Similarly, jobs in the workload vary accordingly. The majority of jobs (71%) is sequential while parallel jobs represent 29% of all jobs. On the other hand, sequential jobs represent only 10% of used CPU time as parallel jobs use 90% of CPU time. A more detailed view showing how jobs and CPU time are spread over existing queues with respect to job parallelism is presented in Fig. 10, where x-axis represents job parallelism and y-axis represents number of jobs and used CPU time, respectively. The y-axis is in log scale in both cases. Fig. 10 (top) shows that the "shape" of distribution of job parallelism is similar for all queues, and most jobs (88%) belong to "short" queues (walltime $\leq$ 24 hours). On the other hand, Fig. 10 (bottom) shows that "long" (walltime $>$ 24 hours), parallel jobs are those that are responsible for the majority of used CPU time. The distribution of jobs and CPU time with respect to walltimes (i.e., queues) can be found in Fig. 4, showing that "long" jobs — which represent only 12% of all jobs — are responsible for 80% of used CPU time. Finally, an example of the variability of job CPU and RAM requirements is shown in Fig. 7 (top).

The job workload is presented in more than usual detail. Beside common parameters that are routinely provided, e.g., in the Standard Workload Format (SWF) [3], we provide additional job specifications that influence job scheduling and allow for more detailed simulations and analysis. Here we use input parameters of the `qsub` command. For example, `2:ppn=4:x86:linux:cl_minos` input parameters mean that the job is requesting 2 nodes, with 4 processors per node (ppn). Both nodes must be operated by linux-like OS, lie within *minos* cluster and have x86 architecture. Similarly, `1:ppn=1:gpu=1:cl_gram` means that the

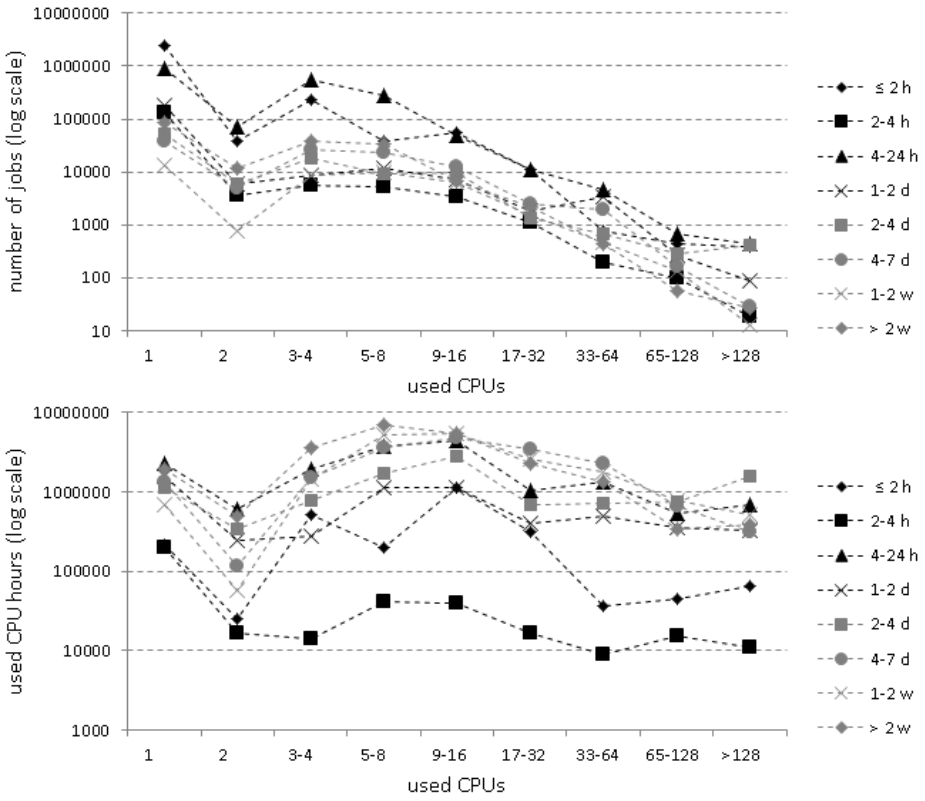

Fig. 10. The number of jobs per queue (top) and the utilized CPU time per queue (bottom) with respect to job parallelism.

job can only be executed on cluster *gram* and requires 1 node with 1 CPU and 1 GPU card[7].

Last but not least, information about queues, their priorities and per-user CPU limits are provided as well. The whole job workload formatted in an extended SWF format as well as related information concerning resources and queues can be obtained at: `http://www.fi.muni.cz/~xklusac/jsspp/`.

5 Job Scheduling Simulator

Designing a well working scheduler for HPC, Cloud or a Grid-like system is a complex task. One needs not only to consider the workloads the system will need to process, but is also constrained by the requirements of the resource

[7] A detailed description of `qsub` semantics is available at: `https://wiki.metacentrum.cz/wiki/Running_jobs_in_scheduler`.

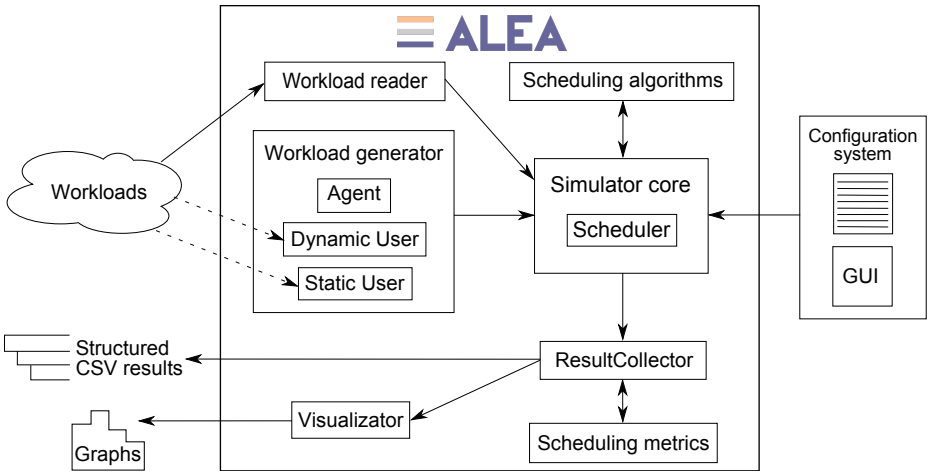

Fig. 11. The structure of the Alea simulator.

providers and users. Simulators can simplify this task by allowing fast iterations over different system configurations.

In MetaCentrum, simulations are used regularly for testing new setups and features of the scheduling system as well as for designing new scheduling algorithms. For this purpose, *Alea job scheduling simulator* [8] based on GridSim [21] has been developed and is continuously upgraded [18]. It provides advanced features that allow for detailed simulations. A high-level scheme of the structure of the simulator is shown in Fig. 11.

The major role is played by the *scheduler* entity which represents the centralized scheduler. The scheduler holds current simulation data, handles communication and delegates scheduling decisions to a chosen scheduling algorithm. Variety of scheduling algorithms is provided, including trivial First Come First Served and its prioritized versions such as Shortest Job First, Earliest Deadline First, etc. Also policies using backfilling are supported, including aggressive backfilling (no reservations), EASY backfilling [20] and Conservative backfilling [15].

Of course, a simulator cannot work properly unless it is supplied with an appropriate workload. Therefore, parsers for common workload formats are provided. However getting access to a historical workload that fits the expected workload of the system is often a complicated process. One possible solution to the problem of finding a matching workload is to simply generate one. Solutions for generating workloads, with varying degrees of complexity, have been available for some time [14]. From models based on statistical analysis [12] that generate jobs fitting a particular parameter distribution to dynamic models that react to the behavior of the evaluated scheduler [22]. Alea takes a step ahead and provides a *dynamic workload generator* which extends existing approaches further and concentrates on modeling the behavior of users in the system using *user agents*. Agents have access to scheduling information and therefore react

to stimuli, such as a completion of a job. This allows us to model different user behavior from very batch-oriented users that submit sets of jobs and wait for the entire batch to complete, to interactive users that submit one job at a time, wait for its completion, process the results and then submit a new job. Realistic modeling of day-night and week-weekend cycles is matter of course. This approach allows for more thorough testing of scheduling setups, but also enables testing of hypothetical scenarios. Determining how the system will behave if we add, e.g., another user, becomes a matter of modifying a configuration file.

During the simulation, the *result collector* entity collects the data and — using either default or user-provided metrics — stores results into CSV files and (optionally) uses them for visualization. Importantly, Alea supports various fair-sharing policies allowing for simulations where fairness is of importance. Also, various queues including limits as well as complex job specifications (see Section 4) are supported, resulting in much more realistic simulations.

This complexity of simulation capabilities resulted in a newly designed configuration system [18]. In a user-friendly fashion, it allows to adjust parameters of simulations by providing an intuitive way how to choose different data sets, scheduling algorithms, measured metrics as well as additional features, e.g., the type of fair-sharing algorithm. Alea is freely available at GitHub [2].

6 Conclusion and Future Work

In this paper we are sharing our experience with a *major reconfiguration* of the job scheduling system in MetaCentrum. Our analysis measures the impact of two major modifications — the queue reconfiguration and the new multi-resource aware fair-sharing algorithm. Both the results and the feedback from the users of the system indicate that the reconfigured system is more efficient than it was previously. First of all, thanks to the new queue setup the throughput is now much larger while job wait times remained decent. Concerning the new fair-sharing algorithm, the results revealed that it works as intended, assigning higher penalties to jobs with large requirements concerning RAM. The effect of increased penalties was observed as well, i.e., wait times of RAM-heavy jobs have increased compared to the period when single-resource based fair-sharing was used. Last but not least, we are presenting our jobs scheduling simulator and the complex workload traces from MetaCentrum to the scientific community.

Still, our work has some limitations. Although we have mentioned some general rules (see Section 2.1), values of several (important) parameters such as queue-related limits are currently based on an empirical knowledge or an (hand-tuned) expert assessment. In the future we would like to develop more rigorous methods that would allow for a (semi)automatic identification of proper and efficient system setups. For starters, it would be very helpful to have some method that — given a current workload — would perform a dynamic adaptation of various queue-related limits.

Acknowledgments. We highly appreciate the support of the Grant Agency of the Czech Republic under the grant No. P202/12/0306. The support provided by the programme "Projects of Large Infrastructure for Research, Development, and Innovations" LM2010005 funded by the Ministry of Education, Youth, and Sports of the Czech Republic is highly appreciated. The access to the MetaCentrum computing facilities and workloads is kindly acknowledged.

References

1. Adaptive Computing Enterprises, Inc. *Maui Scheduler Administrator's Guide, version 3.2*, January 2014. `http://docs.adaptivecomputing.com`.
2. Alea job scheduling simulator, February 2015. `https://github.com/aleasimulator/`.
3. S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1659 of *LNCS*, pages 67–90. Springer, 1999.
4. A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *8th USENIX Symposium on Networked Systems Design and Implementation*, 2011.
5. D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In D. G. Feitelson and L. Rudolph, editors, *Job Sched. Strategies for Paral. Proc.*, volume 2221 of *LNCS*, pages 87–102. Springer, 2001.
6. C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *31st Annual International Conference on Computer Communications (IEEE INFOCOM)*, pages 1206 – 1214, 2012.
7. D. Klusáček, V. Chlumský, and H. Rudová. Planning and optimization in TORQUE resource manager. In *High Performance and Distributed Computing (HPDC)*. ACM, 2015. To appear.
8. D. Klusáček and H. Rudová. Alea 2 – job scheduling simulator. In *3rd International ICST Conference on Simulation Tools and Technique*. ICST, 2010.
9. D. Klusáček and H. Rudová. Multi-resource aware fairsharing for heterogeneous systems. In *Job Scheduling Strategies for Parallel Processing*, volume 8828 of *LNCS*, pages 53–69. Springer, 2015.
10. D. Klusáček and Š. Tóth. On interactions among scheduling policies: Finding efficient queue setup using high-resolution simulations. In F. Silva, I. Dutra, and V. S. Costa, editors, *Euro-Par 2014*, volume 8632 of *LNCS*, pages 138–149. Springer, 2014.
11. B. G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In *In Job Scheduling Strategies for Parallel Processing*, pages 72–87. Springer Verlag, 2002.
12. U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *J. Parallel & Distributed Comput.*, 63(11):1105–1122, 2003.
13. MetaCentrum workload logs, February 2015. `http://www.fi.muni.cz/~xklusac/workload/`.
14. Parallel workload models, February 2015. `http://www.cs.huji.ac.il/labs/parallel/workload/models.html`.

15. A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.
16. Ohio Supercomputer Center. *Batch Processing at OSC*, February 2014. `https://www.osc.edu/supercomputing/batch-processing-at-osc`.
17. PBS Works. *PBS Professional 12.1, Administrator's Guide*, January 2014. `http://www.pbsworks.com`.
18. G. Podolníková. Configuration and presentation system of job scheduling simulator, 2014. Bachelor's thesis. `http://is.muni.cz/th/396214/fi_b/Gabriela_Podolnikova_BP.pdf`.
19. P. Sempolinski and D. Thain. A comparison and critique of Eucalyptus, OpenNebula and Nimbus. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 417–426. IEEE Computer Society, 2010.
20. J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY – LoadLeveler API project. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *LNCS*, pages 41–47. Springer, 1996.
21. A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice & Experience*, 20(13):1591–1609, 2008.
22. N. Zakay and D. G. Feitelson. Preserving user behavior characteristics in trace-based simulation of parallel job scheduling. In *22nd Modeling, Anal. & Simulation of Comput. & Telecomm. Syst. (MASCOTS)*, pages 51–60, 2014.