# On Interactions among Scheduling Policies: Finding Efficient Queue Setup Using High-Resolution Simulations

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

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

**Abstract.** Many studies in the past two decades focused on the problem of efficient job scheduling in HPC and Grid-like systems. While many new scheduling algorithms have been proposed for systems with specific requirements, mainstream resource management systems and schedulers are still only using a limited set of scheduling policies. Production systems need to balance various policies that are set in place to satisfy both the resource providers and users (or virtual organizations) in the system. While many works address these separate policies, e.g., fairshare for fair resource allocation, only few works try to address the interactions between these separate solutions. In this paper we describe how to approach these interactions when developing site-specific policies. Notably, we describe how (priority) queues interact with scheduling algorithms, fairshare and with anti-starvation mechanisms. Moreover, we present a case study describing how an advanced simulation tool was used to find new configuration for an actual resource manager deployed in the Czech National Grid, significantly increasing its performance.

**Keywords:** Scheduling, Queues, Fairshare, Simulation.

## 1 Introduction

For many years, researchers have been searching for a perfect job scheduling algorithm that would improve the performance of HPC and Grid-like systems. Still, there are few algorithms that are being used in practice [18] as can be seen in many production schedulers applied in nowadays general resource management systems. For example, the core of the system is generally based on the trivial first come first served (FCFS) approach and backfilling is typically the most advanced option available [2,1,17,18]. Since backfilling has been proposed in 1995 [13], it is obvious that there is some misunderstanding between the research community and system administrators concerning "what is really important".

In this paper we show that the problem of operating a production scheduler is far more complex than just choosing a proper scheduling algorithm. Using our experience from Czech National Grid Infrastructure *MetaCentrum* [14] we

F. Silva et al. (Eds.): Euro-Par 2014, LNCS 8632, pp. 138–149, 2014.

explain several additional challenges that appear when searching for a functional solution. These problems are related to the fact that real systems must meet far more complicated requirements than those that are typically considered in classical research papers. For example, real life systems have to focus on maintaining fairness among users of the system [9,19], rather than just trying to optimize simple criteria like the average slowdown or makespan. In practice, it quickly turns out that those widely used "theoretical" models and optimization goals are mostly impractical in real life [5,18].

The contribution of this paper is based on our ability to provide detailed insight into a real, complex job scheduling system. In detail, we explain several important features that current resource managers offer to the system administrator in order to establish robust, efficient and fair computing infrastructure (Section 2). In Sections 3 and 4, we provide a real life example from MetaCentrum, describing how the actual resource manager has been reconfigured in order to increase the overall performance and fairness. Furthermore, Section 5 demonstrates how advanced simulation and evaluation tools can be used to evaluate new possible setups of complex scheduling systems prior actual deployment. We conclude the paper in Section 6.

## 2 Main Components of a Resource Management System

Resource managements systems are rather conservative in their choices of scheduling policies and mostly rely on well established and robust approaches [18]. The desired overall behavior is then achieved through the interactions of a chosen set of policies and additional mechanisms. This section describes these commonly employed components of resource management systems and their impacts.

### 2.1 Ordering Policy

Ordering policies determine the order of jobs in which they are then processed by a scheduling policy. Resource management systems usually provide a set of static ordering policies (ordering between two jobs does not change once established) as well as dynamic policies. Jobs can be either kept in the order of their arrival (static ordering), or can be ordered dynamically according to their length (Shortest Job First, Longest Job First), according to their resource requirements (Largest CPU/Memory Requirements First,...) or their (user configured) priority. Combinations of ordering policies are also possible [1,7].

*Fairshare* is a dynamic priority ordering policy designed to provide user-to-user fairness. Job ordering is usually based on users previous resource consumption [7,12]. Typically, the more resources a user consumes the lower her priority becomes. Fairshare self-balances itself around an equilibrium where all users have consumed the same amount of resources. Practical implementations of fairshare also *reflect aging* [7] by periodically decreasing all recorded consumption using so called decay factor [1]. This is suitable for systems with faster job turnaround times that put higher emphasis on more recent resource consumption.

### 2.2 Scheduling Policy

Commonly used scheduling policies range from trivial FCFS, aggressive backfilling (no reservations), to EASY [13] or Conservative backfilling [7], each with it's own shortcomings. *FCFS* guarantees the execution of jobs in the order of arrival by considering the first job only (provided by the ordering policy). FCFS will wait until the first job can be executed and only then continues processing the rest of the jobs. *EASY backfilling* [13] builds on top of FCFS but instead of strictly following the job order as mandated by the ordering policy it only guarantees the earliest possible start for the first job. Other jobs are allowed to start, as long as they do not interfere with the first job's reservation. *Conservative backfilling* extends EASY by providing reservation for every job that cannot start immediately. Remaining jobs are allowed to start as long as they do not interfere with any previously established reservation. The notions of "first job" and the order of jobs are mandated by the ordering policy as was described in Section 2.1.

*Job starvation* is an undesirable process where a particular job (or a user) is subject to excessive wait time due to the presently configured policies. The notion of excessive is of course subject to interpretation. For example, fairshare ordering priority will deliberately cause starvation of users with recent high resource consumption, which is however considered desirable. FCFS and Conservative backfilling algorithms provide anti-starvation mechanisms, guaranteeing that jobs are not undesirably delayed. More aggressive forms of backfilling like EASY or aggressive backfilling need to be combined with other mechanisms in order to prevent starvation, as they can delay the execution of certain jobs without any bounds [15].

### 2.3 Queue Configuration

Previously presented policies provided by resource management systems are relatively simple. At the same time, a single policy cannot cover the usually complex requirements used in production systems. To deal with more complex requirements, resource management systems provide the notion of queues which can be configured separately. Then, it is the interaction between queue-specific policies and the global system policies that dictates the overall behavior of the system.

Queues can handle different policies, that are mostly represented by a set of various limits [1,2,17]. These limits then apply on jobs that are executed from that queue. The limits usually cover per-user, per-group and per-queue limitations concerning the maximum number of running jobs and/or amount of 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. Such policy establishes pools of resources, where several queues can compete for a limited set of resources, thus preventing a (potentially dangerous) saturation of the entire system.

While such configuration can increase resource fragmentation [7], it is necessary when dealing with different classes of users accessing the system. We need to

be very careful when saturating the system with jobs from a single user, or even when saturating the system with a single class of jobs. For example, saturating the system with long running jobs (i.e., jobs with expected runtime of several weeks) will naturally lead to great deterioration in performance characteristics of the system (e.g., huge wait times for shorter jobs).

Such situations are approached in different manners. For example, in Zeus cluster in PL-Grid, all long jobs as well as jobs that require whole node(s) are planned ahead using reservations which enables the forward detection of potential problems [4]. In Ohio Supercomputer Center several combined approaches are used together. For example, long serial jobs are only allowed if a user is able to reasonably explain why he or she needs to run such a long experiment [16]. Moreover, parallel jobs have in general smaller maximal runtime limit compared to sequential jobs. Also, per-user and per-group limits are used together with fairshare accounting [16].

Surprisingly, we are not aware of any work that would describe how to determine suitable combinations of global policies and queue configurations. Clearly, a more in-depth analysis must be performed to better understand these issues. We provide such a case study in the following text.

## 3 Configuration of MetaCentrum Resource Manager

So far, we have provided an overview of several techniques that are available in current resource management systems. In the remaining text we demonstrate how these techniques interact together. We also describe how existing setup can be significantly improved by proper reconfiguration, using a real-life based example from MetaCentrum (Sections 4 and 5).

Before we start, we would like to stress out that there is no widely accepted and universal definition describing "the one and only suitable setup of the system". In fact, different people and/or organizations may have different notion of "what is efficient" when it comes to job scheduling. In this paper, we use examples coming from the Czech NGI MetaCentrum. The approaches and solutions presented in the following sections are presented in the context of this system. Still, we believe that they are applicable to a wide range of systems.

### 3.1 Historical Setup

Historically, MetaCentrum used three major queues (`long`, `normal`, `short`) that had different maximum walltime limits per job (30 days, 24 hours, 2 hours), different priorities (70, 50, 60) and different limits on maximum running concurrent jobs of one user (70, 300, 250). Together with the user limits, `long` and `short` queues were also limited to a subset of machines. Using the combination of priorities, user limits and limited resource pools the system originally provided balanced performance for each of the three job classes (under 2 hours, 2-24 hours, up to 30 days). There was also a low priority (20) queue called `backfill` that only accepted single node jobs (max limit per user is 1000) that run up to 24

hours. Beside these, there were several other queues for special purposes, e.g., administrator's testing queue. Still, majority of jobs used those 4 main queues. A scheme of the historical setup is shown in Fig. 1.

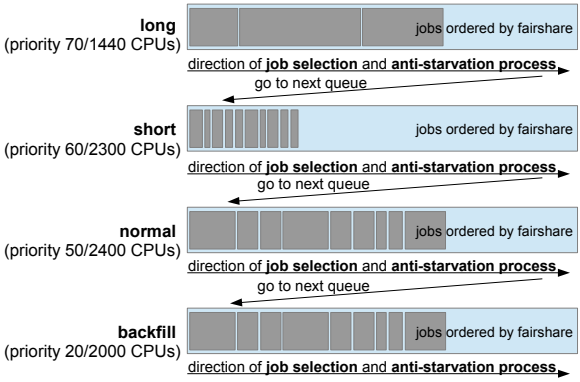


**Fig. 1.** Historical queues setup as applied in MetaCentrum

Jobs were dynamically ordered within queues using priorities based on fairshare [7]. A backfill-like algorithm was used to scan the queues, starting with the highest priority queue. It immediately started every job that could execute. Those jobs that could not start immediately received reservations using an *anti-starvation* mechanism (see Section 2). A reservation blocked every node that was potentially suitable to execute a job, that is any node that is capable of providing the requested amounts of resources and properties. This approach has been applied as classical reservations computed according to estimated completion times of jobs were very imprecise. This was caused by the fact that users of the system often did not provide detailed runtime estimates, instead simply choosing one of the job classes available (under 2 hours, 2-24 hours, up to 30 days). By reserving all suitable nodes the scheduler was able to guarantee the earliest possible start time, at the cost of decreasing opportunities for backfilling.

### 3.2 Problems with Historic Setup

The major problem with the historic setup was that *it only used one queue for jobs longer than 1 day*. Therefore, this queue had to be used by every job that was expected to last longer than 24 hours. At the same time, it was also used by very long jobs that are "dangerous" as we have explained in Section 2.3. Therefore, the queue had quite strict limits concerning number of available CPUs (1440), while `short`, `backfill` and `normal` had significantly larger pools of CPUs (2300, 2000 and 2400, respectively). While such a restriction was necessary, it was obvious that it limits efficient usage of resources.

For example, our historic workload logs indicated that majority of utilized CPU time was based on jobs from `long` queue. An example of job arrivals and

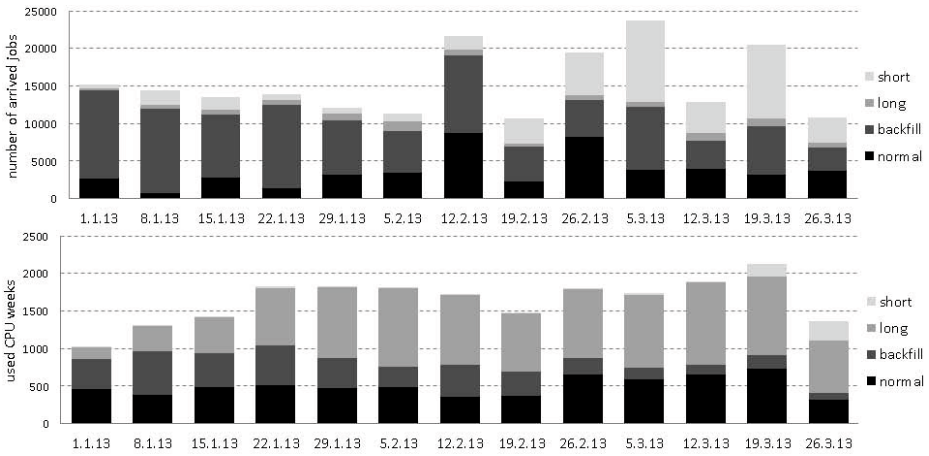


**Fig. 2.** Job arrivals (top) and used CPU time (bottom) per week and queue

CPU time distribution with respect to queues is shown in Fig. 2. Clearly, `long` queue, having the least CPUs was at the same time responsible for the most of the overall utilization (see Fig. 2 (bottom)).

## 4 Proposed Modifications of the Scheduling Scheme

After performing detailed analysis of historic workloads, MetaCentrum management decided that a new setup of the whole scheduling system must be developed. We now present main features of the two new setups that were proposed and evaluated (Section 5), in order to remove aforementioned inefficiencies.

### 4.1 Conservative Extension

The first considered modification was rather conservative. The main goal was to increase the pool of available CPUs for longer jobs. 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. The scheme of the system with newly refined queues is shown in Fig. 3.

Once the `long` queue has been replaced with several new queues it is now possible (and safe) to increase the number of CPUs for selected newly created queues as is shown in Fig. 3. Importantly, we have significantly increased the number of CPUs for jobs lasting at most 2 weeks, while very long jobs (`q_2w_plus`) obtain at most 1024 CPUs[1]. No other modifications were considered in this scheme.

[1] Different queues may share some CPUs, i.e., in general, CPUs available for a given queue are not exclusively reserved for such a queue.

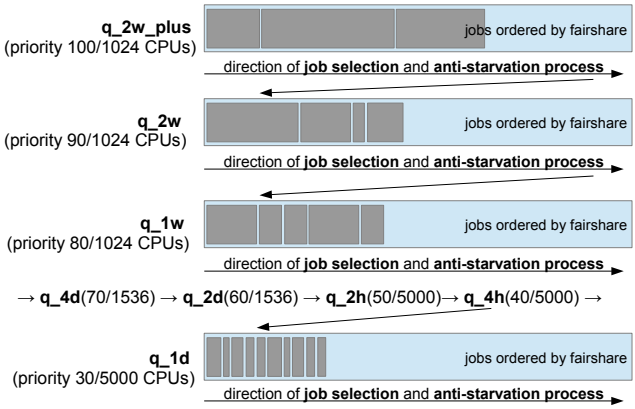


**Fig. 3.** The scheme of queues with refined walltime limits

### 4.2 Complex Extension

While the conservative modification described in the previous section was rather simple and straightforward, we also tried to develop a more complex modification that would also address overall fairness and efficiency of the anti-starvation mechanism.

Concerning fairshare, we have replaced the original single-resource aware mechanism that only reflected CPU consumption with a new multi-resource aware solution that also reflects RAM consumption. As discussed in the literature, single-resource based fairshare is highly unfair for heterogeneous systems and workloads [6,8,12]. Beside the fairshare metric itself, we have also started to consider the *effect of newly added queues on fairness*. For example, if a job has low priority (due to the fairshare) but ends up in a high priority queue (due to its walltime) it will often start much earlier than a high priority job residing in a low priority queue, which is highly unfair. Therefore, we have proposed more complex modification of the scheduling scheme, which extends the previous conservative, multi-queue setup. In this case, the queues are only used to (1) *setup CPU limits* and (2) *provide information on job's maximum walltime* (if not specified directly by a user). All (major) queues have the same priority, i.e., the ordering in which a job is being selected for execution is now only based on a given user's fairshare. Therefore, those queues are now only "virtual" and the actual scheduling process is performed over *one single queue* that contains all jobs from those "virtual" queues, as depicted in Fig. 4.

In the second step, we have proposed a modification of the anti-starvation mechanism. So far, all suitable nodes were reserved for starving job (see Section 3.1), which often led to resource wasting. Since the queues are now more fine-grained with respect to maximum job runtime, we can compute estimated job completion times far more precisely and only reserve those CPUs that are expected to be the soonest available. The calculation of reservations uses runtime

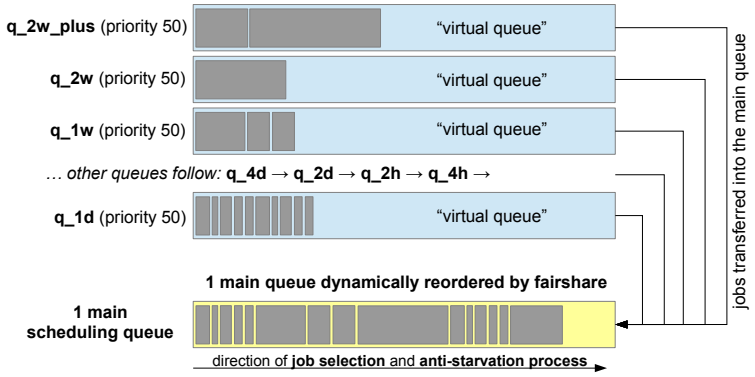


**Fig. 4.** "Virtual queues" with only 1 main scheduling queue managed by fairshare

estimates (or refined queue walltime limits) of currently running jobs. Reservations are updated in every scheduling cycle with respect to dynamic changes like early completing jobs or changes in fairshare-based priorities.

## 5 Experimental Evaluation

The two possible modifications of the scheduling scheme described in Section 4 were experimentally evaluated through detailed simulations. It must be said that according to MetaCentrum management, the conservative extension was the prime candidate to become the new production setup in MetaCentrum. The intuition within the management was that *it is a simple and safe evolution of the historical setup*. On the other hand, we believed that the complex extension was more suitable for our purposes, as it introduces new and important features including *multi-resource fairshare and optimized anti-starvation mechanism*. Therefore, it was necessary to perform detailed simulations, analyzing pros and cons of these two candidates.

### 5.1 Simulation Environment

The simulations were performed using our GridSim-based job scheduling simulator *Alea* [10]. Alea provides advanced capabilities that allow for very detailed and complex simulations. These capabilities include support of several scheduling algorithms, complex job specifications (based on standard `qsub` syntax used in real systems), multi-resource aware fairshare policies, multi-queue setups including related limits, etc. Alea is regularly used in MetaCentrum to test new setups prior their deployment in the production service.

### 5.2 Simulation Results

The simulations used a historic workload from MetaCentrum, covering 5 months of execution in 2013. This workload contains 376,722 jobs coming from 302

different users and is publicly available at: `www.fi.muni.cz/~xklusac/workload`. Due to the space limitations, we only present the most important findings related to performance and fairness.

The initial comparison considered all 3 scenarios (historic, conservative and complex). The avg. weighted wait/response time (AWWT/AWRT) [3] and the avg. weighted slowdown (AWSD) [3] were used to measure the general performance. These metrics are weighted by jobs CPU consumption to prevent that smaller jobs have a relatively larger impact on a metric than jobs with a higher resource consumption [3]. Concerning fairness, we have used a per-user metric called normalized user wait time (NUWT) [11][2]. Then we have measured the avg. of all NUWT values (ANUWT) and their standard deviation (NUWT-dev). The lower the average value and/or the standard deviation are, the more efficient and fair are the results, respectively [11].

The results for these metrics are shown in Table 1 with the best results being highlighted by bold font. Clearly, the complex extension is highly improving, delivering (nearly) best results in all criteria. In fact, the slightly worse NUWT-dev is acceptable as the ANUWT has decreased significantly compared to the historic scenario. Surprisingly, the conservative approach has worse results than both considered setups, which was not anticipated. In fact, all criteria have shown large deterioration compared to historic and complex scenarios. Importantly, the large standard deviation of normalized user wait times (NUWT-dev) suggested that the deteriorating results are likely related to insufficient fairness.

**Table 1.** General results concerning performance and fairness

| | AWWT | AWRT | AWSD | ANUWT | NUWT-dev |
|---|---|---|---|---|---|
| historic | 33795 | 629448 | 2.32 | 0.11 | **0.50** |
| conservative | 56207 | 647769 | 4.37 | 1.07 | 13.52 |
| complex | **18346** | **609909** | **1.66** | **0.08** | 0.56 |

The initial experiment was a surprise, indicating that *conservative extension is not a good solution* due to a significant deterioration in both performance and fairness related metrics. To better understand the situation, we have measured how the two new setups influence the wait times of users in the system. For this purpose, we have measured the percentage of users/jobs having their wait time (WT) improved or deteriorated compared to the original (historic) setup. Also, we have measured the average improvement/deterioration of wait times for these jobs. The overall results are presented in Table 2. For most criteria, the complex setup behaves similar to the conservative. A closer inspection reveals that the actual problem is the huge difference in the avg. wait time for delayed jobs. Complex increases the avg. job wait time by 2.1 hours while conservative

[2] In NUWT, the total user wait time is normalized by the amount of user-consumed CPU time. It uses the same idea as classical *max-min* fairshare [6], i.e., users with high CPU consumption may wait longer than (so far) less active users.

**Table 2.** Detailed results showing impact on users wait times

| | users with impr. WT(%) | jobs with impr. WT(%) | avg. WT impr. (hours) | users with deter. WT(%) | jobs with deter. WT(%) | avg. WT deter. (hours) |
|---|---|---|---|---|---|---|
| conserv. | 26.5 | **13.9** | 6.7 | 19.2 | **2.7** | 55.7 |
| complex | **31.1** | 13.4 | **7.2** | **13.9** | 3.2 | **2.1** |

increases it by 55.7 hours on average! Such a huge increase corresponds with the overall unsatisfactory results seen in Table 1.

Still, further analysis was required to exactly identify the source of the problem. So far, the data indicated that this a fairness-related problem caused by huge wait times of particular jobs. Therefore, we have decided to construct heatmaps showing the avg. wait time of jobs (shown by color intensity) with respect to time ($x$-axis) and queues ($y$-axis) for both considered extensions. Fig. 5 shows the results for conservative (top) and complex (bottom) approaches. Using this "high resolution" tool, we can better understand why the conservative approach performs much worse compared to the complex extension.

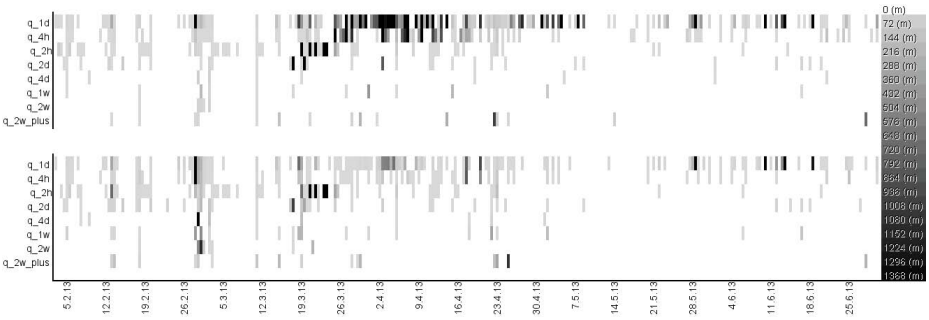


**Fig. 5.** Heatmap of avg. wait time (in minutes) wrt. queues and time for conservative (top) and complex (bottom) extensions

As was mentioned in Section 4, the conservative approach uses fixed ordering of queues which is potentially dangerous as low priority queues may be "blocked out" by higher priority queues, which is unfair with respect to global fairshare. This "blocking effect" is a result of the applied (historic) "greedy" anti-starvation mechanism. Fig. 5 (top) shows such situations on several occasions where the low priority `q_1d` and `q_4h` queues exhibit significant delays compared to higher priority queues. As can be seen, this situation does not appear for complex approach (see Fig. 5 bottom) as (1) all queues are only virtual and all jobs are strictly ordered using fairshare and (2) an optimized anti-starvation mechanism is used.

To sum up, the experiments surprisingly demonstrated that a simple conservative extension of known setup is not a good solution. They revealed previously unexpected results such that it is not sufficient to simply increase the pool of available CPUs for longer jobs, without also improving fairness-related features and the anti-starvation mechanism. For example, it turned out that as soon as longer jobs can use more CPUs it means that also the (original) *anti-starvation mechanism can occupy more CPUs* which blocks all other waiting jobs. Moreover, it was shown that a multi-queue based solution with fixed queue ordering is dangerous as it *ignores global fairshare*. From this point of view, the complex extension increases fairness as now a user with high fairshare-based penalty cannot cheat by sending his or hers jobs into a higher priority queue, such as `q_2w`, or so. Similarly, shorter jobs having high priorities are not unfairly overtaken by longer jobs (from high priority queues). Also, thanks to the new multi-resource aware fairshare mechanism [12] we are now able to properly establish fairness priorities subject to (highly) heterogeneous resources and jobs.

## 6 Conclusion and Future Work

We have shown that an efficient job scheduling is a very complex problem when realistic scenarios are considered. Unlike many prior works that only consider scheduling algorithms, we have provided a detailed insight into the complexity of the problem, using several real-life based examples. Especially, we have stressed out how several particular components of the system interact together and influence the resulting performance. Using a real-life based example, we have shown that detailed simulations can be very useful when looking for a better setup of a given system. The proposed complex extension is currently applied in production use within MetaCentrum's TORQUE resource manager.

Still, this work has some limitations, e.g., several decisions used in this paper are based on an empirical knowledge, an expert assessment or hand-tuned parameters. In the future we would like to develop more rigorous methods that would allow to (semi)automatically identify proper and efficient setups of particular policies. For starters, it would be very helpful to have some method for an efficient dynamic adaptation of various queue limits.

**Acknowledgments.** We highly appreciate the support of the Grant Agency of the Czech Republic under the grant No. P202/12/0306 and the support provided by the programme 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. Adaptive Computing Enterprises, Inc. Moab workload manager administrator's guide, version 7.2.6 (January 2014), http://docs.adaptivecomputing.com
3. Ernemann, C., Hamscher, V., Yahyapour, R.: Benefits of global Grid computing for job scheduling. In: GRID 2004: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pp. 374–379. IEEE (2004)
4. Flis, L., Lason, P., Magrys, M., Ozieblo, A., Twardy, M.: Effective utilization of mixed computing resources on zeus cluster. In: Cracow Grid Workshop, pp. 105–106. ACC Cyfronet AGH (2012)
5. Frachtenberg, E., Feitelson, D.G.: Pitfalls in parallel job scheduling evaluation. In: Feitelson, D., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS, vol. 3834, pp. 257–282. Springer, Heidelberg (2005)
6. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: Fair allocation of multiple resource types. In: 8th USENIX Symposium on Networked Systems Design and Implementation (2011)
7. Jackson, D., Snell, Q., Clement, M.: Core algorithms of the Maui scheduler. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 2001. LNCS, vol. 2221, pp. 87–102. Springer, Heidelberg (2001)
8. Joe-Wong, C., Sen, S., Lan, T., Chiang, M.: Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In: 31st Annual International Conference on Computer Communications (IEEE INFOCOM), pp. 1206–1214 (2012)
9. Kleban, S.D., Clearwater, S.H.: Fair share on high performance computing systems: What does fair really mean? In. In: Third IEEE International Symposium on Cluster Computing and the Grid, pp. 146–153. IEEE Computer Society (2003)
10. Klusáček, D., Rudová, H.: Alea 2 – job scheduling simulator. In: 3rd International ICST Conference on Simulation Tools and Technique, ICST (2010)
11. Klusáček, D., Rudová, H.: Performance and fairness for users in parallel job scheduling. In: Cirne, W., Desai, N., Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2012. LNCS, vol. 7698, pp. 235–252. Springer, Heidelberg (2013)
12. Klusáček, D., Rudová, H.: Multi-resource aware fairsharing for heterogeneous systems. In: Job Scheduling Strategies for Parallel Processing (2014)
13. Lifka, D.A.: The ANL/IBM SP Scheduling System. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1995 and JSSPP 1995. LNCS, vol. 949, pp. 295–303. Springer, Heidelberg (1995)
14. MetaCentrum (January 2014), http://www.metacentrum.cz/
15. Mu'alem, A.W., Feitelson, D.G.: 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. Schwiegelshohn, U.: How to design a job scheduling algorithm. In: Job Scheduling Strategies for Parallel Processing (2014)
19. Wierman, A., Harchol-Balter, M.: Classifying scheduling policies with respect to unfairness in an M/GI/1. In: 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 238–249. ACM (2003)