User-Aware Metrics for Measuring Quality of Parallel Job Schedules

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

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

Abstract. The work presented in this paper is motivated by the challenges in the design of scheduling algorithms for the Czech National Grid MetaCentrum. One of the most notable problems is our inability to efficiently analyze the quality of schedules. While it is still possible to observe and measure certain aspects of generated schedules using various metrics, it is very challenging to choose a set of metrics that would be representative when measuring the schedule quality. Without quality quantification (either relative, or absolute), we have no way to determine the impact of new algorithms and configurations on the schedule quality, prior to their deployment in a production service. The only two options we are left with is to either use expert assessment or to simply deploy new solutions into production and observe their impact on user satisfaction. To approach this problem, we have designed a novel user-aware model and a metric that can overcome the presented issues by evaluating the quality on a user level. The model assigns an expected end time (*EET*) to each job based on a fair partitioning of the system resources, modeling users expectations. Using this calculated *EET* we can then compare generated schedules in detail, while also being able to adequately visualize schedule artifacts, allowing an expert to further analyze them. Moreover, we present how coupling this model with a job scheduling simulator gives us the ability to do an in-depth evaluation of scheduling algorithms before they are deployed into a production environment.

Keywords: Grid, Performance evaluation, Metrics, Queue-based scheduling, Fairness, User-aware scheduling

1 Introduction

The Czech National Grid MetaCentrum is a highly heterogeneous environment currently composed of 26 clusters with heavily varying configurations (GPU clusters, RAM heavy clusters, machines with large SSD disks, clusters with infiniband, high speed NFS access, etc.). MetaCentrum currently contains over 10000 CPU cores, servicing 800 concurrently running jobs on average.

The heterogeneous character of the system is mirrored in its users base. User requirements range from instantaneous access requests, through large job

submissions (thousands of jobs in a single batch), long workflows (large batches of jobs that need to be run in sequence) to extremely unbalanced requests (single CPU core, 1TB of memory). To deal with these various challenges present in this environment we are employing a multitude of software/middleware solutions. These range from our virtualized infrastructure [21] to a locally maintained fork of the Torque [2] batch system coupled with a custom scheduler [25].

Specifically for testing the impact of various changes in the scheduler employed in the MetaCentrum we are using an advanced job scheduling simulator *Alea* [14]. Unfortunately, it is rather hard to correctly evaluate schedules generated by the simulator. This mainly stems from the dichotomy of the simulated workloads. Workloads that can be evaluated using human experts are too simple to be applicable for the production environment and real workloads are to complex to be evaluated in any manner beyond easily understood performance metrics. Similar problem occurs when evaluating job traces recorded in MetaCentrum, as these are usually extremely complex.

We have found that commonly used performance metrics and optimization criteria like average response time, wait time or slowdown [8] do not really reflect what we actually need to optimize. Although these metrics are widely used in the literature, their use is rather questionable. For example, the use of mean values to measure (highly) skewed distributions is a problem in itself [17]. Even though we are capable of complex analysis based on user-agnostic metrics [16, 17], this analysis still does not provide satisfactory results. User-agnostic metrics do not consider whether different users are treated in a fair fashion by the system, which is a very important part of MetaCentrum scheduling algorithm.

In this work we propose a new user-aware metric that allows us to measure the suitability of applied scheduling algorithms. In a natural fashion, it models user expectations concerning system performance using individually computed expected end times (EET). We demonstrate the gain of this solution over evaluation using classical and widely used metrics [8], using both synthetic examples as well as experimental evaluation where real workload and the simulator are used. It demonstrates that our solution provides truly detailed insight into the behavior of the scheduling algorithm, highlighting how suitable, i.e., efficient and fair, are various algorithms with respect to specific users demands.

2 Schedule Evaluation

It must be said that there is no general agreement about measuring schedule quality. This is understandable as the proper measure is inherently heavily environment dependent. The most common approach when determining the quality of a schedule is to use a combination of various metrics. There are several well established groups of metrics. Firstly, we have user-agnostic metrics. These can be either related to a system itself (machine usage, power consumption, etc.) or related to jobs in a system (slowdown, response time, wait time, etc.). Secondly, we have user-aware metrics, mostly represented by some form of fairness-related

metric. We will now closely discuss several categories of metrics that are used to evaluate the suitability of scheduling algorithms.

2.1 User-agnostic Metrics

Typical and widely used metrics are those that do not consider users of the system. Among them, the most popular are the *average response time, average wait time* or the *average slowdown*. Average response time measures the mean time from job submission to its termination. The average wait time is the mean time that jobs spend waiting before their execution starts. The slowdown is the ratio of an actual response time of a job to a response time if executed without any waiting. While the response time only focuses on a time when a job terminates, the slowdown measures the responsiveness of the system with respect to a job length, i.e., jobs are completed within the time proportional to jobs demands [8]. Wait time supplies the slowdown and the response time. Short wait times prevent the users from feeling that the scheduler "ignores" their jobs.

Although widely used, these job-related metrics are based on several assumptions that no longer hold in heterogeneous, grid-like environments, as we explain in the following text.

Problems with Job Priority One of the main assumptions of standard job-related metrics is that a shorter job should receive higher priority in the system (see, e.g., the slowdown or the response time). Shorter jobs are easier to schedule and users with more complex (longer) requests are therefore required to expect longer wait times.

This assumption is problematic on several levels. Firstly, as long as we are measuring the total job penalty or the average value (e.g., total/average slowdown/response time) this "shortest job first" priority advantage will remain absolute. This can very easily lead to huge starvation of (few) long jobs[1]. The grid is indeed a dynamic system and the number of jobs submitted by a single user is, to a certain degree, proportional to the number of jobs successfully processed. Given the total job length dispersion (from several minutes to a month) [5, 19], users with extremely long jobs would hardly ever get their requests satisfied.

Secondly, the correlation between the absolute job length and the job urgency is little to none. Again, due to the large dispersion of job lengths, the notion of a "short job" has very different meaning to different users. The increased benevolence toward wait times for long jobs is simply due to the increased absolute users runtime estimation error (10% imprecision on a month long job equates to 3 days).

Problems with Resource Requirements Similar issues occur when dealing with different job resource requirements. If one can split a large CPU demanding

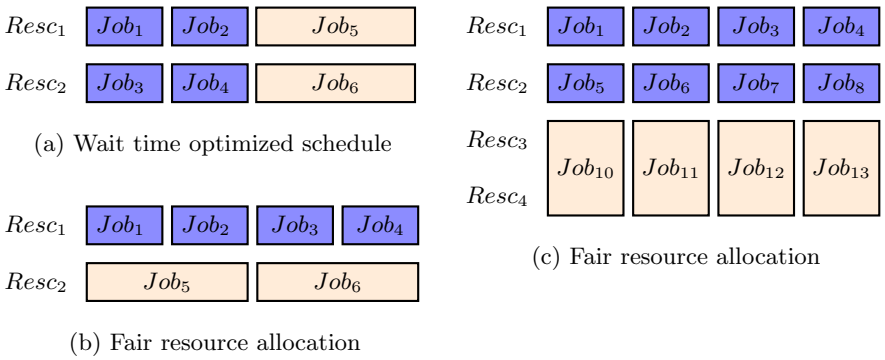
[1] Production systems (including MetaCentrum) usually employ a certain type of anti-starvation technique. Since this approach goes directly against the order suggested by the job-related metric, it naturally leads to skewed results.

job into a set of smaller jobs, these will obtain higher priority. This problem was previously addressed by normalizing the selected metric using the number of CPU cores a job is requesting [7] [13], as a weight.

Unfortunately, nearly no metric is designed to reflect combined consumption of multiple resources [15] such as CPUs, RAM, GPUs, HDD, etc. When multiple resources are concerned, further measures need to be employed, like dominant resource [10] or processor equivalent [12] to properly reflect other than CPU-related job requirements.

Why Users Matter Last but not least, we now demonstrate the major problem which causes that user-agnostic metrics are impractical in real systems.

Let us consider an example of a schedule optimized according to average wait time (see Fig. 1a). In this schedule we have two users (bright-orange and dark-blue) and the optimization criterion favors the jobs of the blue user due to their shorter length[2]. The total penalty for this schedule according to average wait time would be $\frac{0+0+1+1+2+2}{6} = 1$. Unfortunately, the orange user will clearly not consider this schedule optimal. He or she is requesting the same amount of resources as the blue user, but has to wait until all jobs of the blue user are processed.



(a) Wait time optimized schedule

(b) Fair resource allocation

(c) Fair resource allocation

Fig. 1. Examples of optimal schedules

Let us consider a different schedule, this time using fair resource allocation among the blue and the orange user (see Fig. 1b). In this case both users receive one resource exclusively for their jobs and both users receive the complete results of their jobs at the same time. The total penalty for this schedule according to average wait time would be $\frac{0+1+2+3+0+2}{6} = \frac{4}{3}$, which is more than in the previous example. Indeed, we would get similar results for both response time and slowdown.

[2] $Resc_1$ and $Resc_2$ represent resources, e.g., CPU cores.

An analogous problem occurs when we simulate similar situation where, instead of job runtimes, overall resource requirements are considered (see Fig. 1c). Again the presented fair resource allocation among the blue and the orange user is not considered optimal according to user-agnostic metrics.

2.2 User-Aware Metrics

User-aware metrics aim at maximizing "benefits" regarding the users of the system. Very often, the metric applied for such goal is related to fairness. In production environments, some type of fairness guaranteeing process/metric is usually provided. These measures are highly dependent on the system itself and range from simple measures that try to maintain the order in which the requests entered the system [22] to much more complicated measures concerned with the combined consumption of various resources [10].

Job-to-job Fairness Fairness is often understood and represented as a *job-related* metric, meaning that every job should be served in a fair fashion with respect to other jobs [22, 18, 23]. For example, a *fair start time* (FST) metric [22, 18] measures the influence of later arriving jobs on the execution start time of currently waiting jobs. FST is calculated for each job, by creating a schedule assuming no later jobs arrive. The resulting "unfairness" is the difference between FST and the actual start time. Similar metric is so called *fair slowdown* [23]. The fair slowdown is computed using FST and can be used to quantify the fairness of a scheduler by looking at the percentage of jobs that have a higher slowdown than is their fair slowdown [23]. Sadly, these job-to-job metrics do not guarantee fair behavior with respect to different users of the system.

User-to-user Fairness Instead of the job-to-job fairness, the resource management systems frequently prefer to guarantee fair performance to *different users*. One of the commonly employed techniques is *fairshare*. Fairshare-based fairness is supported in many production resource management systems such as in PBS [20], TORQUE [2], Moab, Maui [1], Quincy [11] or in Hadoop Fair and Capacity Schedulers [4, 3]. Fairshare tries to balance the mid-to-long term resource usage among *users* of the system[3]. More precisely, if a user A and a user B have identical priorities, they will receive the same amount of resources, when averaged over a reasonably long time period [12]. This is of course only true when both user A and user B actually request these resources.

Fairshare can also be expressed using the following equation:

$$\forall u; \lim_{time \to \infty} Usage(u) = AvailableResources \cdot time \cdot DesignatedFraction(u)$$

Simply put, if we take a large enough time period, the total used resources by one user ($Usage(u)$) will equate to the amount of resources designated for

[3] Depending on the implementation, fairshare can also prevent usage spikes.

this user. Or formulated in a different manner, the amount of used resources will converge toward the amount designated by a user priority.

While the methods applied in production fairshare algorithms are well documented [12], there is — surprisingly — no common agreement about how to actually measure, i.e., evaluate, analyze or even compare, the level of (un)fairness for such user-to-user approaches. Authors that need to employ such methods usually rely on measuring the variability (using, e.g., the standard deviation) of user-agnostic metrics [26].

Therefore, in the following text we propose a novel user-aware model and a metric that can overcome previous issues by evaluating the schedule quality on a user level. In a natural fashion, the model represents user expectations concerning system performance, using individually computed EETs and allows us to compare quality of different schedules in a reasonably detailed manner.

3 Proposed User-Aware Model

As we discussed in the previous section, commonly used metrics may provide misleading results. Moreover, production-based anti-starvation and fairshare mechanisms can further delay executions of certain jobs, thus confusing those metrics even more. Therefore, we can no longer rely on these metrics when evaluating the quality of generated schedules.

This situation leads us to the formulation of a new user-aware model that can be used instead. The model is designed with several important features in mind: the model needs to be simple enough, so that its output remains easy to analyze; the model needs to provide information-rich output; the model needs to be robust enough to ignore irrelevant differences, e.g., in particular job resource specifications (see next Section 3.1).

3.1 The Model of User Expectations

Our model tries to capture the expectations of users concerning the target system and then it evaluates the quality of a given schedule by determining how well were these expectations satisfied.

User Expectations on Job Wait Times As we have observed during the past years in MetaCentrum, there are some common expectations of the users toward the (expected) behavior of the scheduling system. User expectations toward job wait time mainly correlate with the complexity of the job requirements and the number of jobs present in the batch. The more resources a job requires, the more tolerant a user tends to be toward its wait time. Similar correlation exists when considering a set of jobs as a group, the expected response time (of the whole group) correlates with the total amount of required resources. On the other hand, users exhibit very little tolerance in situations when their jobs are waiting, while jobs of other users with similar requirements are starting regularly.

In order to follow these expectations, we have decided to model each user separately, independent of other users in the system. Our model is also built to disregard insignificant differences, e.g., in the specification of resource requirements. For example, if 4 simultaneously submitted jobs require 1 CPU and 4 GB RAM each, then we consider this to be equal to 1 job that requires 4 CPUs and 16 GB of RAM.

User Expectations on Available Capacity According to our experience, user expectations toward the capacity the system can provide for a single user tend to be rather reasonable. Users understand that they cannot allocate the entire system for themselves. Therefore, we are matching the capacity expectation by giving each user a virtual share of the system, following the idea of the well known fairshare principle [12].

As we have already mentioned, our model has been designed in order to match user expectations. Similarly, the proposed metric is designed to determine whether these expectations were fulfilled in the provided schedule. Let us now closely describe the proposed expected end time (EET) metric.

3.2 Proposed *EET*-based Metric

The proposed EET-based metric works in a simple but robust fashion. Simply put, specific EET expressing user expectations is calculated for every arriving job and then it is checked against the provided schedule[4] analyzing which of the calculated EETs are satisfied and which are violated.

Simply put, EET for a particular job is calculated by assigning the job to a given resource of predefined capacity. This capacity represents the share of real resources a user is expecting to receive from the system (at any time). We can assign different capacity of this resource to different users, thus modeling more or less demanding users (users with different priorities). User's jobs are then inserted into the available capacity in a "tetris-like" fashion. It means that we need to rearrange jobs that do not fit into the remaining capacity of this resource, essentially treating them as moldable/malleable [9]. Just like moldable jobs, a runtime of a job is proportionally increased if the amount of available resources is smaller than requested by that job. On the other hand, unlike moldable jobs, a job cannot allocate more resources than it requires, i.e., its runtime cannot be decreased by using more resources. This approach has been adopted based on our experiences concerning users expectations, as were discussed in Section 3.1. However, such transformations are only done for the purposes of the EET calculation and the job scheduling algorithms we are considering in this paper only work with non-moldable jobs that have constant resource usage.

Algorithm for *EET* Calculation The computation of EET is described in Algorithm 1 and works as follows. Based on the expectations we mentioned in

[4] By default, we assume that this provided schedule is a historic schedule as found in a workload trace. If needed, it can be extended for a use within "live" scheduler.

Section 3.1, each user is assigned a resource with a predefined capacity, that he or she expects to have access to (Line 2). Each job is then allowed to consume this resource starting from its arrival time. Available share is being allocated in blocks, where 1 block has duration ("length") of 1 time unit and size ("height") equal to 1 resource unit. In this paper, 1 block equals to 1 second $\times$ 1 CPU. Of course, a job may require more resources per time unit than is available. In such situation, a job is rearranged to fit within available blocks in that share. Simply put, such job is then "longer" and "narrower". On the other hand, if the available share is larger than the resources requested per time unit ($RescReq_j$), we do not rearrange such job. Simply put, jobs that are "long" and "narrow" cannot become "shorter" and "higher", since they are still limited by the amount of resources requested ($RescReq_j$) in each time unit.

Algorithm 1 Calculation of EET for a given user

```
1: for time = 0 to SheduleEnd do
2:     capacity_time = ExpectedCapacity
3: end for
4: sort_arrival(jobs)
5: for j = 0 to JobCount do
6:     jobCapacity = rescReq_j · runtime_j
7:     time = arrival_j
8:     while jobCapacity > 0 do
9:         consumedCapacity = min(capacity_time, rescReq_j)
10:        capacity_time = capacity_time − consumedCapacity
11:        jobCapacity = jobCapacity − consumedCapacity
12:        time = time + 1
13:    end while
14:    EET_j = time
15: end for
```

The presented Algorithm 1 will calculate the EET for a single resource type (e.g., CPU cores requested), but extending this algorithm for multiple resources (e.g., Memory, GPU cards, ...) is relatively straightforward. After calculating an EET for each resource separately, we then select the latest/maximum EET, thus the final EET is dictated by the scarcest and/or most utilized resource.

Processing one job after another in order of arrival (Line 4), each job is represented by the product ($jobCapacity$) of its resource requirements ($rescReq_j$) over time ($runtime_j$). In a loop, we determine the capacity that will be consumed ($consumedCapacity$) in the user share at a given $time$ by the job j (Line 9). The capacity of user share could be already (partially) consumed by previously added jobs, therefore the $consumedCapacity$ at given $time$ is either the remaining share capacity at that $time$ ($capacity_{time}$) or the $rescReq_j$ (when a job requests less resources than is the current $capacity_{time}$). Next, we allocate this $consumedCapacity$ and update both the remaining $capacity_{time}$ (Line 10) and the remaining $jobCapacity$, i.e., the "job remainder" that remains unallocated

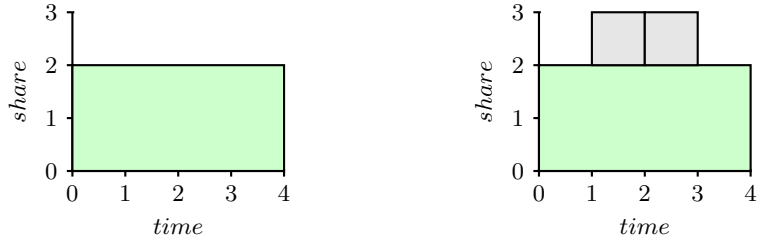
(Line 11). Finally we increase the *time* (Line 12) and proceed to next iteration. Once the whole job is allocated ($jobCapacity \leq 0$), the inner while loop ends and the job EET_j is equal to the *time*.

As designed, the *EET*-based Metric is used in a "post mortem" fashion, meaning that it is used offline using historic workload trace as an input. Therefore, an exact $runtime_j$ is known for every job. If properly modified, it can also be used online, e.g., inside a job scheduling heuristic. So far, we have not use it in such a way, and we only use it for offline analysis of various scheduling algorithms.

Example of *EET* Calculation For simplicity, let us consider a user having two jobs with parameters specified in Table 1. We have decided to grant this user share $ExpectedCapacity = 3$. The first job of this user only requires two units. Since no units were consumed yet and $ExpectedCapacity > rescReq$ the *EET* will trivially correspond to job runtime. The second job will arrive at $time = 1$. There is only one unit of share remaining from $time = 0$ to $time = 4$ and the job requires two units of share. The job will therefore consume one unit of share at time $time = 1$ and the second unit of share at $time = 2$, ending up with $EET = 3$. This process is illustrated in Fig. 2.

Table 1. Calculated *EET* for a user with two jobs and $ExpectedUserCapacity = 3$.

	arrival	*rescReq*	*runtime*	*EET*
Job_1	0	2	4	4
Job_2	1	2	1	3



(a) *EET* calculation for Job_1 (b) *EET* calculation for Job_2

Fig. 2. Visualization of *EET* calculation

For two such users, presented in the example, we can construct an optimal schedule with no broken *EET*s using a system with $capacity = 6$ (see Fig. 3). This example also demonstrates the fundamental difference between performance

metrics and our model. Using performance metrics, Job_4 would receive penalty according to most metrics, since it was delayed by one time unit. Our model makes such distinction irrelevant by only considering the violated EETs.

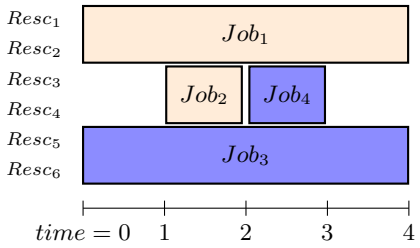


Fig. 3. Optimal schedule (all EETs satisfied) for two identical users that both have those two jobs from Table 1.

3.3 Application of the Proposed Model

Critical part of the model is of course our ability to extract important information out of it. This is directly related to the robustness and simplicity of the model, which allows us to draw direct links between the generated schedule and the output of the model.

Violated *EET* and Tardiness First layer of information output is related directly to the violated EETs. We can measure how many EETs were violated, as well as to what extent. For a crude comparison, we measure the number of violated EETs ($violatedEET_u$) for each user u, and transform it to percentage of jobs that did not meet their EETs ($VEET_u$) as shown in Formula 1.

$$VEET_u = \frac{violatedEET_u}{jobCount_u} \cdot 100\% \tag{1}$$

Using a statistical box-plot, we can then examine the distribution of this value across all users and visually compare the results across multiple algorithms. An example of such a comparison is presented in Fig. 6a.

If we want to examine how significantly were those EETs violated we use a simple metric based on tardiness. First, we compute job tardiness using Formula 2, where C_j is the actual completion time of a job and EET_j is the expected end time of job j. Next, we compute the total weighted tardiness for a given user (WT_u) using Formula 3. Here, J_u is the set of jobs belonging to that user u and w_j is the number of CPUs used by a job j, representing the "weight" of that job.

$$T_j = \max(0,\ C_j - EET_j) \tag{2}$$

$$WT_u = \sum_{j \in J_u} (w_j \cdot T_j) \tag{3}$$

Again, we then plot the distribution of this value across all users and visually compare the results across multiple algorithms. An example of such a comparison is presented in Fig. 6b. The interpretation however does still require further analysis due to the nature of value distributions themselves[5].

***EET* Heatmaps** For a more in-depth analysis we can analyze how *EET*s are violated throughout the schedule. This is done by plotting the number of violated *EET*s during the time using a *heatmap* [17]. In a heatmap, we use x-axis for time (samples are taken each minute) and the y-axis for users of the system. A color at given coordinates then corresponds to the number of violated *EET*s at that point in time and given user as can be seen in Fig. 7.

Such a heatmap is represented as a 2D array of integers. It has m rows where m is the number of users of the system. Each row has a length of t, where t is the length of the schedule in time units. Initially, each cell is equal to 0 (no violated *EET*s). Next, the structure is updated according to the actual schedule and calculated *EET*s. To illustrate the process, let us consider a situation when a given user job is executed so that its *EET* (calculated using Algorithm 1) is violated by 2 minutes ($T_j = 2$). Since the resolution (sampling rate) of the heatmap is 1 minute, and the *EET* has been violated by 2 minutes, two consecutive horizontal cells in the data structure will be incremented by 1. For example, $[x][y]$ and $[x+1][y]$ will be both incremented by 1. Here, y is the index of given user. Clearly, such a heatmap captures both the number of violated *EET*s at given time, as well as the tardiness ($T_j = 2$ implies that two consecutive cells are incremented). The question is how to determine the x, i.e., what should be used as the "start time" of that violated *EET*. There are, of course, several possible approaches for determining this time. In our case, we set the "start time" x of the violated *EET* as: $x = EET_j - runtime_j$, i.e., x is the latest possible job start time that enables us to meet the job *EET*. An illustration of the aforementioned example is shown in Fig. 4. When the violated *EET*s are plotted in this way, they are usually aligned with the areas of high system usage, which are frequently the root cause of such a violation.

When such a heatmap is used in conjunction with the actual schedule it allows us to better understand "what is happening" in the system and why. We will provide such an example in the following section.

[5] What is better, a more disperse distribution with a better median, or a less disperse distribution?

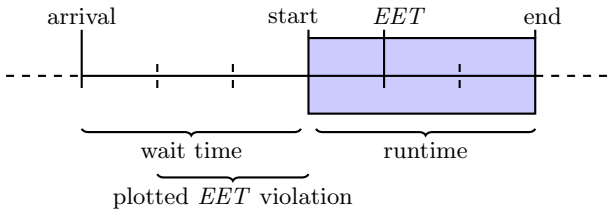


Fig. 4. An example of a violated *EET* plotting.

4 Experiments

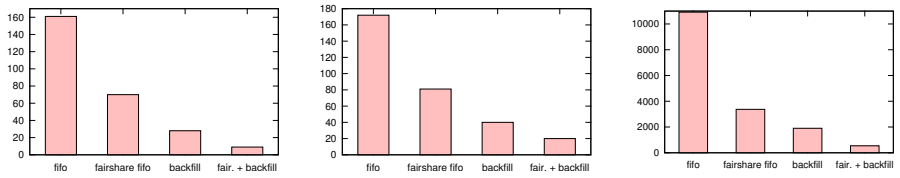
In the experimental part of this work, we demonstrate how the proposed model can be used to deeply evaluate the suitability of several scheduling algorithms with respect to a given computing system.

4.1 Experimental Setup

For the purpose of evaluation, the proposed model and the *EET*-based metric has been implemented within the *Alea* job scheduling simulator [14]. To simulate the system workload, we have used a real historical workload trace coming from MetaCentrum which covered 5 months of job execution on the *zewura* cluster. Zewura contains 7 shared memory computing nodes, each consisting of 80 CPUs and 504 GB of RAM. The log contains 17,250 jobs belonging to 70 distinct users. These jobs are either sequential (1 CPU required only) or (highly) parallel. Also, job runtimes as well as memory requirements are highly heterogeneous with some jobs running only few minutes while others running up to 30 days. Concerning RAM, jobs are requesting anything between few MBs of RAM up to 504 GBs of RAM. This workload log is freely available at: `http://www.fi.muni.cz/~xklusac/workload`.

In order to demonstrate how the proposed model and metric can be used to evaluate the behavior of scheduling algorithms, we have applied two widely used scheduling algorithms — the trivial *First In First Out (FIFO)* and the well known *backfilling* [19]. FIFO always schedules the first job in the queue, checking the availability of the resources required by such job. If all the resources required by the first job in the queue are available, it is immediately scheduled for execution, otherwise FIFO waits until all required resources become available. While the first job is waiting for execution none of the remaining jobs can be scheduled, even if required resources are available. In practice, such approach often implies a low utilization of the systems resources, that cannot be used by some "less demanding" job(s) from the queue [22, 19]. To solve this problem algorithms based on *backfilling* are frequently used [19].

Backfilling is an optimization of the FIFO algorithm that tries to maximize resource utilization [19]. It works as FIFO but when the first job in the queue cannot be scheduled immediately, backfilling calculates the earliest possible starting



(a) Average wait time (hours) (b) Average response time (hours) (c) Average slowdown

Fig. 5. Experimental results when measured by classical user-agnostic metrics.

time for the first job using the processing time estimates of running jobs. Then, it makes a reservation to run the job at this pre-computed time. Next, it scans the queue of waiting jobs and schedules immediately every job not interfering with the reservation [19]. This increases resource utilization, as idle resources are utilized by suitable jobs, while decreasing the mean job wait time.

In our experiments, both FIFO and backfilling were applied either with or without fairshare-based priorities enabled. If fairshare-based priorities are enabled, it means that the jobs waiting in the queue are periodically reordered according to actual users' fairshare priorities. Fairshare priorities are updated continuously as jobs are being processed. In other words, those jobs belonging to less active users obtain higher priority and vice versa as was discussed in Section 2.2. Further details can be found in, e.g., [15, 12, 1].

As a result of such a simulation setup, we have obtained four different job schedules, which are marked according to the applied scheduling algorithm as: *fifo, fairshare fifo, backfill* and *fair. + backfill*.

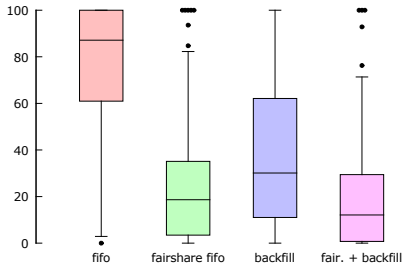
4.2 Experimental Results

In the Fig. 5 we show a simple comparison of applied scheduling algorithms using the popular user-agnostic metrics (wait time, response time and slowdown) that have been described in Section 2.1.

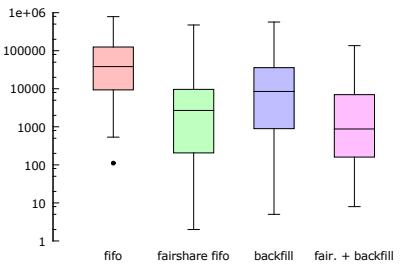
As we can see, the results suggests that backfilling algorithm (both with fairshare enabled or disabled) is much better than both FIFO versions. This is not surprising and sounds with results from the literature [19] (see FIFO and backfilling comparison in Section 4.1). What remains unclear is whether the application of fairshare priorities results in better, i.e., more fair performance for the users of the system. As we have explained in Section 3.1, good fairness and a high performance are the two most important factors that make the system users satisfied — at least this is the case in MetaCentrum.

Therefore, let us now demonstrate how the EET-based metric that emulates user expectations can be used for much more detailed analysis. For a very crude comparison of these algorithms, we can use the percentage of violated EETs ($VEET_u$). This information is calculated for each user by Formula 1 and plot-

ted using a box-plot[6] as shown in Fig. 6a. In our case, the y-axis denotes the percentage of violated EETs and the boxes show how the users are distributed according to the percentage of violated EETs. This gives us immediate information about the quality of the algorithms with respect to both performance (small percentage of violated EETs indicates good performance) and fairness (bigger deviation equates less fairness). As can be seen, only algorithms with enabled fairshare-based priorities (fairshare fifo, fair. + backfill) are able to provide at least some fairness to the users (75% of users have at most 35% violated EETs). On the other hand, no algorithm was able to eliminate users with 100% of violated EETs.



(a) Percentage of users (boxes) having at most y% of violated EETs (y-axis).



(b) Total weighted tardiness in seconds (y-axis, log. scale) according to users (boxes).

Fig. 6. Simple comparison of different scheduling algorithms

For a more quantitative comparison, we use the total weighted user tardiness (WT_u) as defined by Formula 3. Again, we calculate this metric on a per-user basis and then plot these values as a box-plot as shown in Fig. 6b. Here, y-axis represents the calculated total weighted user tardiness (seconds in log. scale). This figure provides further details to the results observed in Fig. 6a, by showing that fair. + backfill significantly decreases those job delays observed by users.

For an in-depth analysis we plot a time overview of violated EETs using so called heatmap [17]. For each user (y-axis) and a point in time (x-axis) we plot a color corresponding to the number of violated EETs at that coordinates. Moreover, we also plot actual CPU and RAM usage bellow each heatmap, to put the results in context with the resource utilization. Such graphs are available in Fig. 7. It gives us the ability to evaluate the behavior of the algorithms with respect to each user of the system and analyze the specific artifacts that may appear in the schedule. For example, for those algorithms using fairshare there are clear artifacts created by the fairshare algorithm, appearing as long lines of violated EETs (second and fourth row in Fig. 7). This is an expected behavior for

[6] Box-plot maintains information on the distribution of $VEET_u$ values by showing their minimum, lower quartile, median, upper quartile and the maximum, plus possible extreme outliers marked as dots.

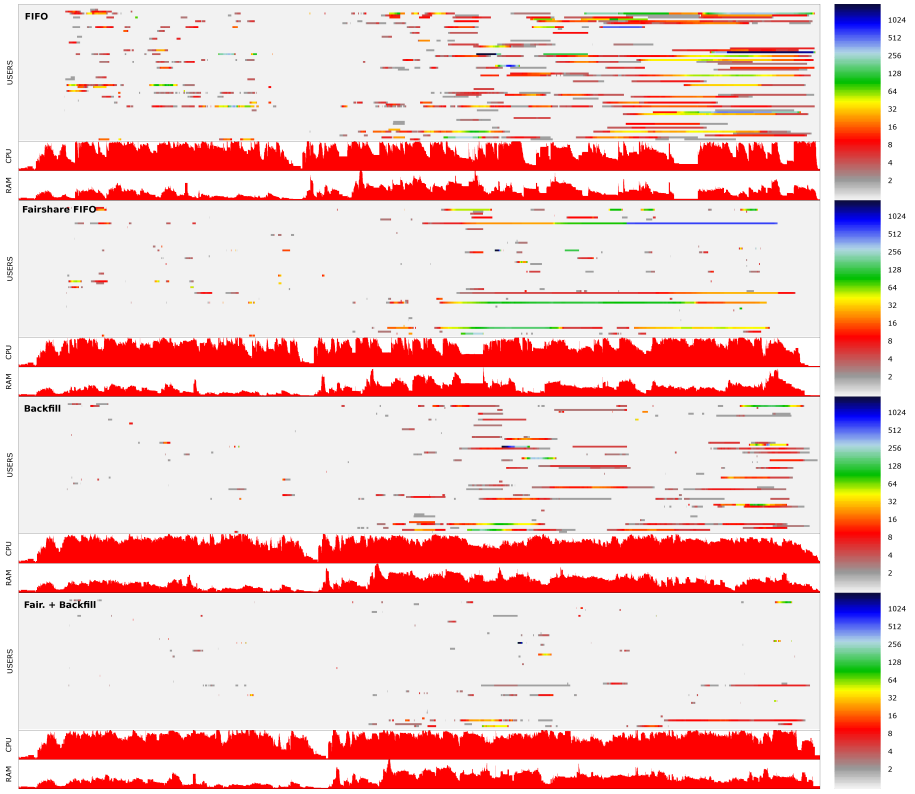


Fig. 7. Heatmaps showing number of violated *EET*s comparing *fifo* and *fairshare fifo* (first and second row) against *backfill* and *fair. + backfill* (third and fourth row).

workload-heavy users, as their priority is decreased proportionally to the amount of resources they consume. Concerning system usage, we can see that FIFO algorithms (first and second row in Fig. 7) performs worse than backfilling. Plain FIFO not only suffers from a large number of violated *EET*s (which indicates poorer usage) but we can also clearly see the gaps in the system usage (bottom part of the top graph). The overall bad impact on *EET*s is a natural effect of poor resource utilization. From this point of view, backfilling should perform very well. However, as we can see in case of plain backfilling, it increases the system utilization at the expense of some users. In this context, plain backfilling performs worse than FIFO with fairshare, which sounds with the results from Fig. 6. Only the backfilling algorithm with fairshare-based priorities (bottom row) performs better, decreasing the number of violated *EET*s and increasing the system utilization. This direct "high resolution" comparison provides clear evidence how fairshare-based priorities and backfilling improve the performance and fairness of the system with respect to its users. It is worth noticing that these

issues and characteristics cannot be captured by classical user-agnostic metrics as we have discussed in Section 2.1 and demonstrated in Fig. 5. For example, according to user-agnostic metrics, plain backfilling is much better than FIFO with fairshare, while our results show the exact opposite.

4.3 Summary

In this section we have demonstrated how the proposed model and the metric can be used to better understand the behavior of various scheduling algorithms. Using a real workload and a simulator, we have shown how different scheduling approaches can be tested and their results compared in a "high resolution" manner. Using our approach, we now better understand how various scheduling setups influence the impact on system users. It is worth noticing that the widely used user-agnostic performance metric are neither suitable for real-life multi-user oriented systems nor are able to provide such a detailed insight.

The tool used here for workload processing and visualization [24] is available at: `http://happycerberus.github.io/Deadline-Schedule-Evaluator` under the MIT license. This tool is capable of processing standard accounting format-based workload data, it also supports SWF-like workload traces from workload archives, or it can even use a trace coming from a job scheduling simulator. This way we can either analyze the current performance of our systems, or do a fast, iteration-like development and analysis of new scheduling algorithms using a simulator. Beside the general, system-wide views presented in Section 4.2, the tool can also provide more detailed views. For example, we can filter out specific users (e.g., system administrators) and queues (high/low priority), or even generate a heatmap for single queue or user.

5 Conclusion and Future Work

In this paper we have presented a novel model for measuring the quality of schedules, based on user expectations. The main purpose of this model is to provide an alternative to popular job metrics that capture certain aspects of the schedule but do not properly represent the suitability of the schedule as a whole. Our experiments on real workload from the Czech National Grid MetaCentrum have shown that this model does indeed provide expected level of detail and explains clearly the pros and cons of selected scheduling algorithms. We have shown how the model-based results can be represented both in a simple comparison (Fig. 6) and in an in-depth heatmap representation of the generated schedule (Fig. 7).

In the future, we aim to refine current simple *EET* model. For example, the *EET* calculation should reflect previous user activity. Clearly, if a user utilized a lot of resources in the (near) past, the *EET* shall be higher (i.e., less strict) as such a user is typically having smaller priority compared to less active users. This is a normal situation in systems using fairshare approach, thus we should not penalize it by our metric. Second, many users are executing workflows or

so called "bag of tasks" [6] applications and they are interested in the performance of an entire batch of jobs, rather than focusing on the performance of a single job/task within their workload. For such a workflow or bags of tasks we should therefore generate single *EET* rather than compute separate *EET* for each job/task. This will require more complex data sets as our current workload traces do not make any distinction among normal jobs and workflows and/or bag of tasks-like applications.

Acknowledgments. We highly appreciate the support of the Grant Agency of the Czech Republic under the grant No. P202/12/0306. The support provided under 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. Adaptive Computing Enterprises, Inc. *TORQUE Admininstrator Guide, version 4.2.6*, January 2014. `http://docs.adaptivecomputing.com`.
3. Apache.org. *Hadoop Capacity Scheduler*, January 2014. `http://hadoop.apache.org/docs/r1.1.1/capacity_scheduler.html`.
4. Apache.org. *Hadoop Fair Scheduler*, January 2014. `http://hadoop.apache.org/docs/r1.1.1/fair_scheduler.html`.
5. W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *2001 IEEE International Workshop on Workload Characterization (WWC 2001)*, pages 140–148. IEEE Computer Society, 2001.
6. W. Cirne, F. Brasileiro, J. Sauv, N. Andrade, D. Paranhos, E. Santos-neto, R. Medeiros, and F. C. Gr. Grid computing for bag of tasks applications. In *3rd. IFIP Conference on E-Commerce, E-Business and EGovernment*, 2003.
7. C. Ernemann, V. Hamscher, and R. Yahyapour. Benefits of global Grid computing for job scheduling. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 374–379. IEEE, 2004.
8. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *LNCS*, pages 1–34. Springer, 1997.
9. E. Frachtenberg and D. G. Feitelson. Pitfalls in parallel job scheduling evaluation. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3834 of *LNCS*, pages 257–282. Springer Verlag, 2005.
10. 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.
11. M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *SOSP'09*, 2009.

12. D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *LNCS*, pages 87–102. Springer Verlag, 2001.
13. H. D. Karatza. Performance of gang scheduling strategies in a parallel system. *Simulation Modelling Practice and Theory*, 17(2):430 – 441, 2009.
14. D. Klusáček and H. Rudová. Alea 2 – job scheduling simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. ICST, 2010.
15. D. Klusáček, H. Rudová, and M. Jaroš. Multi resource fairness: Problems and challenges. In *Job Scheduling Strategies for Parallel Processing (JSSPP'13)*, 2013.
16. D. Krakov and D. Feitelson. High-resolution analysis of parallel job workloads. In W. Cirne, N. Desai, E. Frachtenberg, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 7698 of *Lecture Notes in Computer Science*, pages 178–195. Springer Berlin Heidelberg, 2013.
17. D. Krakov and D. G. Feitelson. Comparing performance heatmaps. In *Job Scheduling Strategies for Parallel Processing*, 2013.
18. V. J. Leung, G. Sabin, and P. Sadayappan. Parallel job scheduling policies to improve fairness: a case study. Technical Report SAND2008-1310, Sandia National Laboratories, 2008.
19. 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.
20. PBS Works. *PBS Professional 12.1, Administrator's Guide*, January 2014. `http://www.pbsworks.com/documentation/support/`.
21. M. Ruda, Z. Šustr, J. Sitera, D. Antoš, L. Hejtmánek, P. Holub, and M. Mulač. Virtual Clusters as a New Service of MetaCentrum, the Czech NGI. In *Cracow'09 Grid Workshop*, 2010.
22. G. Sabin, G. Kochhar, and P. Sadayappan. Job fairness in non-preemptive job scheduling. In *International Conference on Parallel Processing (ICPP'04)*, pages 186–194. IEEE Computer Society, 2004.
23. S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Selective reservation strategies for backfill job scheduling. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *LNCS*, pages 55–71. Springer Verlag, 2002.
24. Š. Tóth and D. Klusáček. Tools and Methods for detailed Analysis of Complex Job Schedules in the Czech National Grid. In M. Bubak, M. Turała, and K. Wiatr, editors, *Cracow Grid Workshop*, pages 83–84. ACC CYFRONET AGH, 2013.
25. Š. Tóth and M. Ruda. Practical Experiences with Torque Meta-Scheduling in The Czech National Grid. *Computer Science*, 13 (2):33–45, 2012.
26. S. Vasupongayya and S.-H. Chiang. On job fairness in non-preemptive parallel job scheduling. In S. Q. Zheng, editor, *International Conference on Parallel and Distributed Computing Systems (PDCS 2005)*, pages 100–105. IASTED/ACTA Press, 2005.