A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times

Andrew Burkimsher, Iain Bate, Leandro Soares Indrusiak

Department of Computer Science, University of York, York, YO10 5GH, UK

Abstract

This paper considers the dynamic scheduling of parallel, dependent tasks onto a static, distributed computing platform, with the intention of delivering fairness and quality of service (QoS) to users. They key QoS requirement is that responsiveness is maintained for workloads with a wide range of execution times (minutes to months) even under transient periods of overload. A survey of schedule QoS metrics is presented, classified into those dealing with responsiveness, fairness and utilisation. These metrics are evaluated as to their ability to detect undesirable features of schedules. The Schedule Length Ratio (SLR) metric is shown to be the most helpful for measuring responsiveness in the presence of dependencies. A novel list scheduling policy called Projected-SLR is presented that delivers good responsiveness and fairness by using the SLR metric in its scheduling decisions. Projected-SLR is found to perform equally as well in responsiveness, fairness and utilisation as the best of the other scheduling policies evaluated (Shortest Remaining Time First/SRTF), using synthetic workloads and an industrial trace. However, Projected-SLR does this with a guarantee of starvation-free behaviour, unlike SRTF.

Keywords: , Scheduling, Metrics, Responsiveness, Fairness, Utilisation, List Scheduling, Dependencies, Network Delays, Heterogeneity

1. Introduction

High-Performance Computing systems (*HPCs*) made up of a large number of parallel processors have become ever more popular in recent years, due to their ability to provide large quantities of computing capacity at a relatively low cost. Where a single HPC cluster cannot satisfy an organisation's desire for computing power, geographically-distributed networks of such clusters have been created, and these are known as *grids* [2]. HPC platforms built on a grid architecture have now been created to sell their computing capacity in real-time, and these are now known as *clouds* [10].

These parallel architectures are ideally suited to workloads where work can be divided into independent pieces, and distributed accordingly. However, many grid workloads can only be parallelised to an extent, as they contain dependencies that necessitate some serial execution. The pieces these workloads are broken into may also be of significantly different sizes. While producing a schedule using a policy such as First In First Out may be trivial, scheduling over a grid to meet quality of service requirements or a level of optimality in the presence of dependencies can be difficult [18].

Optimal scheduling in the general case is an NP-Complete problem [11]. Therefore, optimal scheduling is intractable at the scale of grid systems, where heterogeneity [16] and network delays [7] are also present. Many heuristic scheduling policies have been proposed (see [4, 14, 16] for surveys), but these must be evaluated in order to identify their respective strengths for particular platforms and workloads.

Email addresses: amb502@york.ac.uk (Andrew Burkimsher), iain.bate@york.ac.uk (Iain Bate), leandro.indrusiak@york.ac.uk (Leandro Soares Indrusiak)

Organisations that run their own HPC capacity as well as cloud providers have an interest in providing Quality of Service (QoS) to their users. The productivity of an organisation may be impacted if its users are having to wait too long for their work to be returned from the HPC. Poor QoS for cloud providers may lead to its users changing provider and them losing business. Low levels of QoS can be caused by poor scheduling decisions, leading to low throughput or unacceptably long task waiting times. Metrics are essential for providers to be able to monitor the QoS they are delivering and these metrics should be appropriate to the needs of their users.

There is an inherent challenge in maintaining QoS because the kinds of workload run on HPCs, grids and clouds tend to have significant variations in demand. This variation is one of the reasons that customers choose to use the resources of a cloud provider in the first place. While cloud providers can attempt to manage demand through the adjustment of spot pricing, there is still a significant likelihood that there will be times when the rate of work arrival is greater than the maximum possible rate of processing. This is because it is likely to be uneconomic for a cloud provider to maintain significant idle capacity just for the servicing of such peaks. Therefore, providers need metrics to know whether a peak in load on constrained resources will actually lead to an inadequate level of service.

The scheduling of work matters to the provision of QoS, because there will often be a mix of high- and low-priority work in the system. If the provider's scheduling policy handles periods of overload well by ensuring sufficiently graceful degradation, this may reduce the amount of computational capacity required in order to achieve acceptable QoS. Alternatively, it may make it possible to make better use of their existing capacity by running at a higher average utilisation.

The paper will be divided into several sections, examining the background of the research, a classification and evaluation of QoS metrics, followed by the definition and evaluation of scheduling policies.

Section 2 will describe the industrial context that the work described in this paper is based on. This section will also provide the motivation for the metrics and improved scheduling presented throughout the rest of the paper. Genericised application, platform and scheduling models of the industrial scenario will be presented. These models will form the base for the evaluations throughout the rest of the paper, while their general nature, derived from models already in the literature, should ensure that the results of this paper are relevant outside the scope of the industrial scenario.

Section 3 will present a survey of metrics for each category identified in the industrial scenario. These categories are the Utilisation, Responsiveness and Fairness metrics. Several scheduling issues that have been observed to cause user dissatisfaction in an industrial case study will be described. The metrics proposed in the survey will be evaluated as to their ability to detect these scheduling issues. The identification of a situation where overly-high interleaving of work leads to low responsiveness lead the authors to propose a novel metric: the peak in-flight count. The evaluation identifies some inherent weaknesses in alternative metrics, such as their inability to expose issues related to workloads containing dependencies. Therefore, we will specifically draw attention to the insight power of the Schedule Length Ratio (SLR) metric [22] due to its appropriate handling of dependencies.

Section 4 presents a short survey of common scheduling policies. The authors then present a novel list scheduling policy termed Projected Schedule Length Ratio (P-SLR) that aims to perform well across all three categories of metrics, especially in situations of high- and overload. This novel scheduler will then be evaluated against the other surveyed schedulers in two ways. Firstly, an evaluation will be made using a range of synthetic workloads so that the performance of the scheduler can be evaluated across a wide parameter space. Secondly, to validate the results gained from the synthetic studies, an analysis of the performance of P-SLR will also be performed using a large-scale industrial workload.

2. Context

2.1. Industrial Scenario

The research in the paper was done in the context of an industrial case study, for an organisation operating in the aerospace industry. This organisation owns and operates a significant HPC capacity, organised into a grid of *clusters* with significant geographical distribution and joined by point-to-point network links. The

purchase of additional capacity happens sufficiently infrequently that for the scheduling of tasks, the platform can be considered to be static. The HPC is used primarily for Computational Fluid Dynamics (CFD) studies, although many other kinds of work are also run. Each CFD study (a *job*) is formed of a number of separate *tasks* that are linked through data dependencies. We were able to make use of the log files of the HPC that spanned a period of approximately 10 months and contained in excess of 100,000 jobs.

Each task runs on a number of *cores*, that are required to be within the same cluster for performance reasons. Tasks require input data files, and produce output files, which may then be consumed by further tasks, or returned to the user. The production and consumption of the data files mean that there are dependencies inherent between tasks. The patterns we found tended to reflect fork-join computing models. Furthermore, as the studies tended to combine several stages of work, we observed a pattern where each stage was computed in a fork-join manner, but with a single data dependency between each stage. Alternatively, some larger jobs contained detailed hand-crafted connections of data production and consumption between tasks.

The execution times of jobs, especially when used for analysing the performance of scheduling policies, have often been assumed to follow normal or uniform distributions [22]. However, we found a distribution of task and job execution times that follows a power law, or a logarithmic distribution. There are several classes of users, each producing differently sized jobs with differing responsiveness requirements. Their respective jobs have execution times of tasks ranging from a few minutes to several months. The users submitting the shortest jobs (minutes to hours) require a quick turnaround as a fast cycle time is essential to their productivity. We confirmed with the users that the longest jobs were not anomalies or jobs that had entered an infinite loop - instead, there is a need for final validations to be performed in very high fidelity. The longest jobs also tend to require the highest number of cores and in addition require significant disk and memory resources.

The tasks are executed without pre-emption - they either terminate or fail. On small-scale systems, this would seem to preclude the possibility of achieving good QoS, because short, urgent jobs may be stuck waiting for the largest ones to finish. However, with an HPC on the scale we observed, the rate that tasks finish is more than sufficient to ensure that something is always about to finish. This negates the need for the complexity of pre-emptive scheduling, because the time taken to wait for the next free resources is always small.

All the work runs on the same platform. There is a natural cycle during each day, where the vast majority of work is submitted during working hours. However, the queues on the platform are longest outside of working hours, as a significant fraction of the work is submitted at the end of the day with the results desired the next morning. Similar patterns are also evident on slightly longer timescales, with users submitting jobs at the end of the week to run over the weekend, and submitting large jobs before they go away on vacation. Despite these cycles of submission rates, the platform is almost always fully loaded and has some work queueing.

The scheduling policy currently used follows a 'Fair Share' scheme [20, 21]. This is a list scheduling scheme that attempts to mimic a partitioned scheduling scheme. Fair Share works by comparing each user and each group's entitlement against what resources they are currently using, and prioritising the work of those who are below their entitlement. This works poorly in practice, because although the longest-running jobs can require the most capacity, and hence the highest share, this leaves the smallest jobs with lower capacity, and therefore lowest share. This then means that the smallest jobs, with the highest responsiveness requirement are given the lowest queue priority. Balancing this tradeoff is done by manually adjusting the shares assigned to each group, but this balancing must be frequently re-done, because the load placed on the cluster by different groups varies over time. The automatic, online load balancing between the clusters is done purely on core utilisation statistics, and does not take into account the share allocations. This can also lead to poor performance, where there are too many jobs of one share group on one cluster, and too few on another.

Although this industrial scenario considers the classes of users and the teams that they are part of as being members of a single organisation, this need not be the case from a cloud perspective. In a cloud context, each 'team' in the organisation would represent a separate customer. The demands of each team/customer

on the infrastructure change over time, and are subject to significant peaks.

In the case study, we found that the perspectives of the system owners and the teams using the computing resources tended to differ. The system owners and administrators are concerned with metrics that show whether they are making the best use of the hardware that they own. Owners of the platform like the platform to be busy, because they feel that shows that the investment in such capacity was worthwhile. However, users tend to be less interested on whether the main computational capacity is busy or not. Instead, they care about getting their jobs back quickly. If the users' jobs ever do have to wait, however, they would like to be feel that their jobs are being treated with a level of fairness compared to those of other users of the system.

The fundamental research challenge, therefore, is to design a scheduling policy that can ensure responsiveness for the small jobs but avoid starvation for the largest jobs, while ensuring a high level of throughput. Although research into answering this question is set within the context of this case study, we believe that the scheduling problem it represents is not atypical for owners of substantial computing capacity. All the teams or customers require a level of QoS to be maintained, and it is the responsibility of the provider to ensure this. The next two sections will present models designed to represent the infrastructure, application and scheduling structures of an HPC or cloud resource provider.

2.2. Models

A composition of three primary models are considered as context for the definitions and evaluations undertaken in this paper. An application model represents the kinds of workflows run on HPC and cloud platforms when dependencies are present. A platform model represents the computing infrastructure owned by the provider. A scheduling model is also presented which captures what scheduling decisions are to be made, and where they are made.

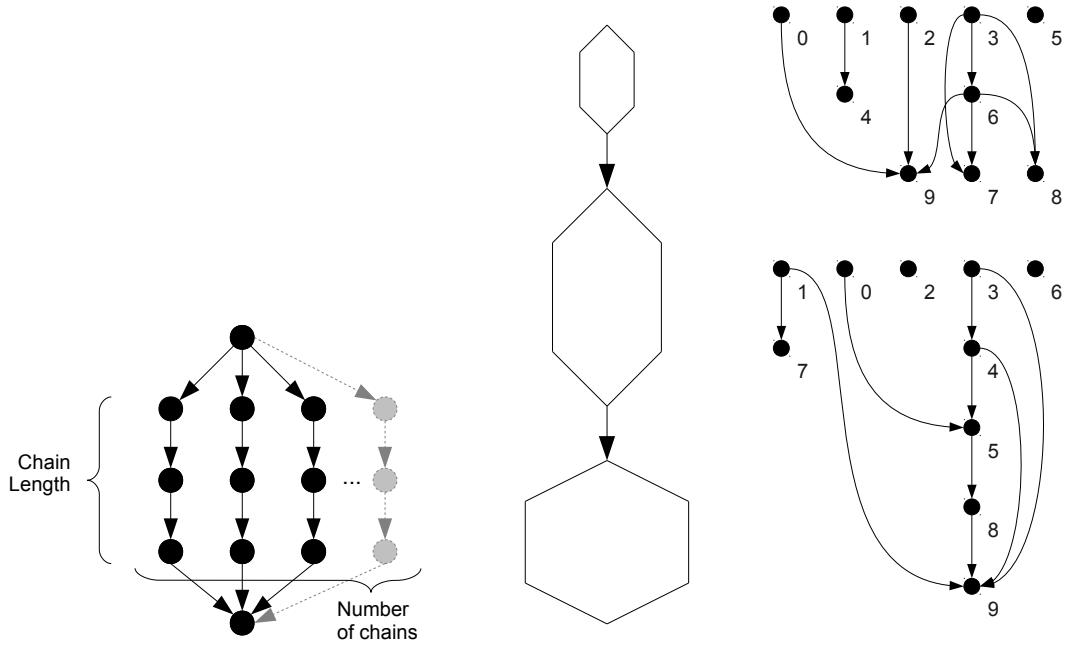
2.2.1. Application Model

The problem that is investigated in this paper is the dynamic scheduling of multi-core tasks with dependencies onto a static platform. Jobs are continuously arriving and require scheduling. The execution time of tasks is known when they arrive. Heterogeneity in the resources is considered in the sense of partitioning the grid into zones where tasks can and cannot run. Each task runs on one or more resources (cores) simultaneously. Each resource can only run one task at once.

Throughout this paper, the application model is considered to be as follows. A single, non-preemptible piece of work to be executed on one or more processors concurrently will be known as a task, denoted T^i. A set of tasks with dependencies between each other are grouped into a job, denoted J^k. A set of jobs will be known as a workload W. This follows the nomenclature of Chapin [7].

The dependencies inside a between tasks inside a job will take the form of a Directed Acyclic Graph (DAG), following the usual construction for HPC workflows, as referred to by [16, 22]. The structure of the DAGs can take a wide variety of forms. In this work, we consider four principal patterns, based on work in [5]. The first is that of independent tasks, which is a baseline model without dependencies. The second pattern is termed Independent Chains (see Figure 1a), and reflects the common fork-join parallel programming model. If each fork-join unit is considered as a block, the third pattern shows a chain of such blocks (Figure 1b). This third model is common where different stages of processing are required, and where each stage follows the fork-join model. These patterns of dependencies were observed in the workloads studied as part of our industrial case study. In order to capture a wide variety of graphs, the fourth model chosen was the Erdős–Rényi [9] or probabilistic dependencies model (Figure 1c). These more randomly-structured graphs are helpful to evaluate the scheduler over a wide variety of DAG shapes, but can also more closely parallel the shapes of hand-crafted dependency trees that were observed in the industrial case study.

Tasks and jobs have several parameters that can be defined, relative to a time-base. This paper uses a discrete model of time, with all events taking place at time ticks $\tau \in \mathbb{N}^0$. However, the following parameters and the metrics in Section 3.1 could equally be calculated for a continuous model of time.



(a) Independent Chains (b) Chain of Blocks (c) Erdős–Rényi (Probabilistic) Dependencies ($T = 10, P = 0.3$)

Figure 1: DAG Shape Patterns (reproduced from [5])

- Task execution time : $T^i_{\texttt{exec}} \in \mathbb{N}^\star$
- Task cores required : $T^i_{\texttt{cores}} \in \mathbb{N}^\star$
- Task start time: $T^i_{\texttt{start}} \in \mathbb{N}^0$
- Task finish time: $T^i_{\texttt{finish}} = T^i_{\texttt{start}} + T^i_{\texttt{exec}}$
- Job arrival time (not necessarily the same as start time): $J^k_{\texttt{arrive}} \in \mathbb{N}^0$
- Job start time: $J^k_{\texttt{start}} = \min\left(T^i_{\texttt{start}}\right) \bullet \forall\, T^i \in J^k$
- Job finish time: $J^k_{\texttt{finish}} = \max\left(T^i_{\texttt{finish}}\right) \bullet \forall\, T^i \in J^k$
- Job response time: $J^k_{\texttt{response}} = J^k_{\texttt{finish}} - J^k_{\texttt{start}}$
- Job total execution time: $J^k_{\texttt{exec}} = \sum\left(T^i_{\texttt{exec}} \times T^i_{\texttt{cores}}\right) \bullet \forall\, T^i \in J^k$
- Workload total execution time: $W_{\texttt{exec}} = \sum J^k_{\texttt{exec}} \bullet \forall\, J^k \in W$

A job is considered to be *in flight* during the interval $\left[J^k_{\texttt{start}},\, J^k_{\texttt{finish}}\right)$. The critical path time $J^k_{\texttt{CP}}$ (CP) of a job is the longest path through the DAG of the dependencies [15], and defines the minimum time that the job can be executed in even if the number of processors was unbounded. The edges of the DAG are weighted to represent possible network delays between tasks. If tasks are run on the same cluster, these delays are not manifested. However, tasks with different architectures may never be able to run on the same cluster, and hence have an unavoidable network delay. Any unavoidable network delays must be taken into account when determining the critical path of a job.

2.2.2. Platform and Scheduling Model

While each job that executes on a grid can be considered a batch job, we consider the scheduling problem over modern grids to be dynamic, not static. There are static schedulers that mimic dynamic behaviour by repeatedly queueing up a certain amount of work and then executing it, as in the Generational Scheduling approach [6]. However, this is unsuitable for our workload, because the large execution times of some jobs and low desired response times for others cannot both be satisfied in a single batch.

Therefore, we consider the family of scheduling algorithms known as list schedulers. These split the scheduling problem into two phases. One phase is known as ordering, where a set of tasks are sorted according to an ordering policy. The other phase is known as allocation, where tasks are assigned to resources in the order specified by the ordering policy.

In a simple case, a single list scheduler for a whole grid would suffice. A single queue would manage all incoming work, and a single allocator would send work to the various clusters that make up the grid as and when resources became free. However, in reality, this poses practical problems because a single scheduler is likely to be a significant bottleneck once the grid reaches a certain scale. Furthermore, grids have a distributed nature and are hence subject to network costs and limitations in bandwidth between their component clusters. This means that a single scheduler may not be able to have highly detailed and up-to-date information about the state of the whole grid, as simply communicating this information to a central node would swamp the available bandwidth.

We therefore consider a hierarchical scheduling model: a tree of list schedulers. Nodes in the tree are referred to as *routers*, and the leaves of the tree are the *clusters* that make up the computational resources of a grid. Jobs are submitted to the root of the tree, and are scheduled through child routers until they reach the leaves of the tree, which represent clusters. Each cluster itself also contains a list scheduler. An example of this platform, which will form the basis for the later synthetic evaluations (Section 4.4), is shown in Figure 2.

In the model considered in this paper, the routers cascade a job down to a cluster as soon as it arrives. The job will only spend time queuing once it has already been allocated to a cluster. The router list schedulers order the jobs in FIFO order, and allocate them between clusters based on a load balancer. This load balancing algorithm calculates the expected queue length by taking the amount of work (in core-seconds) in each cluster's queue and dividing it by the number of processing resources that cluster contains. The jobs are assigned to the cluster with the smallest expected queue length. These statistics are considered suitably high-level that they could be obtained by routers in a grid without imposing an undue performance penalty. Where the load balancing takes place between routers, each router will offer the best performing value of any the clusters beneath it.

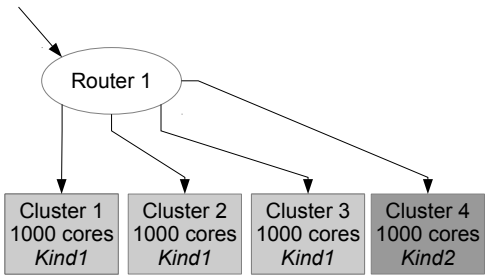


Figure 2: Platform Architecture

Each task in a workload and each cluster have an associated architecture. Each cluster can only contain processing units of a single architecture. Jobs can contain tasks of diverse architectures. If a router can find a sub-router that can supply all the architectures, the job is passed down to the sub router whole. If this is not possible, then the job is split up into its component tasks, and the tasks are allocated down the tree independently. This is done because the network costs between the routers in our model follow a thin tree model [19]. Network delays are only considered between tasks running on different clusters, as these links are likely to represent long-distance geographical links in reality. Within a cluster, network delays are assumed to be small enough to be negligible. This means that tasks running on different clusters will see network delays between them, but a multicore task running on a single cluster will not see network delays between the running cores. Multicore tasks are only run inside a single cluster, and it is assumed that there is at least one cluster in the grid able to provide sufficient processing resources.

Inside a cluster, allocation is simply done to processors as they become free (an Earliest Start Time allocation), because of the lack of network costs. Within a homogeneous cluster, this is equivalent to an

Metric		Utilisation	Responsiveness	Fairness
Workload Makespan		•		
Flow		•		
Average Utilisation		•		
Peak In-Flight		•		
Cumulative Completion		•	•	
Average or Worst Case	Speedup		•	
	Stretch		•	
	Schedule Length Ratio		•	
Standard Deviation	Speedup			•
	Stretch			•
	Schedule Length Ratio			•

Table 1: Insight given by selected metrics

Earliest Finish Time allocation [22].

Although routers and clusters both implement list schedulers, the ordering policy of the load balancer and the allocation policy of the cluster are trivial. This architecture effectively gives the result that allocation is done first through the load balancing in the routers, and then ordering is applied when the tasks are on the clusters. This is the reverse of most list scheduling architectures, and yet is suitable for the architecture of grids where perfect knowledge of the whole cluster is impractical to achieve due to the slow network links between clusters.

The key part of this model is the ordering policy applied on each cluster, because it is reasonable to assume that at this level, a great deal more information about the state of the cluster and the work to be performed can be analysed. It is also where the jobs will actually spend their time queuing. The cluster ordering policy will therefore be the one that will most affect the ability of the grid to achieve good QoS. Measuring the ability of a scheduler to achieve good QoS requires metrics, which the next section will describe and evaluate.

3. Metrics

3.1. Metric Definitions

In order to objectively compare schedulers, metrics are required. Different metrics are more or less relevant to different stakeholders in the system, however. We have selected metrics for consideration in this paper that represent each of the industrial stakeholder perspectives. The metrics relevant to the system administrators correspond to those related to utilisation, and the metrics that represent the users' point of view correspond to the responsiveness and fairness metrics. A further category of metrics was noted as being commonly used in the literature, and these were the relative metrics. Relative metrics compare schedulers by counting the number of 'best' schedules (by another metric) over a number of scenarios in a problem space.

All these metrics are considered here specifically within the context of the industrial scenario outlined above, which is the dynamic or online scheduling of jobs onto a fixed, distributed grid platform. However, the metrics are not limited to being used in such circumstances, and most should provide insight into both static and dynamic scheduling approaches. A summary of the applicability of each metric is presented in Table 1.

3.1.1. Utilisation Metrics

Utilisation metrics measure of how much of a platform's maximum potential is actually being used. Achieving a high throughput of work is contingent on achieving good utilisation. Wherever possible, it is desirable to avoid having idle resources if there is ever work queuing. The platform will be denoted as G, with a number of processing resources $G_{\texttt{cores}}$.

Workload Makespan. The classic metric used to compare schedulers is the workload makespan, which is widely referenced in the literature [1, 4, 12, 14, 15, 17]. This is defined by the time at which all the work in the workload was completed.

$$W_{\texttt{makespan}} = \max\left(J^k_{\texttt{finish}}\right) \bullet \forall\, J^k \in W \tag{1}$$

While some papers use only this metric for comparing schedulers, it is insufficient for measuring the responsiveness or fairness in a schedule. This is because, in the simulation of a dynamic system, the workload makespan may be mostly determined by the last few jobs in the workload to arrive. What it can help to measure, on the other hand, is utilisation, as a component of the Flow or Average Utilisation metrics. Because it requires the workload to complete execution, the workload makespan metric only really applies to the evaluation of static scheduling problems.

Flow. A measure of throughput is simply to count the number of tasks or jobs completed over the workload makespan. This is known in the literature as flow [3].

$$\frac{|W|}{W_{\texttt{makespan}}} \tag{2}$$

Flow does not attempt to account for the differing sizes of work, so a platform may be able to achieve wildly different values of flow depending on the makeup of the workload. This makes it

In a dynamic system, it may not be possible to measure the makespan of a workload, because work is continually arriving. In this case, flow can be defined as the number of jobs to finish in a given time interval $(\tau_{\texttt{start}}, \tau_{\texttt{finish}}]$.

$$\frac{\left|J^k\right|}{\tau_{\texttt{finish}} - \tau_{\texttt{start}}} \bullet \forall J^k \in W \wedge \tau_{\texttt{start}} < J^k_{\texttt{finish}} \leq \tau_{\texttt{finish}} \tag{3}$$

Average Utilisation. A further metric can be derived from the workload makespan, known as average utilisation [13] or efficiency [22]. This is defined as the proportion of the possible execution time determined by the workload makespan that was actually consumed. The number of processing units in the grid can be denoted $G_{\texttt{cores}}$.

$$\frac{\sum J^k_{\texttt{exec}}}{W_{\texttt{makespan}} \times G_{\texttt{cores}}} \bullet \forall J^K \in W \tag{4}$$

This metric can also be extended to dynamic systems by taking the utilisation between two points in time, although the calculation for this is a little more involved because it has to consider the calculation for tasks that are running at the interval time points. Interval utilisation is useful because weekly or daily average utilisation values can be monitored.

Peak In-Flight Count

As mentioned in Section 2.2.1, a job can be considered in-flight between when the first task of that job starts execution and the last task of that job finishes. We propose a novel metric, known as peak in-flight count, that gives the maximum number of jobs in flight at any given time.

$$\max\left(\left|J^k\right| \bullet \forall J^k \in W \wedge J^k_{\texttt{start}} \leq \tau < J^k_{\texttt{finish}}\right) \bullet \forall \tau \in \left[0, W_{\texttt{makespan}}\right] \tag{5}$$

This can be used to determine how much the scheduler has interleaved the jobs in the workload. Very high levels of interleaving may indicate scheduling problems, especially because high peaks may indicate that some jobs are starving for resources. The peak in-flight count can also reveal the effect of network delays. An abnormally high peak in-flight count might indicate that the scheduler is starting work on new jobs because all the current in-flight jobs are blocked waiting for network transfers to complete. This may point to using an alternative scheduler that is better suited to avoiding network bottlenecks.

3.1.2. Responsiveness Metrics

Responsiveness metrics compare how a scheduler is able to keep job latency low. There will always be a minimum time that a job will take to execute, and this is determined by its critical path. However, the time spent queueing or on network transfers will impact the responsiveness of a job. Responsiveness metrics can be a tool for measuring how well the scheduler is able to cope under periods of heavy load. The metrics of Speedup, Stretch and SLR are defined for each job in a workload. Therefore, the average value of these metrics for all the jobs in a workload can be used to provide a single value to compare scheduler performance. It can also be useful to compare the worst-case performance of the responsiveness metrics, because it is the users whose jobs are experiencing worst-case performance that will be the ones to complain, especially if the worst-case is significantly different to the average.

Cumulative Completion. A metric that rewards early completion of work, and hence good average responsiveness, was proposed by Braun et al. [4]. Whereas the utilisation metrics only derive value from the time the workload was completed, this gives some insight into the way this was achieved. This metric calculates the sum of completed *job* execution times at each time tick in the execution. Because it is assumed that only a completed job is useful to a user, it can only count the completed tasks' execution times once the whole job is finished.

$$\sum \left(1 + W_{\texttt{makespan}} - J^k_{\texttt{finish}}\right) \times J^k_{\texttt{exec}} \bullet \forall J^k \in W \tag{6}$$

The cumulative completion metric values work being completed early on in the schedule, by cumulating the values of completed jobs at each subsequent tick. If the workload makespans between schedules are different, the values of cumulative completion are not directly comparable. Therefore, where cumulative completion values need to be compared, the cumulative completion value should be calculated with the workload makespan value of the longest schedule.

This metric also gives partial insight into utilisation, because schedulers that achieve higher utilisation and higher throughput will cause more jobs to finish sooner, and hence raise the Cumulative Completion value. A shortcoming of this metric is that it is most suited to static schedules, because the finishing of the workloads is all relative to their makespan. However, it can be extended to the dynamic case by only sampling jobs that arrived in a given duration.

Speedup. A common metric to measure responsiveness is known as Speedup Topcuoglu et al. [22]. It is defined as how much faster the job was able to run compared to if it had been run on a single processor.

$$\frac{J^k_{\texttt{exec}}}{J^k_{\texttt{response}}} \tag{7}$$

This can be useful to see how much parallelism the scheduler has been able to extract from the job. However, in most HPC and grid systems, jobs are usually designed to be highly parallel in order to take the fullest advantage of the grid platform and because it would take vastly too long on a single processor. Therefore, while a speedup above 1 may intuitively sound desirable, speedup values may only be considered acceptable at a much larger value. Furthermore, it has no notion of comparing the actual speedup to the maximum possible speedup, when dependencies are present, because it does not take into account the critical path.

Stretch. Stretch is the reciprocal value to speedup, as described by [3].

$$\frac{J^k_{\texttt{response}}}{J^k_{\texttt{exec}}} \tag{8}$$

The stretch metric is useful because it removes the effect that jobs of different sizes have on their execution times. It shows the 'retardation' of jobs due to the scheduling and load of the system. However, it may be somewhat misleading because the minimum execution time of a job is not necessarily correlated to its total execution time. This is because the parallelism available in two jobs with the same total execution time can

be different due to differences in the core count of tasks or the structure of dependencies (see an examination of this issue in Section 3.2.3).

Schedule Length Ratio. To counteract the problem of the stretch metric not taking into account the minimum execution time of a job, Topcuoglu et al. [22] introduced the concept of Schedule Length Ratio (SLR). This is a similar metric to stretch, but is defined relative to the critical path rather than the total execution time. This is because the shortest execution time of a job on a highly parallel platform is determined by the length of its critical path.

$$\frac{J^k_{\texttt{response}}}{J^k_{\texttt{CP}}} \tag{9}$$

Of the three responsiveness metrics, SLR is the most representative of the performance of the scheduler alone. This is because it is simply a comparison between the actual and ideal response times. SLR is independent of the total execution time or the parallelism available in the job. The SLR metric is particularly useful to measure scheduler performance where jobs have a wide variation in the lengths of their critical paths even if their total execution times are the same. This is pertinent because this situation was observed in the industrial case study.

The ideal value for SLR is represented as 1, where the actual response time is equal to the ideal response time. This ideal value may be impossible to achieve in a finite grid. Furthermore, network delays that are not present on the critical path, but are still introduced by the scheduling decisions made, may contribute to raising the SLR value above 1.

To obtain a single value for the performance of the scheduler over a whole workload, the mean or worst-case SLR values for the whole workload can be used. These metrics are particularly useful in the case of system overload, where some SLR values must increase over a value of 1.

3.1.3. Fairness Metrics

It is possible to achieve a kind of perfect fairness in a naïve way by only running a single job at a time. However, this will almost certainly mean that utilisation and throughput over the whole grid are unacceptably low. This means that there can be a tradeoff in a non-pre-emptive system between fairness and utilisation. Hence, metrics are needed to quantify the level of fairness, to ensure that the tradeoff is managed appropriately. As far as we have found, fairness metrics with respect to grid scheduling have been little mentioned in the literature, because the overwhelming focus is on achieving high throughput. There may be an underlying assumption that by raising utilisation, responsiveness is maximised, and hence fairness will be near optimal as well. This assumption may hold when the task/job execution times follow a normal distribution. However, it breaks down when a power-law distribution is encountered, because even if there is high utilisation, this may be where all the largest jobs are running, and the smallest jobs experience very poor responsiveness. Therefore, when a power-law distributed workload is encountered, it is necessary to measure fairness.

The average values of the Speedup, Stretch and SLR metrics can be used to gauge the responsiveness a scheduler is able to achieve with a given workload. By examining the distribution of the responsiveness metrics, however, we propose that fairness metrics can be developed. Tight clustering of the responsiveness values may indicate a fair distribution of grid resources to jobs. The spread of the responsiveness values can be measured using the standard deviation of each of the responsiveness metrics.

The importance of measuring fairness can be illustrated with the following example. A First In First Out scheduler might introduce a relatively constant delay to all jobs that come through the system. However, this would penalise the SLR of jobs with a short critical path far more than that for jobs with a long critical path. This is likely to be perceived by users as an unfair situation. Furthermore, this may be particularly undesirable because short jobs may well also be the ones for which responsiveness is the most important, as was observed in the industrial case study.

A small value for the standard deviation of the responsiveness metrics is likely to be considered fair if the desire is to treat each job equally. However, the perception of fairness can depend on human factors external

to the system. Discussion of these human factors that may imply some kind of prioritisation, capacity reservation or otherwise is beyond the scope of this paper.

3.1.4. Relative Metrics

A further common means of comparing schedulers is by their relative performance over a set of problems. For a given problem instance, the performance of all the considered schedulers are compared against a given metric, often the workload makespan [17]. The winner is then decided. This is repeated over a number of problem instances. The 'best' scheduler is then considered to be the one that had the highest number of wins over the problem space.

These approaches are known as relative metrics. Relative metrics can often be useful for real-world scheduling problems, because finding the optimal schedule is computationally intractable. A simple count may not be able to show how much better the best scheduler is. Where a numerical value for relative performance is desired instead of a count, it is common to compare the metric(s) for the considered scheduler against some accepted 'baseline' scheduler [17]. While they may help in the end decision of which the best scheduler is, they do not provide any greater insight into the schedules produced than the underlying metrics that they are based on. Therefore, they will not be evaluated in this paper.

3.2. Metric Evaluation

Metrics are used to provide insight into schedules. The different classes of metrics defined above provide different kinds of insight. This section will apply the metrics to three example schedules that contain known scheduling issues. The ability of the metrics to identify the issues involved will be evaluated. The examples show the importance of being able to measure issues of utilisation, responsiveness and fairness, respectively. The examples contained in this section are deliberately small so that they can be completely described briefly, yet still demonstrate the presence of the scheduling issues. For the purposes of simplicity, all the jobs are given arrival times of $\tau = 0$. Nevertheless, they are designed to be viewed as dynamic scheduling problems, as the issues of responsiveness and fairness are less relevant to static scheduling problems.

The discussion here will attempt to identify those metrics that provide the best insight into these scheduling issues. This evaluation will be reinforced and validated by applying these metrics to the larger-scale scheduler evaluation in Section 4.4.

3.2.1. Low Utilisation Issue

If the packing of tasks on to processors is not sufficiently dense, then low utilisation of the processors will result. Graham [12] contains a classic example of contrasting schedules. The workload given by Graham is presented in 3a and is intended to run on three processors. Schedule A is Graham's workload scheduled with an anomaly that increases makespan, whereas schedule B is a schedule without the anomaly (see 3b). Metrics for these two different schedules are presented in Tables 3d and 3c.

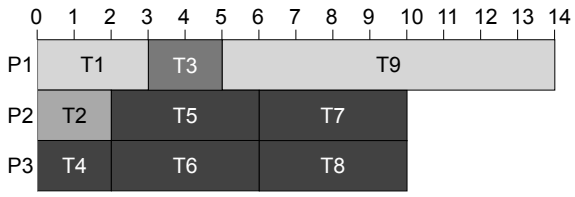
The significant feature of this workload is that the critical path of J^1 is long enough that it defines the minimum workload makespan (Table 3d). In schedule A, the whole workload makespan is extended because T^3 delays the execution of T^2. The flow and average utilisation metrics depend on the workload makespan. Because the workload is the same but its makespan in schedule A is longer than in schedule B, then the flow and average utilisation metrics are lower for schedule B. The peak in-flight count metric remains the same, although Schedule B only has a single duration of the peak between τ_0 and τ_2, whereas schedule A has two periods of time at the peak value, $\tau_0 - \tau_2$ and $\tau_3 - \tau_5$. The utilisation metrics are useful here, because they show that schedule B contains less wasted capacity in the schedule, and hence makes more efficient use of the resources.

All the responsiveness metrics except cumulative completion show an improvement from schedule A to schedule B (Table 3d), because three jobs finish earlier and only one job finishes later. The cumulative completion metric rewards the early finish of the larger J^4 in schedule A. This is because the cumulative completion metric rewards jobs that finish earlier, and the movement of empty scheduling space to the end of a schedule. If a new job arrived only after this space had passed, the capacity represented by the empty space would have been wasted. This stands in contrast to the average utilisation metric, which would suggest

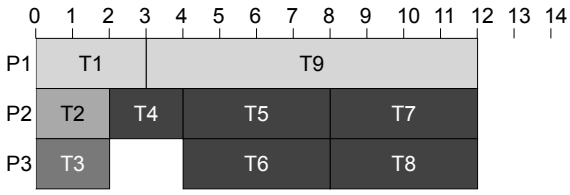
Job	Task	Dependencies	$T_{\texttt{exec}}$
J^1	T^1	-	3
J^1	T^9	T^1	9
J^2	T^2	-	2
J^3	T^3	-	2
J^4	T^4	-	2
J^4	T^5	T^4	4
J^4	T^6	T^4	4
J^4	T^7	T^4	4
J^4	T^8	T^4	4

(a) Workload

Schedule A



Schedule B



(b) Gantt Chart

Metric	J^1		J^2		J^3		J^4	
	A	B	A	B	A	B	A	B
Stretch	1.17	1	1	1	2.5	1	0.56	0.67
SLR	1.17	1	1	1	2.5	1	1.67	2.0
Speedup	0.86	1	1	1	0.4	1	1.8	1.5

(c) Job Metrics

Metric	A	B	B/A
			Utilisation
Workload Makespan	14	**12**	0.86
Average Utilisation (%)	81.0	**94.4**	1.17
Flow (jobs/tick)	0.29	**0.33**	1.17
Peak in-flight	3	3	1.00
			Responsiveness
Mean Stretch	1.31	**0.91**	0.70
Worst-case Stretch	2.50	**1**	0.40
Mean SLR	1.59	**1.25**	0.78
Worst-case SLR	2.50	**2.00**	0.80
Mean Speedup	1.02	**1.13**	1.10
Worst-case Speedup	0.40	**1**	2.50
Cumulative Completion	**148**	142	0.96
			Fairness
Std. Dev. Stretch	0.84	**0.17**	0.20
Std. Dev. SLR	0.67	**0.5**	0.74
Std. Dev. Speedup	0.58	**0.25**	0.42

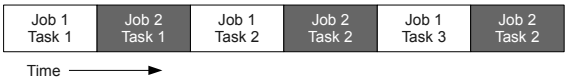
(d) Workload Metrics

Figure 3: Low Utilisation Issue Example

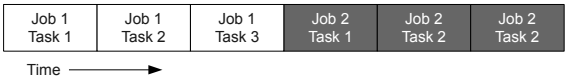
Job	Task	Dependencies	$T_{\texttt{exec}}$
J^1	T^1	-	1
J^1	T^2	T^1	1
J^1	T^3	T^1	1
J^2	T^1	-	1
J^2	T^2	T^1	1
J^2	T^3	T^2	1

(a) Workload

Schedule A:



Schedule B:



(b) Gantt Chart

Metric	A (J^1)	A (J^2)	B (J^1)	B (J^2)
Stretch	1.66	2.00	**1.00**	2.00
SLR	2.5	2.0	**1.5**	2.0
Speedup	0.6	0.5	**1**	0.5

(c) Job Metrics

Metric	A	B	B/A
			Utilisation
Workload Makespan	6	6	1
Average Utilisation (%)	100	100	1
Flow (jobs/tick)	0.30	0.30	1
Peak in-flight	2	**1**	0.50
			Responsiveness
Mean Stretch	1.83	**1.50**	0.82
Worst-Case Stretch	2.00	2.00	1
Mean SLR	2.25	**1.75**	0.78
Worst-case SLR	2.50	**2.00**	1
Mean Speedup	0.55	**0.75**	1.36
Worst-case Speedup	0.50	0.50	1
Cumulative Completion	9	**15**	1.67
			Fairness
Std. Dev. Stretch	**0.24**	0.70	2.94
Std. Dev. SLR	0.35	0.35	1
Std. Dev. Speedup	**0.07**	0.35	5

(d) Workload Metrics

Figure 4: Multiple Waits Issue Example

that the lower average utilisation is better, but does not take into account where in the schedule this low utilisation phase appears.

A high value for cumulative completion may be valuable, but it does not indicate how fairly the jobs in the workload are being treated. The fairness metrics (Tables 3d) show that schedule A is an improvement over schedule B. This is because J^1 and J^3 complete sooner, and hence closer to their critical path time. The finish time of J^4 is extended, but as this is one of the larger jobs, the increase is less when taken as proportional to its execution and critical path time. This means that the variation as a proportion of the job responsiveness metrics for each job is lower (Table 3c), giving a lower standard deviation of these metrics which defines an increase in fairness.

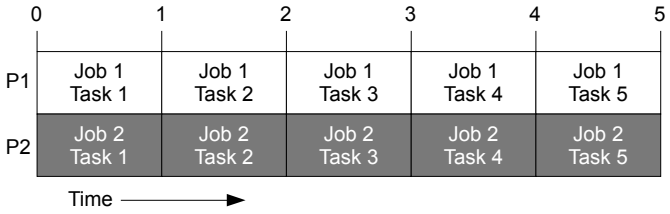
From this example, it can be seen that utilisation metrics are important, because they can reveal inefficiency in how the platform is being used. The responsiveness metrics for each job show how smaller jobs are proportionally affected more than larger ones when they are subjected to delay. This is further revealed in the fairness metrics, which show an improvement in fairness (lower variation) for schedule B compared to schedule A.

3.2.2. Multiple Waits Issue

The multiple waits problem is exhibited when there is too great an interleaving of jobs in a system, leading to low responsiveness even though utilisation is high. This example evaluates how well the metrics can reveal the multiple waits problem. In this example, a single processor is available for execution, and as the two jobs arrive at the same time, the one with the lower index begins first. The dependencies of two jobs are shown in Table 4a.

Job	Task	Dependencies	$T_{\texttt{exec}}$
J^1	T^1	-	1
J^1	T^2	T^1	1
J^1	T^3	T^1	1
J^1	T^4	T^1	1
J^1	T^5	T^2, T^3, T^4	1
J^2	T^1	-	1
J^2	T^2	T^1	1
J^2	T^3	T^2	1
J^2	T^4	T^3	1
J^2	T^5	T^4	1

(a) Workload



(b) Gantt Chart

Metric	Job 1	Job 2
Stretch	1	1
SLR	**1.66**	1
Speedup	1	1

(c) Responsiveness Metrics

Figure 5: SLR Advantages Example

A trivial example of the multiple waits problem is shown in Figure 4b. Schedule A shows a high interleaving of the two jobs, as could have been scheduled by a list scheduler using FIFO ordering over *tasks* (e.g. the scheduler given in Section). Schedule B, on the other hand, shows the two jobs executed in sequence. This could have been created using a list scheduler using a FIFO ordering over jobs instead of tasks (e.g. scheduler from Section). The most pertinent feature of this example is that J^1 completes execution significantly earlier under schedule B than under schedule A, while J^2 completes execution at the same time (Figure 4b).

The utilisation metrics for this example that depend on the makespan are the same, because the workload makespan is the same. Only the utilisation metric of peak in-flight count shows a difference between these two schedules. The peak of 2 in schedule B suggests that there is greater than desirable interleaving of work, because the peak in-flight count is greater than the processor count.

The earlier completion of J^1 in schedule A means that a higher cumulative completion value is achieved (Table 4d). The average stretch, speedup and SLR metrics also favour schedule A, also because J^1 finished earlier. It is important to note that in schedule A, the stretch metric has a value of 1, whereas the SLR metric has a value of 1.5. This is because stretch is defined relative to execution on a single processor, which matches this situation. Having a stretch value of 1 may seem to indicate that there is no further improvement that can be made. However, the dependency structure in J^1 shows that there is parallelism that has not been exploited in this example. The SLR metric reveals the potential for a lower response time if there were more processors available.

The fairness metrics of the standard deviation of stretch and speedup indicate instead that Schedule B is to be preferred(Table 4d), because the two jobs finish closer in time. While this seems more fair, this should be considered in light of the decrease in responsiveness. Interestingly, the standard deviation of SLR does not change, indicating that when dependencies are taken into account, the schedules are equally fair.

This example demonstrates that responsiveness metrics of mean SLR, stretch and speedup are important, because they reveal how quickly each job is getting through the system. The cumulative completion metric also reveals the benefit of having some jobs finish earlier, even if the workload makespan is the same. The peak in-flight count is also shown to be useful, because it reveals where there is excessive interleaving of jobs. The responsiveness metrics also show that while a schedule may seem fairer, it may also be less responsive, and both sets of metrics should be considered if a tradeoff is to be made between them.

3.2.3. Advantages of SLR over Stretch/Speedup

This example is intended to highlight the advantage gained by using the SLR metric over the stretch of speedup metrics on tasksets containing dependencies. The schedule shown in Figure 5b contains two jobs, with identical numbers of tasks and identical execution times (of tasks and of the whole job), as defined in Table 5a. The only thing that differs between the jobs is their dependency structure, and hence the length of their critical path. The critical path length of J^1 is 3, whereas for J^2 the critical path length is 5. When these two jobs are scheduled onto a single processor each, the SLR metric reveals that this is optimal for J^2, because of its dependency structure, and yet it is suboptimal for J^1, because J^1 has further opportunities for parallelism (see Table 5c). Nevertheless, the stretch and speedup metrics cannot distinguish between the scheduler's performance, because both jobs have the same total execution time.

This section has shown, using the examples of three issues, that the measurement of utilisation, responsiveness and fairness is important. Scheduling issues can occur in each of these categories. Where dependencies are concerned, it has been shown that taking the critical path of jobs into account (as the SLR metric does) is essential, as there is otherwise a loss of insight. Now we have defined and evaluated a number of metrics, they will be applied to the evaluation of a number of scheduling policies running over synthetic and industrial workloads.

4. Scheduling

4.1. Policy Definitions

This section will define a set of ordering policies that can be applied on each cluster, within the platform model outlined in Section 2.2.2 above. The next section will evaluate these policies as to their ability to achieve good utilisation, responsiveness and fairness with a range of different synthetic workloads and load ratios, along with an industrial workload.

As previously mentioned in Section 2.1, the load on grids and clouds can vary, and it is likely that there will be times when the arrival rate of work is greater than the maximum processing capacity. If a platform is perennially overloaded, then the queues will grow in an unbounded manner. Under some ordering policies, it may be the case that under such periods, certain jobs will never reach the head of the queue, and so suffer from starvation. On the other hand, the structure of some ordering policies will guarantee that all jobs will eventually run, and these are known as starvation-free ordering policies.

4.1.1. Random

The random ordering policy randomly chooses from the set of ready tasks which should be the next task to run. This policy is useful as it can provide a baseline against which the performance of other ordering policies can be compared, because it operates with no information about the workload. For any ordering policy to be worth using, it must demonstrate that it produces significantly better schedules than the random scheduler. Although in the short-term, the random scheduling policy could suffer from starvation, it is statistically improbable that a job could starve forever.

4.1.2. FIFO Task

The FIFO Task orderer is another simple ordering policy, albeit one that is widely used. Jobs are decomposed into their component tasks. As tasks become ready, they are placed into a FIFO queue. Tasks are removed from the head of the queue and allocated to the grid as resources become free. Any FIFO queues are starvation-free, because while ever the cluster is executing work, jobs will rise to the head of the queue and be executed in the order they arrived.

4.1.3. FIFO Job

This is a slight modification to the FIFO Task ordering policy, designed to avoid the multiple waits problem. Ready tasks in the queue are ordered first by the order in which their respective jobs were submitted, then by the order in which they became ready. FIFO Job is starvation-free in the same way as FIFO Task, because it is based on a FIFO queue.

4.1.4. Fair Share

The 'fair share' ordering policy is designed to approximate the behaviour of a partitioned scheduling scheme, but implemented as a list scheduler Oracle Corporation [20], Platform Computing Corporation [21]. The fairshare orderer is based on a tree of *shares*. The root of the tree has a 100% share of the cluster. Each branch of the tree divides out this share until the leaves of the tree are reached. These leaves represent the users. The leaves of the tree do not all need to be at the same depth. Each job in the system is assigned a path in the tree, which must be a leaf.

As the grid is executing, a *share factor* for each task is calculated. This is achieved by calculating the share allocated to that path over the share actually used by each node on the path. The share factor for each task is then the multiplication of these factors all the way down the tree until the leaf is reached. The tasks are then ordered by their share factor.

This is intended to ensure that if the cluster is busy that the resources of the cluster are distributed according to the share tree. However, if the cluster is underutilised, then there is still scope for jobs to run outside their share. Jobs with the same share factor are processed in FIFO order. Dependencies are not taken into account by the fair share scheduler, meaning that only ready tasks are added to the task queue.

The Fair Share ordering policy is not starvation-free, even though jobs with the same share factor are run in FIFO order. This is because a job requiring a large number of processors that was assigned to a leaf with a low share may never run because it would need more than its share to run.

4.1.5. Longest and Shortest Remaining Time

The Longest Remaining Time First (LRTF) and Shortest Remaining Time First (SRTF) ordering policies use concept of *Upward Rank* introduced by Topcuoglu et al. [22]. Upward Rank is defined for each task, and is the length of the critical path that remains to be completed after the task has executed. The longest and shortest remaining time ordering policies sort the list of tasks by decreasing and increasing Upward Rank, respectively. The highly regarded HEFT scheduler uses the LRTF ordering policy [22]. This ensures that the largest tasks are started first, which is a useful heuristic when performing bin-packing to optimise the workload makespan for static schedules. In a dynamic schedule, however, the workload will never end, so prioritising other metrics, such as those for responsiveness or fairness is advantageous. Using the LRTF scheduler in a dynamic system would mean that the smallest tasks suffer proportionately far more than the largest tasks. Furthermore, LRTF and SRTF are not starvation-free. This is because in overloaded clusters, either the shortest or the longest jobs may wait forever.

4.1.6. Projected SLR

Having considered the benefit realised by using the SLR metric to measure scheduling performance, we now present a novel ordering algorithm that we call Projected-Schedule Length Ratio (P-SLR). The P-SLR ordering policy uses the concept of upward rank, but uses it to give a projection of when the job would finish if the considered task was run immediately. This projection of the job finish time is used to calculate a projection of what the job's SLR metric would be, which is used as the basis of the ordering policy.

The nominal intent of the P-SLR orderer is that as the load of the system rises (especially into a state of overload), all jobs should 'suffer' equally. At a scheduling instant, the upward rank of every task is used to predict what the SLR of the job would be if this task were executed immediately. The task where the predicted SLR is largest is then run first. This means that the task that is currently most 'late' is the one to be run next. The advantage of using the SLR metric is that small jobs can 'jump' the queue to run quickly because their SLRs are more sensitive to the same waiting time. However, eventually, even large jobs will run because their projected SLR will rise as they wait, just more slowly than for small jobs. This means that the P-SLR orderer is starvation-free, a desirable attribute in systems where overload may be present.

A particular factor of note that is shown in Algorithm 1 is that the predicted finish time is incremented by 1. Two jobs of differing sizes could be submitted at the same scheduling instant. Without this increment, both jobs' projected SLR would be 1, and hence the choice between them would be arbitrary. By adding a lateness penalty to every calculation, the projected SLR is able to distinguish between short and long jobs that arrive at the same time, and prefer running the shorter one first.

Algorithm 1 Projected SLR ordering algorithm

```
projected_slr(task):
    job_predicted_finish_time = now() + task_upward_rank + 1
    job_predicted_response_time =
        job_predicted_finish_time - job_submit_time
    job_predicted_SLR =
        job_predicted_response_time / job_critical_path
    return job_predicted_SLR
```

Another factor to note is that tasks not on the critical path for a job may have a small upward rank, even though they may be ready early on. This can mean that the projected SLR for these tasks is less than 1. However, the result of this is that they are prioritised lower than the tasks on the critical path; a desirable attribute.

4.2. Hypotheses and Testing Approach

The new P-SLR policy need to be evaluated in order to compare its performance to the other policies given. This section will give three hypotheses that we will investigate, along with the ways in which the hypotheses will be tested.

- Hypothesis 1: The P-SLR orderer gives schedules with a higher degree of fairness than alternative policies. i.e. it does not particularly favour small or large jobs, but achieves the same responsiveness across the range of job total execution times.

To measure fairness, the standard deviation of the SLRs for each workload will be used. These will be displayed graphically as a box plot, to show the relative measures. Statistical significance will be tested using a repeated measures t-test. It is useful to use the repeated measures test, because the workload and load ratio are the same, and only the ordering policy has changed; this means that pairs of values can be compared. The threshold for statistical significance is set at the 5% confidence interval. The null hypothesis will be that the P-SLR ordering policy gives values of SLR Standard Deviation indistinguishable from the alternative scheduler.

However, it is useful to visualise how the different ordering policies achieve fairness across the spectrum of job execution times. This will be achieved by plotting the worst-case SLR value by the decile of job execution time. This will make it possible to see which schedulers effectively prioritise large or small jobs, or achieve a fair balance of SLRs across the range of job execution times. At low load ratios, it is possible that no jobs would prioritised over others because anything can run immediately, and hence the plot would be uninformative. Therefore, the worst-case SLRs by decile of execution times will be plotted at 120% load ratio.

- Hypothesis 2: The P-SLR orderer gives schedules with a higher degree of responsiveness than alternative policies

As outlined above, responsiveness is best measured using the SLR metric for each job in a workload. The responsiveness of the P-SLR orderer will be evaluated by examining the worst-case SLR for each workload when run with each scheduler. This will be evaluated for statistical significance also using the repeated measures t-test. For further insight, the worst-case SLR metric will be plotted against load ratio to see how the different ordering policies cope as load increases. Worst-case SLR is a better metric of responsiveness than mean SLR, because the mean could mask poor performance on a small subset of jobs, even though that poor performance may be critical to users.

At each step of load ratio, the worst-case SLRs of each workload will be recorded for each ordering policy. To see if P-SLR is the most responsive, the percentage of cases in which P-SLR dominates the other ordering policies will be calculated. To check whether this dominance is statistically significant, the repeated measures t-test will also be used. The null hypothesis is that the P-SLR orderer gives no significant improvement in worst-case SLR values.

- Hypothesis 3: The P-SLR orderer does not give a significantly different rate of utilisation over alternative policies

Utilisation metrics that use the makespan are not ideal for measuring a dynamic system, because the makespan will tend to be most influenced by the last few tasks to arrive. Average utilisation may be poor if most of the cluster is idle while the last task finishes. However, if an ordering policy gave significantly lower utilisation than others, it may not be as desirable because it cannot make good use of the cluster. Average utilisation values given by each ordering policy will be plotted in a box-plot to see the range of values. The null hypothesis is that there is no statistically significant difference between the average utilisation values of the other orderers and the P-SLR orderer.

Further metrics will not be analysed in as much detail, but will still be plotted for completeness. The cumulative completion metric, using a standard makespan for all schedules, will be plotted on a box-plot. This will be to see how quickly the schedulers are able to finish the work that has arrived. The peak in-flight count will also be plotted on the box-plot, to see how much interleaving of jobs is made to happen by the ordering policies.

4.3. Synthetic Test Generation

In order to fairly evaluate the performance of these scheduling algorithms, they need to be applied to equivalent workloads on an equivalent platform. We use a simulation framework in order to be able to perform a comprehensive comparative analysis. The application model allows a great deal of flexibility in the kinds of workloads that could be produced. Yet in order to be able to fairly evaluate the scheduling policies, it is important to generate a wide variety of realistic workloads.

Both the tasks and jobs that form part of a work can have execution times that follow specific distributions. For this evaluation, we created workloads with uniform and power-law distributions. These distributions were obtained using a method based on UUnifast-Discard [8], and described in detail in [5].

The shapes of the DAGs that define the dependencies have already been described in Section 2.2.1. Workloads were generated according to these shapes, with their parameters presented in Table 2. For the Fair Share scheduler, each job was randomly placed into one of 5 equal shares.

The level of load on the platform can be measured by the percentage rate at which work is arriving compared to the maximum rate at which this work can be processed. Comparing schedulers at a range of loads is essential, because of the variation experienced in grid and cloud scenarios. A load ratio for a workload can only ever be defined with relation to a platform, yet it is desirable to be able to adjust the load ratio independently of the workload and platform. This can be achieved by adjusting the inter-arrival times of jobs, following the algorithm described in [5]. The evaluations were run at load ratios between 80 and 120%, in increments of 10%, to examine how gracefully performance degraded under the different policies as the threshold of overload is passed.

Networking delays are considered in this model whenever there needs to be communication between clusters. The edges of the DAG representing the dependencies can be weighted to represent the data transfer requirements. Rather than calculating the data volume, instead, we use the computation to communication ratio (CCR) to define how long the network transfer will take. If a task's execution time is $T_{\texttt{exec}}$, then the time taken to transfer data from its host cluster to any other cluster is taken to be $T_{\texttt{exec}} \times CCR$. A CCR value of 0.2 was used so that network delays were present, but were relatively small compared to the computation costs.

The platform used in simulation is fixed, and consists of 4 clusters of 1000 cores each. Three of the clusters used an architecture *Kind1*, and one of the clusters was of *Kind2*. The proportion of the workload that requires the *Kind2* architecture (see Figure is deliberately lower than the share of the grid, in order that the network was the the bottleneck in running tasks on the *Kind2* cluster. The network topology was that of a central router connected to each cluster with the same bandwidth network link. This network architecture and topology is displayed in Figure 2.

	Uni Ind.	Log. Ind.	Prob. Dep's	Fan-In/ Fan-Out	Chain of Fan-In/Fan-Out
Task core count distribution	random selection from (1,5,10,15,20)				
Number of workloads	30				
Total Workload Exec Time	10^{10}				
Proportion of Kind1:Kind2 Tasks	80:20				
Communication to Computation Ratio	0.2				
Distribution of Job Exec times	Uni.	Log.	Log.	Log.	Log.
Distribution of Task Exec Times	-	-	Log.	Log.	Log.
Number of Jobs	10000	10000	1000	1000	1000
Tasks per Job (uniform distrib.)	1	1	1-20	-	-
Dependency Probability	-	-	0.3	-	-
Fan-out width	-	-	-	1-10	1-5
Fan-out length	-	-	-	3-15	3-15
Fan-out chain length	-	-	-	-	1-3

Table 2: Parameters used in workload generation

4.4. Scheduler Evaluation (Synthetics)

To give confidence in our investigation the performance of the scheduling policies, a large number of synthetic workloads were generated, according to the parameters in Table 2. There were 5 kinds of workload with 30 individual workloads each, evaluated for 5 load ratios, each using one of 7 ordering policies. This gave 5250 individual schedules produced. The approach described in Section 4.2 was used to evaluate whether or not the hypotheses held.

4.4.1. Fairness

Standard Deviation of SLR. To evaluate the fairness of the ordering policies, the standard deviation of the SLR values was calculated for each schedule produced. These values are displayed in a box-plot, shown in Figure 6. The null hypothesis stated that the P-SLR orderer would produce standard deviations of SLR indistinguishable from the other ordering policies. The null hypothesis was refuted for all orderers except the SRTF orderer. As is visually observed from Figure 6, and confirmed by a repeated measures t-test, the P-SLR and the SRTF orderer produce values for the standard deviation of SLR that are not statistically distinguishable. To further examine this result, we will examine how the scheduling policies affect tasks by their size.

Mean SLR by Decile Figure 7 shows the mean SLR by decile of job execution times at 120% load ratio.

The LRTF orderer prioritises the longest jobs the most, with the lowest decile score for the largest tasks, and penalises the smallest tasks most, with the highest mean SLR score for the smallest tasks. This is exactly what would be expected of the ordering policy.

The Random, Fair Share and FIFO ordering policies all follow a similar profile. This is due to the fact that in these orderers, all tasks will wait in the queue for approximately the same amount of time. Naturally, the penalises the SLR of the shorter tasks more than the larger ones.

The SRTF orderer follows the opposite pattern, prioritising the smallest tasks the most and penalising the largest tasks. Across most of the workload space, the SRTF orderer gives the lowest mean SLR value. However, this crosses over for the 10th decile (the largest jobs), where the highest mean SLR is produced by SRTF. However, because the largest tasks are so large, they are much less sensitive to delays than shorter tasks. In our simulation, the workload were allowed to run to completion after jobs had finished arriving, which meant that every job would eventually finish. In reality, in an overloaded system, this may not be the

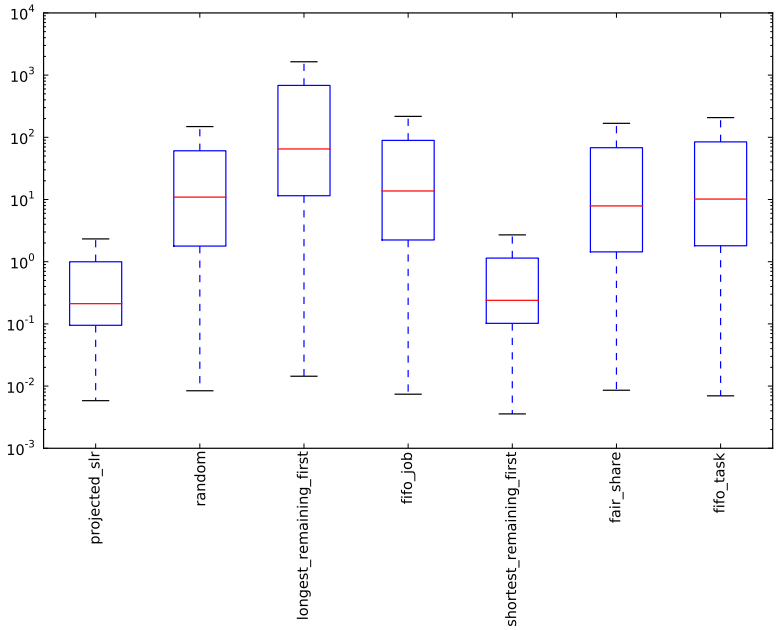


Figure 6: Standard Deviation of SLR by ordering policy

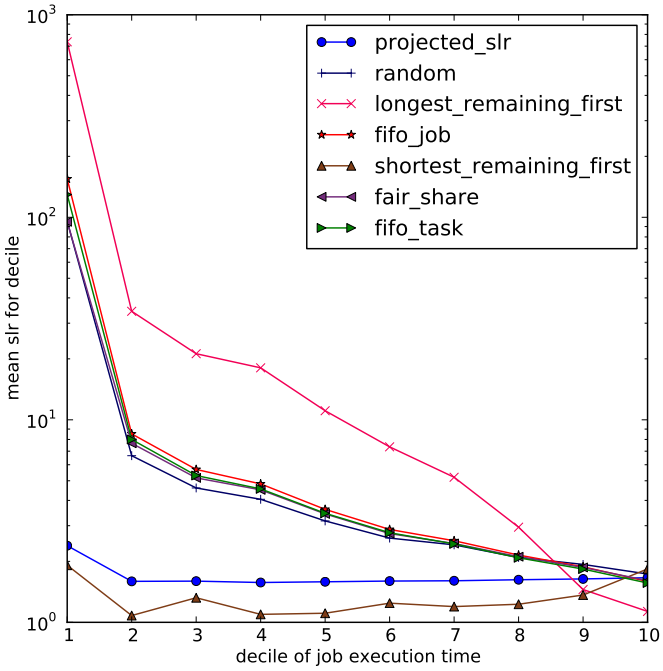


Figure 7: Mean SLR by decile of job execution times, 120% load ratio

case. Because the shortest remaining first scheduler is not starvation-free, the worst-case for the largest jobs may be much worse in reality.

The P-SLR orderer, as intended in its design, shows no bias in terms of SLR across the range of execution times. Because the largest jobs are guaranteed to run, this will have an impact on all of the smaller jobs in the system. However, this penalty is shared out equally across the workload.

The uptick in mean SLR seen in the first decile can be attributed to small jobs arriving when no resources are free in the cluster. Even the delay until the next instant when until some resources become free can therefore cause SLR to increase significantly. As the size of a cluster increases, however, this uptick would be less pronounced for a similar workload, because the expected delay until some processors become free will decrease.

For the fairness metrics, we can see that the P-SLR ordering policy provides a significant improvement in fairness over the LRTF, Fair Share, Random and FIFO-based ordering policies. The P-SLR orderer also delivers a statistically insignificant difference in fairness from the SRTF ordering policy, even while P-SLR offers a guarantee that no job will ever starve.

4.4.2. Responsiveness

Worst-Case SLR. The responsiveness achieved by each ordering policy can be measured by the worst-case SLR for each schedule. The null hypothesis for responsiveness states that worst-case SLR values produced by the P-SLR ordering policy are indistinguishable from those produced by alternative policies. The distributions of worst-case SLR values for each schedule are shown in Figure 8. Statistical significance between the distributions was evaluated using the repeated measures t-test. As with the fairness hypothesis, the null hypothesis is rejected for P-SLR compared to all the ordering policies except SRTF. P-SLR and SRTF give significantly better responsiveness than the other scheduling policies, and are statistically indistinguishable from each other. These ordering policies achieve low worst-case values because they prioritise or give equal treatment to the smaller jobs in the workload, as shown in the previous section. Because the smallest jobs are also the most sensitive to delays, reducing their SLR value is key to achieving the best responsiveness possible.

Mean values of Worst-Case SLR. The ordering policies can also be compared as to how their ability to achieve responsiveness as the load ratio is increased. This is plotted in Figure 9. Throughout most of the range, the longest remaining first orderer has the worst worst-case SLR. This is to be expected, because it prioritises the largest tasks, and the smallest tasks' SLRs suffer proportionately more when they are delayed. At the other end of the scale, SRTF and P-SLR show similar values for the lowest worst-case.

It is not surprising to observe that the random orderer achieves better or similar mean worst-case SLR values across the spectrum of load ratio when compared to the Fair Share and FIFO-based orderers. This is because shorter tasks must always wait the whole length of the queue in FIFO-based ordering policies. This will penalise small tasks heavily, and lead to a high worst-case SLR. The random scheduler, on the other hand, allows some small tasks to 'jump the queue', and hence lower the likelihood that they will have to wait the full duration of the queue before being executed.

It is possible to use the worst-case SLR metric to calculate a relative metric of dominance. Dominance is the number of schedules where the worst-case SLR achieved by P-SLR is less than or equal to that achieved by the alternative orderer. The values of the dominance metric across the load ratio spectrum are shown in Table 3, where values in bold indicate a lack of statistically significant difference between P-SLR and the alternative policies. As with the other metrics, it is found that P-SLR dominates all the other ordering policies except SRTF. In the case of SRTF, the null hypothesis cannot be refuted and therefore P-SLR is statistically indistinguishable, across the load ratio spectrum.

Mean Values of SLR. The dominance metric can also be applied to the mean SLR metric, another responsiveness measure (Table 4, values in bold again indicate a lack of statistically significant difference). As previously observed with the other metrics, P-SLR dominates all the orderers except SRTF. However, at higher load ratios, the null hypothesis is again refuted, showing that there is a significant difference between the performance of P-SLR and SRTF. This is because SRTF achieves better performance for small

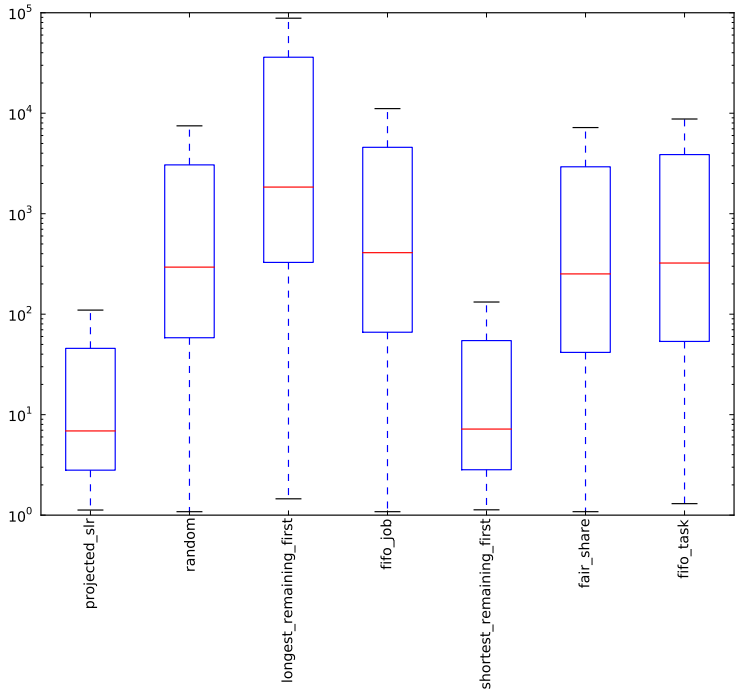


Figure 8: Worst-Case SLR by ordering policy

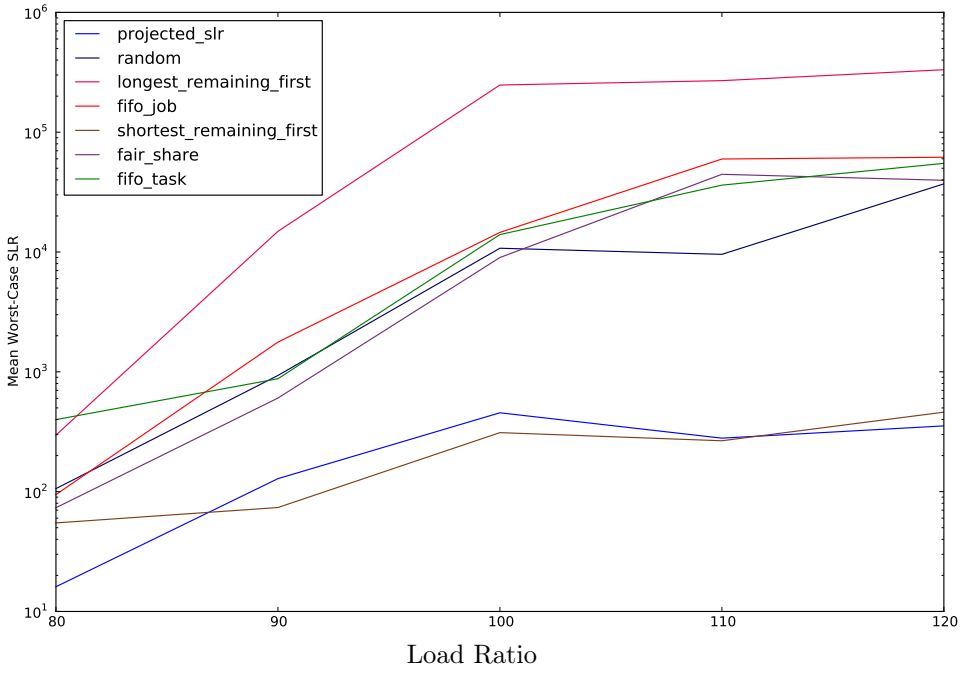


Figure 9: Mean worst-case SLR by load ratio

% Dominated by Projected-SLR	80	90	100	110	120
Longest Remaining Time First	96	100	100	100	100
Shortest Remaining Time First	**54**	**47**	**56**	**58**	**57**
Random	87	93	100	93.3	100
FIFO Task	87	98	100	100	100
FIFO Job	92	94	100	100	100
Fair Share	87	95	98.6	100	99.3

Table 3: Dominance of Projected-SLR orderer over Worst-Case SLRs

% Dominated by Projected-SLR	80	90	100	110	120
Longest Remaining First	97	100	100	100	100
Shortest Remaining First	**59**	**54**	44	26	21
Random	88	97	100	100	100
FIFO Task	91	98.3	100	100	100
FIFO Job	94	97	100	100	100
Fair Share	92	98.6	100	100	99.3

Table 4: Dominance of Projected-SLR orderer over mean SLRs

tasks, which make up the majority of the workload considered. This brings the mean down, and has SRTF dominate P-SLR for mean SLR under high load.

The responsiveness measures, therefore, show that the P-SLR orderer gives more responsive schedules than the Random, LRTF, Fair Share and FIFO-based orderers, by dominating their mean and worst-case SLR values across the load spectrum. The P-SLR achieves worst-case SLR results indistinguishable from the SRTF order, although the SRTF orderer achieves significantly better mean SLR results at high load.

4.4.3. Utilisation

Average Utilisation. Figure 10 shows the average utilisation across the different orderers. In this experiment, the null hypothesis is rejected for all other ordering policies. Statistically, P-SLR has a higher average utilisation than SRTF and a lower utilisation than all other schedulers. However, although this may produce a statistically significant result because of the large sample size, it can be argued that the difference is small, as can be seen from the size of the boxes in Figure 10.

Furthermore, utilisation is calculated based on the workload makespan. In this simulation, where the workload was left to run to completion, the workload makespan is likely to be decided by a single large job that arrived late in the schedule. This is corroborated by the low median values for utilisation shown. These are only low over the whole makespan, because the makespan is significantly extended by large jobs running at the end of the schedule.

We therefore conclude that although the utilisation achieved by the P-SLR scheduler is statistically significantly lower than for the orderers other than SRTF, it is not of a magnitude that is cause for concern. Utilisation can be considered to be effectively equal over the orderers, because the differences between them are so small.

Cumulative Completion. A box-plot showing the cumulative completion values for the different scheduling policies is shown in Figure 11. Although the plots look fairly similar, the P-SLR orderer is statistically significantly better than all other orderers except SRTF, as we have found previously. This is because cumulative completion is linked to responsiveness. If more tasks finish earlier in the schedule, then the schedule will be more responsive, and the cumulative completion metric will be higher. Because the P-SLR and SRTF metrics are indistinguishable in their responsiveness, it would follow that they are indistinguishable in their cumulative completion.

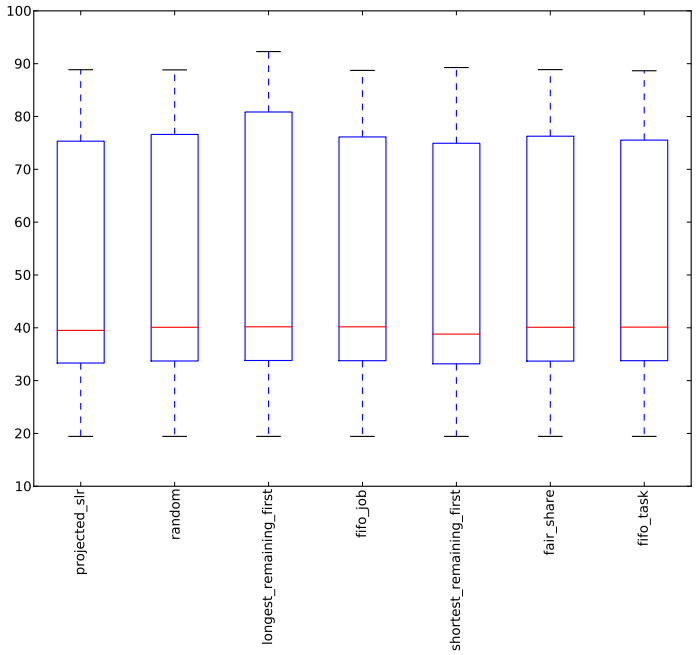


Figure 10: Average Utilisation by Ordering Policy

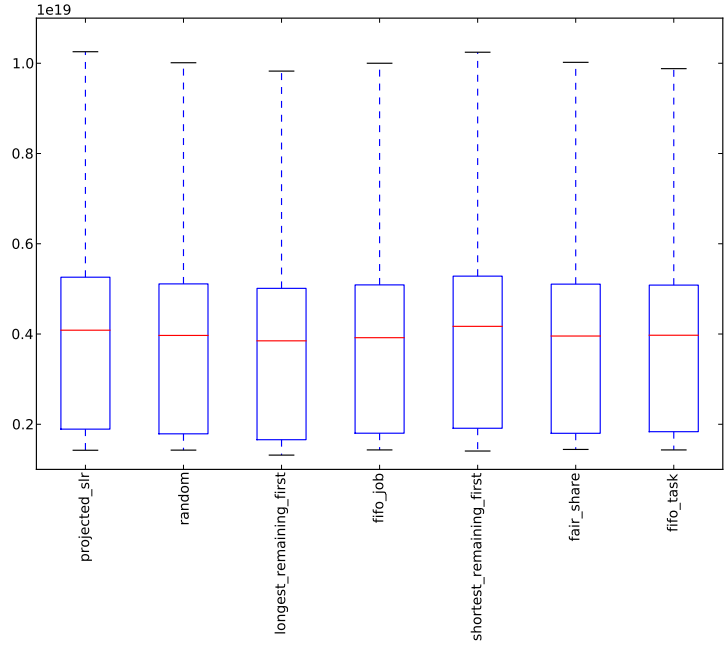


Figure 11: Cumulative Completion by Ordering Policy

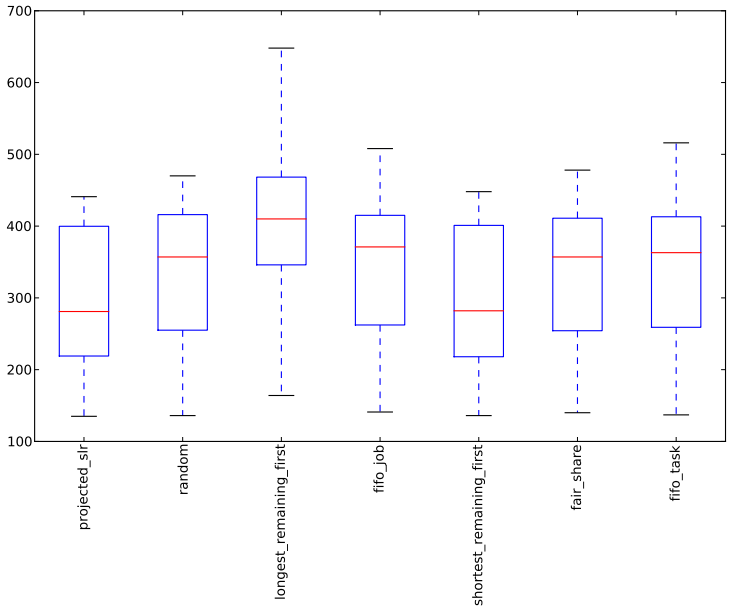


Figure 12: Peak In-Flight by Ordering Policy

Peak In-Flight. The results for the peak in-flight metric are shown in Figure 12. Similarly to the results found for the other metrics, P-SLR has a statistically significantly lower peak in-flight for all other ordering policies except SRTF. This is also to be expected due to the responsiveness findings, because a high level of responsiveness will mean that more tasks are finishing more quickly, and hence there will be fewer in-flight.

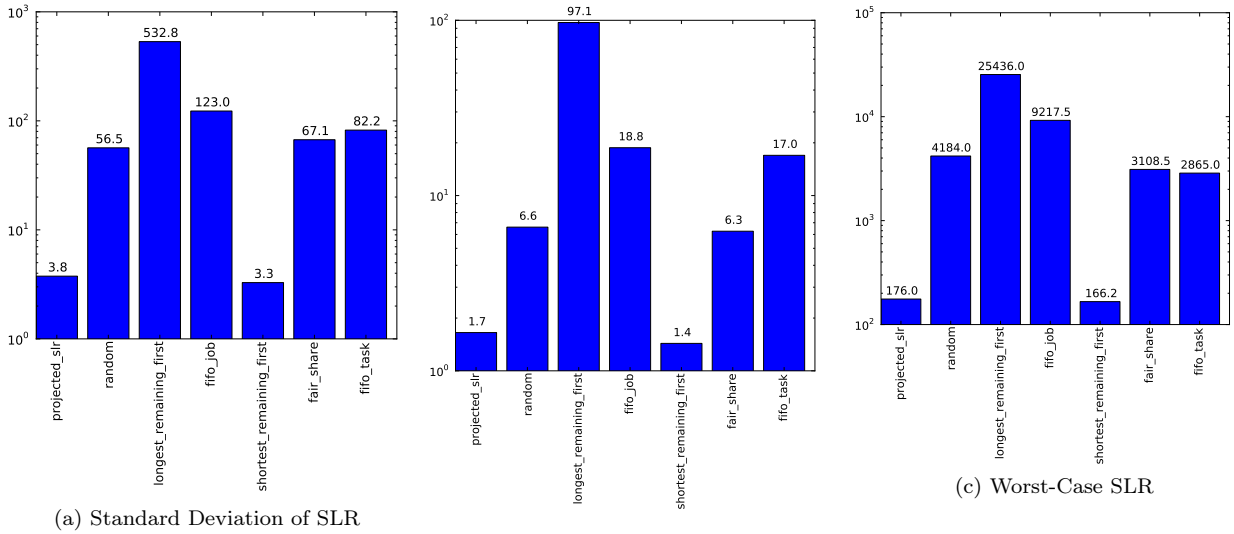
Another interesting feature to notice is to consider the peak in-flight count of the LRTF orderer. From the cores per task presented in the workload parameters above (Table 2), we can see that the average number of cores per task is expected to be just over 10. The median value for peak in-flight jobs given for the LRTF orderer is just over 400. Therefore, at the point in the schedule of peak in-flight, there are more jobs in-flight than there are possible to be servicing at once, given that the platform consists of 4000 cores. This finding reinforces the responsiveness metrics that show the LRTF ordering giving poor responsiveness. LRTF starts a lot of jobs quickly, but takes a long time to finish them, as is shown by the high peak in-flight and the lower responsiveness achieved.

4.4.4. Evaluation Summary

In this evaluation of policies using synthetic workloads, we have shown that the P-SLR ordering policy has significantly improved fairness and responsiveness when compared to the Random, LRTF, Fair Share, FIFO Task and FIFO Job policies. The P-SLR produces fairness and responsiveness results that are statistically indistinguishable from the SRTF ordering policy. However, we would argue that the P-SLR ordering policy is a better choice for a production policy, because it is starvation-free. P-SLR guarantees that all jobs and tasks will eventually run, however large they are. Using SRTF, on the other hand, may lead to the largest jobs starving for resources indefinitely in a system in overload, where the arrival rate of work continually exceeds the ability for the system to service this work.

4.5. Scheduler Evaluation (Industrial)

In this section, the performance of the P-SLR ordering policy will be evaluated using a single workload obtained from the logs obtained in the industrial case study. The platform used for these experiments reflects the platform used in the number, size and connectivity of the clusters. Therefore, the results for the Fair Share policy shown here reflect the values seen in the production system.



(a) Standard Deviation of SLR

(b) Mean SLR

(c) Worst-Case SLR

Figure 13: Functions of metrics over SLR by ordering policy (Industrial Workload)

4.5.1. Fairness

Standard Deviation of SLR. The fairness of the scheduled produced using the industrial workload are shown in Figure 13a, as measured by the standard deviation of their SLR values. It is clear that P-SLR and SRTF show dramatically higher levels of fairness compared to the alternative policies. P-SLR shows a slightly higher standard deviation of SLR, though this difference is small compared to the differences with any of the alternative policies. Furthermore, the benefit of having a guarantee that a schedule will always be starvation-free (given by P-SLR) is likely to outweigh the slight decrease in fairness over SRTF. When also considering SLR across the range of execution times (Figure 14), the slightly lower degree of fairness can be explained by the P-SLR policy having a slightly higher average SLR across the whole workload. Because the shorter jobs are more sensitive to an increase in SLR than the large jobs, this would amplify their differences and hence give a higher standard deviation.

What is worth noting is the strong performance, in terms of fairness, of the random policy. It is slightly fairer than the currently used Fair Share policy and much better than the FIFO and LRTF policies. It can be argued that random ordering could give fair results, but that were equally poor in responsiveness, although this is not the case for reasons outlined below.

Mean SLR over decile of execution time. The pattern for responsiveness when using the industrial workload parallels the patterns seen using the synthetic workloads. The SRTF policy achieves the highest mean responsiveness, although the results given by P-SLR are closely competitive (Figure 13b). Across the deciles of execution time (Figure 14), SRTF consistently outperforms P-SLR, albeit slightly. Interestingly, even for the highest deciles, the mean SLR values are equivalent for SRTF and P-SLR. This is likely due to several reasons. Firstly, the largest jobs are so large that even with a pending time of weeks, when their execution times are in the order of months, their SLR value may still be low. Secondly, load balancing between the clusters may direct shorter jobs to alternative clusters when there are large jobs pending on a given cluster, which may mitigate the likelihood of starvation for the large pending jobs. Thirdly, in the industrial scenario, it is likely that simply through natural variation in the submission rates of work, there will be occasions where the clusters are not fully loaded and therefore the longest jobs can start. Even though these occasions may happen only every few months, this will hardly affect the SLR of the largest jobs, that themselves run for a few months.

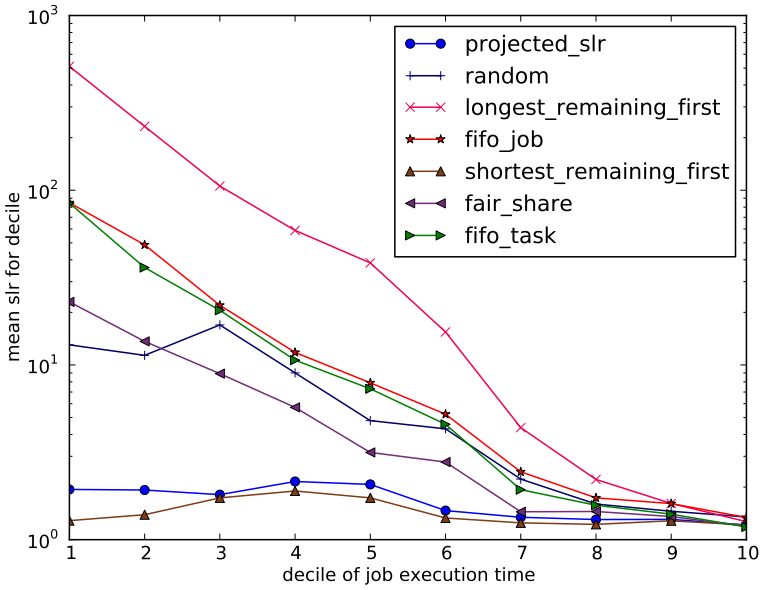


Figure 14: Mean SLR for decile of job execution time (Industrial workload)

It could be argued therefore that SRTF is the most appropriate policy for achieving high responsiveness and fairness for most users most of the time. However, the clusters tend to get busier over time, and procuring a new cluster is a lengthy process. This will lead to it becoming ever more likely that the largest jobs will starve. Furthermore, there are genuine organisational needs of the data, and by using P-SLR, the wait time for these largest jobs will be bounded, which is helpful for organisational planning for when the data is ready. Therefore, the guarantee of non-starvation offered by P-SLR is valuable, and the impact of the slightly higher mean SLR across the workload is so small as to be very likely to be acceptable (especially noting the logarithmic scale on the y-axis of Figure 14).

As expected, the LRTF policy gives the poorest responsiveness of any policy, because it intentionally penalises the shortest jobs to the advantage of the longer jobs. Both of the FIFO policies suffer for responsiveness because of the wide range in execution times between the shortest and longest jobs. The SLR value of a minutes-long job will naturally be very high if it is waiting in the queue behind a month-long job. The Random policy improves slightly on this, because the shortest jobs do have some chance of getting in before the largest jobs.

Most interestingly is that the Fair Share policy seems to be working favourably compared to Random across most of the space of execution times. This suggests that the organisation have crafted their Fair Share table mostly correctly. However, it is poorer than Random for the shortest jobs, which tend to also be those where responsiveness is most highly prized. However, no matter what Fair Share tree is are used, it is not possible for Fair Share to be competitive with P-SLR and SRTF due to the fact that Fair Share does not take into account any information about execution times.

4.5.2. Responsiveness

Worst-Case SLR. During the 10 month duration of the industrial logs, there were some periods of overload. The worst-case SLR results are a useful measure of how well the policies were able to keep up with responsiveness even under such overload as experienced in a real system. The high values for these jobs are most likely to be those jobs with very short execution times, as they are the most sensitive to any changes in pending time. For worst-case SLR, once again P-SLR and SRTF show values of comparable magnitude,

Policy	Average Utilisation	Cumulative Completion	Peak In-Flight
P-SLR	58.64	7.685×10^{18}	489
Random	58.64	7.686×10^{18}	515
LRTF	58.64	7.688×10^{18}	490
FIFO Job	58.64	7.686×10^{18}	543
SRTF	58.64	7.684×10^{18}	439
Fair Share	58.64	7.687×10^{18}	441
FIFO Task	58.64	7.688×10^{18}	407

Table 5: Utilisation Metrics (Industrial Workload)

although SRTF is again slightly ahead for the industrial workload. This is due to its aim in prioritising the shortest jobs that have the most sensitive SLR measurements. LRTF returns the poorest worst-case responsiveness, for much the same reason. FIFO Job does surprisingly poorly, as it is has poorer worst-case responsiveness than Random and FIFO Task. It is to be expected that the worst case for Random would be poor, because of some unlucky short task that has to wait a very long time. For some reason in this particular workload, the multiple waits problem does not cause FIFO Task to be poorer than FIFO Job. Because it is a single worst-case, however, it could well be that it is a single task in a pathological case

4.5.3. Utilisation

The results for the utilisation metrics were so close as to be unhelpful to display graphically, and so have instead been presented in Table 5. The Average Utilisation values were identical because the Workload Makespan values were also identical. This is because of a single long-running job arriving just before the end of the sampling period of jobs, and which kept on running long after everything else in the sample had completed. This is also the reason the average utilisation seems so low - it is not that the clusters were actually that quiet in reality, instead it is because of a significant period where in the simulation, only the last single long-running task was executing. However, both the Average Utilisation and the Cumulative Completion values demonstrate that P-SLR is able to keep utilisation as high as the alternative policies, even while increasing fairness and responsiveness.

The Peak In-Flight values are not identical but none have a huge degree of difference. SRTF has the lowest peak, which is not surprising because it will tend to get short jobs out of the way quickly. In contrast to the comments of the users about this workload, the multiple waits problem does not seem to be manifested statistically here, with the peak in-flight value for FIFO Job being higher than for FIFO Task, rather than vice versa. However, this could be a pathological case where this workload on this platform causes FIFO Job to perform particularly poorly on this metric.

4.5.4. Industrial Evaluation Summary

The results from the industrial evaluation corroborate those from the synthetic workloads. When the P-SLR ordering policy is applied to the workload derived from the trace of an industrial HPC, it gives fairness, responsiveness and utilisation results comparable to that of the best alternative policy - SRTF. However, it does this while still providing a starvation-free guarantee. It is seen in Figure 14 that P-SLR can achieve responsiveness across the range of execution times, just as SRTF can.

5. Conclusions

The collection of metrics is essential for the owners and operators of grid and cloud platforms to ensure good utilisation of their platforms and quality of service for their users. This was motivated with an industrial scenario of familiarity to the authors, and which argues that the perspective of the users necessitates the evaluation of metrics that deal with quality of service. A number of metrics were then presented, grouped into those dealing with Utilisation, Responsiveness and Fairness. It was shown that while utilisation metrics

have traditionally been used to evaluate scheduling policies, they are less suitable in a dynamically-scheduled system such as a grid or a cloud. Instead, responsiveness and fairness metrics are better able to show how a scheduling policy is managing the resources under its control in order to maximise the benefit to users.

The metrics were evaluated as to their ability to give insight into scheduling issues. The Schedule Length Ratio (SLR) metric of [22] was shown to be particularly useful for workloads with dependencies, because it uses the critical path of the job as the performance benchmark to compare against.

The authors compared the ordering decision of list scheduling policies designed to run at the cluster level within a hierarchical grid scheduling scheme. The Projected-Schedule Length Ratio (P-SLR) policy was then developed with the aim of achieving high responsiveness and fairness even under periods of overload, without significantly impacting the cluster utilisation.

Evaluation of these scheduling policies were then performed in simulation. Firstly, using synthetic heterogeneous workloads with a logarithmic distribution of execution times and a selection of dependency patterns. This evaluation took place using a simulated grid comprising a number of heterogeneous clusters with networking delays between them. Secondly, by using a workload extracted from 10-month trace of an industrial grid running over a simulated platform designed to reflect the configuration of the production grid.

The P-SLR scheduler was found to give more responsive and fairer schedules than the Random, Longest Remaining Time First, Fair Share, FIFO Task and FIFO Job ordering policies; without having a major impact on utilisation. The P-SLR orderer achieved responsiveness and fairness performance that was statistically indistinguishable from the Shortest Remaining Time First ordering policy, even though P-SLR is guaranteed to be starvation-free, while SRTF is not.

The authors conclude that the Average, Worst-Case and Standard Deviation of Schedule Length Ratio metrics provide a suitable level of insight for evaluating quality of service from a users' perspective. This is because these capture the users' concerns about responsiveness and fairness while taking into account the structure of dependencies for workloads that contain them. The authors also propose that the Projected-SLR policy is a suitable candidate for production use as a scheduler for HPC systems, due to its ability to achieve good responsiveness, fairness and utilisation and to degrade gracefully under periods of overload.

Future extensions of this work envisaged by the authors include investigating the robustness of the P-SLR scheduler under more varied conditions. This may include the situations where network delays are more significant than considered in this paper, or where the execution times of tasks can only be estimated and not known precisely in advance. The authors also wish to pursue industrialisation of the P-SLR scheduler.

Acknowledgements

We would like to thank the EPSRC (grant number EP/F501374/1) for funding this research through the UK's Large-Scale Complex IT Systems (LSCITS) programme.

References

[1] Albers, S., 1997. Better bounds for online scheduling. In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. STOC '97. ACM, New York, NY, USA, pp. 130–139.
URL `http://doi.acm.org/10.1145/258533.258566`

[2] Albodour, R., James, A., Yaacob, N., 2012. High level qos-driven model for grid applications in a simulated environment. Future Generation Computer Systems 28 (7), 1133 – 1144, special section: Quality of Service in Grid and Cloud Computing.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X11002159`

[3] Bender, M. A., Chakrabarti, S., Muthukrishnan, S., 1998. Flow and stretch metrics for scheduling continuous job streams. In: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms. SODA '98. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 270–279.
URL `http://dl.acm.org/citation.cfm?id=314613.314715`

[4] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., Freund, R. F., June 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing 61, 810–837.
URL http://dl.acm.org/citation.cfm?id=511973.511979

[5] Burkimsher, A., October 2011. Dependency patterns and timing for grid workloads. In: Proceedings of the Fourth York Doctoral Symposium on Computer Science. pp. 25–33.
URL http://www.cs.york.ac.uk/ftpdir/reports/2011/YCS/468/YCS-2011-468.pdf

[6] Carter, B. R., Watson, D. W., Freund, R. F., Keith, E., Mirabile, F., Siegel, H. J., 1998. Generational scheduling for dynamic task management in heterogeneous computing systems. Information Sciences 106 (3-4), 219–236.

[7] Chapin, S. J., 1996. Distributed and multiprocessor scheduling. ACM Computing Surveys 28 (1), 233–235.

[8] Davis, R. I., Burns, A., 2009. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. IEEE International Real-Time Systems Symposium 0, 398–409.

[9] Erdős, P., Rényi, A., 1960. On the evolution of random graphs. In: Publication Of The Mathematical Institute Of The Hungarian Academy Of Sciences. pp. 17–61.

[10] Expósito, R. R., Taboada, G. L., Ramos, S., no, J. T., Doallo, R., 2013. Performance analysis of hpc applications in the cloud. Future Generation Computer Systems 29 (1), 218 – 229.
URL http://www.sciencedirect.com/science/article/pii/S0167739X12001458

[11] Garey, M. R., Johnson, D. S., 1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

[12] Graham, R. L., 1969. Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 416–429.

[13] Iverson, M. A., Özgüner, F., 1999. Hierarchical, competitive scheduling of multiple dags in a dynamic heterogeneous environment. Distributed Systems Engineering 6 (3), 112.
URL http://stacks.iop.org/0967-1846/6/i=3/a=303

[14] Khan, A., Mccreary, C., Jones, M., August 1994. A comparison of multiprocessor scheduling heuristics. In: International Conference on Parallel Processing, 1994. Vol. 2. pp. 243 –250.

[15] Kwok, Y.-K., Ahmad, I., 1999. Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing 59 (3), 381–422.

[16] Maheswaran, M., Braun, T. D., Siegel, H. J., 1999. Heterogeneous distributed computing. In: In Encyclopedia of Electrical and Electronics Engineering. John Wiley, pp. 679–690.

[17] McCreary, C., Khan, A., Thompson, J., McArdle, M., April 1994. A comparison of heuristics for scheduling dags on multiprocessors. In: Proceedings of the Eighth International Parallel Processing Symposium. pp. 446 –451.

[18] Nascimento, A. P., Boeres, C., Rebello, V. E. F., 2008. Dynamic self-scheduling for parallel applications with task dependencies. In: Proceedings of the 6th international workshop on Middleware for grid computing. MGC '08. ACM, New York, NY, USA, pp. 1:1–1:6.
URL http://doi.acm.org/10.1145/1462704.1462705

[19] Navaridas, J., Miguel-Alonso, J., Ridruejo, F. J., Denzel, W., 2010. Reducing complexity in tree-like computer interconnection networks. Parallel Computing 36 (2-3), 71 – 85.
URL http://www.sciencedirect.com/science/article/B6V12-4Y1MRPG-2/2/27aa5554ff69bfd5adb984e77d6b2283

[20] Oracle Corporation, 2010. N1 grid engine 6 administration guide - configuring the share-based policy. Online.
URL http://docs.oracle.com/cd/E19080-01/n1.grid.eng6/817-5677/i999588/index.html

[21] Platform Computing Corporation, 2008. Fairshare scheduling. Online.
URL http://www.cisl.ucar.edu/docs/LSF/7.0.3/admin/fairshare.html#wp215541

[22] Topcuoglu, H., Hariri, S., Wu, M.-Y., March 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13 (3), 260 –274.