Scheduling HPC workflows for responsiveness and fairness with networking delays and inaccurate estimates of execution times

Andrew Burkimsher, Iain Bate, Leandro Soares Indrusiak

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

Abstract. High-Performance Computing systems (HPCs) have grown in popularity in recent years, especially in the form of Grid and Cloud platforms. These platforms may be subject to periods of overload. In our previous research, we found that the Projected-SLR list scheduling policy provides responsiveness and a starvation-free scheduling guarantee in a realistic HPC scenario. This paper extends the previous work to consider networking delays in the platform model and inaccurate estimates of execution times in the application model. P-SLR is shown to be competitive with the best alternative scheduling policies in the presence of network costs (up to 400% computation time) and where execution time estimate inaccuracies are within generous error bounds (<1000%) while still giving starvation-free behaviour.

1 Introduction

High-Performance Computing systems (*HPCs*) made up of a large number of parallel processors have become increasingly popular. To increase the available computing power, geographically-distributed networks of such clusters have been created, and these are known as *grids* [1]. These grids are often heterogeneous, with machines of varying capacity and architecture. The geographic distribution of the clusters in the grid gives rise to network delays when transferring data.

The pieces of work run on grids are rarely run in isolation, but instead form part of workflows, with dependencies between different computational tasks [2,3]. Each self-contained workflow is known as a *job*, made up of non-pre-emptible multicore *tasks* with dependencies.

Organisations that own grid or HPC capacity as well as cloud providers have an interest in providing Quality of Service (QoS) to their users. A particularly important aspect of QoS for many users is responsiveness. It is almost inevitable for grids and clouds to experience significant variations in demand, which can lead to transient periods of overload where some jobs have to wait. Industrial users interviewed by the authors indicated their desire for response times of jobs to be proportionate to the jobs' execution times. Furthermore, users desire fair treatment of their jobs. An example of a particularly unfair situation is if some jobs experience starvation (unbounded waiting time) under overload.

Previous work on fair scheduling for workflows with dependencies has been performed for offline [4] and batching [5] schedulers. These are ineffective when there is a wide variation in runtimes [6,7,8] because the response times required of the smallest tasks (hours) are orders of magnitude smaller than the execution times of the largest tasks (months), so no batch size will suit both. Effective prioritisation by the scheduler is required to keep the system responsive for the smallest tasks but avoid starvation for the largest ones.

In previous work [6], the authors developed an online list scheduling policy for dependent tasks called P-SLR that achieves responsiveness for small tasks and also provides a guarantee that no task will ever starve. P-SLR gave statistically indistinguishable responsiveness and fairness when compared to the best alternative scheduler, Shortest Remaining Time First (SRTF), even though SRTF is not starvation-free.

The P-SLR scheduler requires an estimate of the task execution time. The previous work [6] assumed that these times were known exactly. However, it is an ongoing field of research to accurately predict task execution times before they run, and they will always have their limitations [9]. Although users can provide hints about their task execution times, these are also far from accurate [10]. This paper describes how the models of the original work were extended to include these. The performance of the P-SLR scheduler is evaluated as to the impact of these inaccurate estimates.

Network delays also have an effect on scheduling. These can be described using the communication to computation ratio (CCR) value [11]. In the original analysis, only a single value of CCR was considered, using a network with a single central router. We extend the network model to a hierarchical architecture, and investigate the impact of changing CCR on the responsiveness and fairness of the P-SLR scheduling policy.

The context and models which define the scenario considered are presented in section 2. The considerations of measuring responsiveness and fairness, along with the P-SLR scheduling policy are defined in Section 3. Section 4 will describe the experimental method used to evaluate P-SLR against the alternative scheduling policies, and Section 5 will present the results of this evaluation.

2 Models

2.1 Application Model

A 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 is a job, denoted J^k. A set of jobs will be known as a workload W. Dependencies will be in the form of a directed acyclic graph (DAG), following [2]. Each task has an associated *architecture*, which defines which resources in the grid are available for it to execute on. The following terms define the attributes of tasks and jobs.

- 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: $T^i_{\texttt{finish}} = T^i_{\texttt{start}} + T^i_{\texttt{exec}}$
– Task dependendents/successors: $T^i_{\texttt{succ}}$
– Task upward rank: $T^i_{\texttt{R}} = T^i_{\texttt{exec}} + \max(T^j_{\texttt{R}}) \bullet \forall T^j \in T^i_{\texttt{succ}}$
– 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: $J^k_{\texttt{response}} = J^k_{\texttt{finish}} - J^k_{\texttt{arrive}}$
– Job total execution time: $J^k_{\texttt{exec}} = \sum\left(T^i_{\texttt{exec}} \times T^i_{\texttt{cores}}\right) \bullet \forall T^i \in J^k$
– Job critical path: $J^k_{\texttt{CP}} = \max\left(T^i_R\right) \bullet \forall T^i \in J^k$

2.2 Inaccurate Estimates of Execution Times

In a realistic system, it is assumed that an estimate of execution time, albeit inaccurate, will be available from the user or from an automated job profiler. In simulation, however, the exact execution times are known in advance, so inaccuracies need to be introduced into the model. In this work, two possible ways are considered to convert exact execution times ($e_{\texttt{orig}}$) into inaccurate estimates, in order to evaluate the impact these inaccuracies have on the schedule quality:

Normal Error

This creates an estimate by sampling a normal distribution, shown in Equation 1, with a parameter N to vary the standard deviation, and hence the inaccuracy, of the estimate. The evaluation considers a wide range of values for N, between 1 % and 10^8 %.

$$e_{\texttt{est}} = \left\lceil \texttt{normal}(\mu = \mathrm{e}_{\texttt{orig}}, \sigma = e_{\texttt{orig}} \times \frac{N}{100}) \right\rceil \tag{1}$$

Log Rounding

This form of inaccuracy (Equation 2) reflects the expertise of users in knowing whether a job will take minutes, hours or days, but without much greater precision. M is the base of the logarithm used, with smaller values giving larger numbers of possible classes. The evaluation considers bases between 1 (no rounding) and 10^7 (all are in the same class).

$$e_{\texttt{est}} = M^{\lceil \log_M(e_{\texttt{orig}}) \rceil} \tag{2}$$

2.3 Platform Model

The resources in the grid are grouped into homogeneous clusters. These are connected together in a tree structure with a router at each node and a cluster at each leaf. Network delays are only considered between clusters, as delays inside a cluster are assumed to be negligible.

Jobs are submitted to the root of the tree and are randomly cascaded down the tree until a cluster is reached. Tasks from a single job that share the same architecture are kept together and are allocated to the same cluster, to avoid unnecessary network costs. The detailed scheduling decisions are then made by a list scheduler executing on the clusters.

2.4 Network Delay Model

The network model is considered to be that of a thin tree [12]. This is where nodes at lower levels of the tree have greater communication speed between them than nodes higher in the tree. This reflects what is seen in geographically distributed networks, where nodes further apart tend to have slower connections. This can approximate a real network, because all fully connected networks possess a spanning tree [13].

The aim of this network model is to provide an acceptable model to investigate the effects of network delays on scheduling while adding minimal computational overhead. The network speed is calculated by using the fact that the network is tree structured. Therefore, any two clusters will share a common parent node, and the number of nodes in between a cluster and the common parent is measured in levels. The speed equation takes a parameter p to vary how much slower the higher levels of the network become.

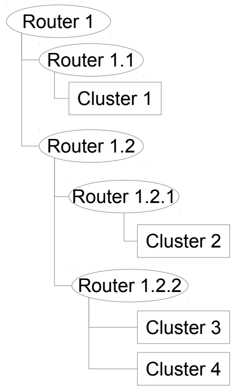


Fig. 1: Thin Tree Network Diagram

$$N_{\texttt{speed}} = (\texttt{max\_levels\_to\_common}\,(C_1, C_2))^p \tag{3}$$

To find the data volume to transfer, the communication to computation ratio parameter (CCR) is used [11], along with the execution time of the task $T^i_{\texttt{exec}}$, as shown in Equation 4.

$$T^i_{\texttt{data}} = \frac{T^i_{\texttt{exec}}}{CCR} * (1 - CCR) \tag{4}$$

The time taken to transfer data between two tasks is determined by dividing the data volume required by the speed of the network between them.

$$T^i_{\texttt{net\_delay}} = \frac{T^i_{\texttt{data}}}{N_{\texttt{speed}}} \tag{5}$$

3 Metrics and the P-SLR Scheduler

In order to evaluate the responsiveness and fairness achieved by scheduling policies, appropriate metrics are required. We suggested in [6] that the most informative metric for measuring responsiveness is the *Schedule Length Ratio* (SLR) [3], when applied to each job in a workload. The *critical path* of a job is the longest path through the job's dependency graph and defines the minimum time

Algorithm 1 Projected SLR ordering algorithm

$\texttt{projected\_slr}(T^i, J^k, \texttt{curr\_time}, Q) =$

$$\frac{\left(T_R^i + \texttt{curr\_time} + 1\right) - J_{\texttt{arrive}}^k}{J_{\texttt{CP}}^k} + \left\lfloor \frac{\texttt{curr\_time} - J_{\texttt{arrive}}^k}{\max\left(J_{\texttt{CP}}^n \bullet J^n \in Q\right)} \right\rfloor^2$$

that the job can be executed in even if the number of processors was unbounded [14]. The SLR of a job is its response time relative to the length of its critical path.

We proposed using the distribution of SLR values to measure fairness [6], of which three kinds can be described (Figure 2). Class 1 is where the responsiveness of longer jobs is prioritised over that of short jobs, with Class 2 being the opposite case. Class 3 is where there is equal prioritisation of responsiveness with respect to execution times.

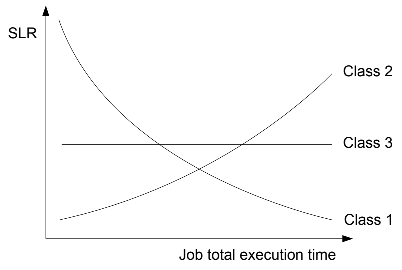


Fig. 2: Classes of prioritisation by execution time

The common scheduling policy First In First Out (FIFO) falls into Class 1 because on average, each job will wait in the queue for the same length of time. This waiting time is proportionately larger relative to execution time for smaller tasks, penalising the SLR of short-running jobs. This pattern is true for any policy not considering execution times. The Longest Remaining Time First (LRTF) scheduler also falls into Class 1. The Shortest Remaining Time First (SRTF) scheduler is of Class 2. The authors designed the P-SLR scheduler to exhibit Class 3 behaviour, and demonstrated this in [6].

In order to make use of execution time estimates, LRTF, SRTF and P-SLR use the concept of *Upward Rank* introduced by [3]. 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. LRTF and SRTF sort the list of tasks by decreasing and increasing Upward Rank, respectively. These policies can suffer from starvation under overload, because the shortest (LRTF) or longest (SRTF) tasks may never reach the head of the queue (Q).

P-SLR uses the upward rank to calculate a forward projection of the job finish time and hence SLR if the considered task was run immediately (Algorithm 1). The task where the P-SLR is largest (is most 'late') is run first, letting small jobs 'jump' the queue as their SLRs are more sensitive to waiting time. This is distinct from the approach used by [5] which uses the downward rank (looks backward) to calculate a partial value for SLR based on the tasks that have already completed.

P-SLR is starvation-free because the projected SLR rises for all jobs as they wait, which means all jobs will eventually run, as long as overloads are transient. In the case of extreme overload where the work queue continually grows unboundedly, the waiting time term (the second part of the equation in Algorithm 1) comes to dominate, reverting the ordering to that of FIFO, thus avoiding starvation in all cases.

4 Evaluation method

The evaluation method will seek to investigate two principal hypotheses:

1: Projected-SLR delivers better responsiveness and fairness than schedulers which do not use execution time estimates, even when the estimate inaccuracy is significant. P-SLR is competitive with scheduling policies that do make use of execution time estimates.

2: Projected-SLR delivers competitive responsiveness and fairness metrics independent of communication to computation ratios.

4.1 Simulation Details

The evaluation will be run using the simulation framework developed and validated in [6] which implements the models and extensions described above. The platform used is the one shown in Figure 1, with each cluster having 400 cores, and Cluster 1,3,4 being of architecture *Kind1*, whereas Cluster 2 is of architecture *Kind2*. These values have been chosen to reflect those we observed in industry. Because there are elements of randomness in the allocation and in the workloads used, numerous simulation trials will be run for each experiment.

Responsiveness will be measured using the median value of the worst-case SLRs observed in each trial. The worst-case SLR is used because of the desire for high responsiveness to be achieved for all users. Using the median value instead of the mean will prevent any truly pathological cases from biasing the results.

Fairness will be measured using the median of the Gini Coefficients [15] calculated for the SLRs in each trial. The Gini Coefficient (GC) is a measure of the inequality of resources allocated to a given population. In this instance, it is the allocation of responsiveness to jobs by the scheduler. The GC takes a value between 0 (completely fair) and 1 (completely unfair).

Statistical significance will be tested using a repeated measures t-test because the workloads are the same, meaning the job SLRs can be directly compared. The threshold for statistical significance is set at 5%.

4.2 Scheduling Policies

Several common policies will be used as a basis for comparison with P-SLR. The random ordering policy simply chooses a random task to run. While it does not guarantee to be starvation-free, the probability of a task starving forever tends towards zero as time passes. The FIFO Task policy runs tasks in the order that they become ready, which is starvation-free. FIFO Job is an alternative to FIFO Task that runs tasks in the order that their jobs arrived on the HPC . This avoids a problem with FIFO Task where tasks that have just become ready are added to the tail of the queue, which means that jobs with many levels of dependencies can end up waiting the length of the queue multiple times.

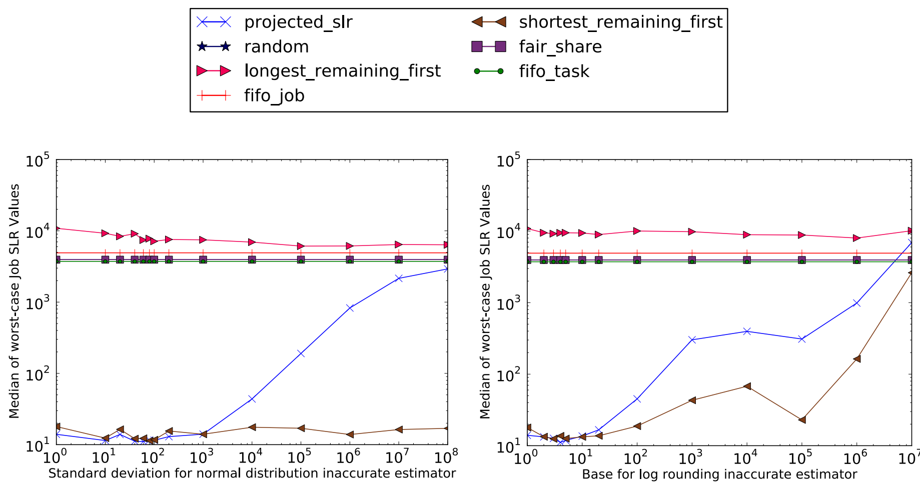
The Fair Share ordering policy [16] is based on the user who submitted a job. Each user is a member of a group, which has a certain share of the resources of the HPC. The share of these groups is organised in a tree. The priority of tasks is not static, but depends on the instantaneous number of resources already being consumed by the user and their parent groups divided by their allocated share.

Simulation of an overload situation is necessary so that some jobs will have to wait, and the ability of the schedulers to keep responsiveness and fairness high can be compared. The overload rate can be defined as a percentage of the arrival rate of jobs compared to the maximum processing rate achievable, and a figure of 120% is used.

5 Results

5.1 Inaccurate Execution Times

For all the results with inaccurate execution times, the Random, FIFO Task, FIFO Job and FairShare policies are not affected by the inaccurate estimates, because they do not make use of these estimates.



(a) Normally distributed inaccuracies (b) Log rounding inaccuracies

Fig. 3: Responsiveness

Responsiveness With normally distributed inaccuracies (Figure 3a), the P-SLR policy dominates by having the lowest worst-case SLR values until the standard deviation is 1000% of the value of the exact time. It is reasonable to assume that virtually all real-world estimates will have ranges less than 1000%.

The difference between P-SLR and SRTF in this range is not statistically significant, which shows the strength of the P-SLR policy as it adds the guarantee of non-starvation. The divergence after 1000% is due to this guarantee because SRTF is letting the largest tasks starve. The largest tasks have SLRs which are least sensitive to waiting time, keeping the worst-case SLR fairly low.

Once the estimation error gets sufficiently large, the estimates become effectively random. Therefore, the worst-case SLR of the P-SLR orderer rises to similar levels as the schedulers that do not make use of execution time estimates.

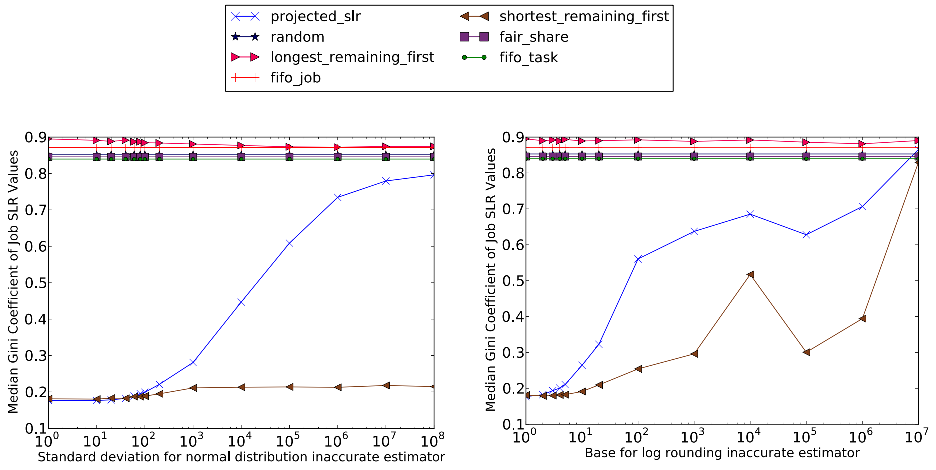
Similar results are apparent where estimates are log-rounded (Figure 3b). Where execution times are rounded to the nearest power of 10 or below, P-SLR dominates the worst-case SLR values, although it is not statistically distinguishable from SRTF. Still, it is to be expected that users could give a good indication of their job taking closer to 1, 10, 100, etc., minutes.

As the estimates get yet more coarse above a base of 10, SRTF provides better worst-case responsiveness than P-SLR. This is to be expected, because inaccurate estimates move the behaviour of schedulers closer to Class 1 behaviour. As P-SLR with accurate estimates exhibits Class 3 behaviour, any perturbation to this will make it tend towards Class 1 behaviour. Whereas for SRTF, because it shows Class 2 behaviour, perturbations will initially make its behaviour more like Class 3, although eventually it too will exhibit Class 1.

The LRTF orderer, as expected, shows poorer worst-case responsiveness than any of the policies that do not consider execution time. This is because it makes the smallest tasks starve, and these tasks are the ones whose SLR is most sensitive to waiting time. LRTF is useful, though, because it gives an upper bound on how poor responsiveness can get because it shows the most extreme Class 1 style behaviour.

These results show that up to a threshold value of 10^3, the P-SLR and SRTF policies have statistically insignificant differences in responsiveness. Responsiveness for P-SLR approaches that of the schedulers that do not include execution time estimates when the error for either normal standard deviation percentage and log rounding is around 10^7. These values are far above the maximum levels of inaccuracy of around 100% found by [10]. This would suggest that in reality, the P-SLR scheduler could be considered most favourable for practical scheduling, because it gives a guarantee of non-starvation, unlike SRTF, and leads to an improvement in responsiveness performance over that of schedulers that do not consider execution time estimates.

Fairness As with the results for responsiveness, the fairness results for normally distributed error (Figure 4a) are dominated by P-SLR at the lowest values, although they are also statistically indistinguishable from SRTF up to a threshold of 100%. This is to be expected, as P-SLR is designed to show Class 3 behaviour,



(a) Normally distributed inaccuracies (b) Log rounding inaccuracies

Fig. 4: Fairness

which emphasises fairness. Above this threshold, P-SLR exhibits progressively more Class 1-like behaviour, as poor estimates for small tasks cause their responsiveness to fall. SRTF causes the largest jobs to starve, but because their SLRs are less sensitive to waiting time, the SLR distribution remains closer to Class 3.

The normally-distributed inaccurate estimator was not able to introduce sufficient error below a standard deviation percentage of 10^8 to cause significant impact on the fairness of the SRTF policy. If the estimation errors are normally distributed, therefore, SRTF may provide better fairness than P-SLR when the standard deviation of the errors is above 100%.

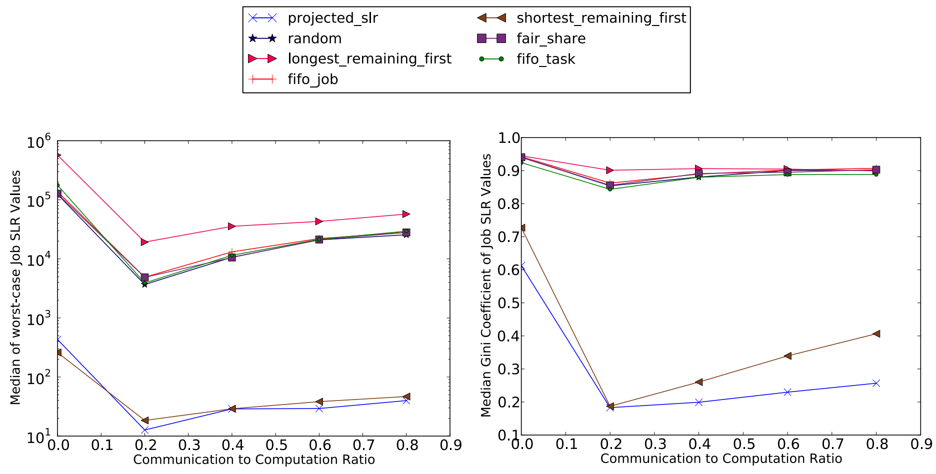
With the log rounding estimator (Figure 4b), other than the case where there was no inaccuracy, the SRTF orderer was statistically significantly more fair, according to the Gini Coefficients, than for P-SLR. As before, this is due to the SRTF causing the largest jobs to starve, but this not having a large effect on those jobs' SLR values. P-SLR immediately starts to exhibit Class 3 behaviour in the presence of inaccurate estimates, whereas SRTF moves from Class 2, then to Class 3, before eventually showing Class 1 at a rounding power of 10^7 .

The LRTF policy shows the worst-case unfairness, as it is the most extreme example of Class 1 behaviour. The bound on how unfair it makes things improve as estimates get worse, because it is not as able to achieve the worst case.

The fairness results show that for small inaccuracies in execution time estimates, P-SLR and SRTF show similar results. However, for larger inaccuracies, SRTF gives fairer results as it shows more of a Class 3 behaviour profile, although this is due to the largest jobs being starved of resources.

Hypothesis 1 stated that P-SLR would deliver better responsiveness and fairness than schedulers that do not use execution times, even when the estimate accuracy is significant. This has been shown to be the case, with better responsiveness and fairness when the standard deviation inaccuracy percentage is less than 10^7 and when the log rounding base is less than 10^8, all extremely high levels of inaccuracy. P-SLR has been shown to be competitive with SRTF in responsiveness up to a threshold inaccuracy of 10 times the value of the original estimate. In fairness, P-SLR is competitive at small inaccuracies, but SRTF dominates above this, refuting a part of the hypothesis. It is then a tradeoff for a grid owner to decide whether, if estimates of execution time have large inaccuracies, absolute fairness (SRTF) or an absence of starvation (P-SLR) is more important.

5.2 Networking Delays



(a) Responsiveness (b) Fairness

Fig. 5: Network Delays

Responsiveness A pronounced feature (Figure 5a) is that there is an improvement in worst-case responsiveness when network costs become present at a CCR of 0.2. This is because any network costs will increase the length of the critical path, which means CPU resources are no longer the single bottleneck.

Throughout the range of network delays examined, P-SLR and SRTF showed similar levels of responsiveness. SRTF was slightly better when there were no network delays, but P-SLR was slightly better when there were delays present. However, P-SLR and SRTF were not statistically significantly different.

The LRTF policy again shows the worst case bound of responsiveness because it tends to starve the smallest tasks.

Fairness The results in Figure 5b also show greater fairness in the presence of network delays, because of the improvements in overall responsiveness. However, in this case, P-SLR is stasticically significantly more fair than SRTF, except where the CCR takes a value of 0.2. As CCR is increased, the unfairness increases more slowly for P-SLR than for SRTF. This is because although their worst case values are similar (Figure 5a), P-SLR shows more Class 3 behaviour, giving a better balance of SLR values overall.

The LRTF scheduler is the least fair at low values of CCR, but converges to a similar level of fairness as those schedulers that do not consider execution times at higher network costs. All the schedulers other than SRTF are very unfair across the space of network delays when compared to P-SLR.

Hypothesis two considered whether P-SLR delivers competitive responsiveness and fairness across the range of CCRs. P-SLR dominated all schedulers other than SRTF in responsiveness, although it was statistically indistinguishable from SRTF. In fairness, it dominated all other schedulers, except for SRTF where CCR was 0.2. As network costs increased, the rate of decrease in fairness was lower for P-SLR than for SRTF. This confirms this hypothesis, showing that across the space of network costs, P-SLR provides equal or better responsiveness and fairness than the best alternative scheduler, SRTF, but does so while in addition providing a guarantee that no job will ever starve.

6 Conclusion

This paper revisited the work of [6], and investigated the robustness of the P-SLR scheduling policy in the presence of networking delays and inaccurate execution times.

The responsiveness and fairness performance of the P-SLR scheduler was found to be robust to network delays. P-SLR provides equal or better responsiveness (measured by worst-case SLR) and fairness (measured by the Gini Coefficient of SLRs) in the presence of network delays than the best alternative scheduler, SRTF, but does so while in addition providing a guarantee that no job will ever starve.

The responsiveness performance of P-SLR was found to be robust below a certain threshold of execution time inaccuracy. This threshold was 10 times the original execution time of the task. Above this threshold, SRTF was able to provide better responsiveness. P-SLR was not able to give the best fairness compared to SRTF once any significant estimation inaccuracies were present, because SRTF is better at keeping SLRs low for small tasks whose SLRs are more sensitive to longer waiting times. However, P-SLR still dominated all other alternative policies, showing that where estimates of execution time are available, it can make good use of them, even where the inaccuracies are large.

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. Albodour, R., James, A., Yaacob, N.: High level QoS-driven model for grid applications in a simulated environment. Future Generation Computer Systems **28**(7) (2012) 1133 – 1144
2. Maheswaran, M., Braun, T.D., Siegel, H.J.: Heterogeneous distributed computing. In: In Encyclopedia of Electrical and Electronics Engineering, John Wiley (1999) 679–690
3. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems **13**(3) (March 2002) 260 –274
4. Zhao, H., Sakellariou, R.: Scheduling multiple dags onto heterogeneous systems. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. (2006) 14 pp.–
5. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., González-García, J., Röblitz, T., Ramírez-Alcaraz, J.: Multiple workflow scheduling strategies with user run time estimates on a grid. Journal of Grid Computing **10**(2) (2012) 325–346
6. Burkimsher, A., Bate, I., Indrusiak, L.S.: 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. Future Generation Computer Systems (In press. Published online 27 Dec 2012) DOI: http://dx.doi.org/10.1016/j.future.2012.12.005
7. Chiang, S.H., Vernon, M.K.: Characteristics of a large shared memory production workload. In: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing. JSSPP '01, London, UK, UK, Springer-Verlag (2001) 159–187
8. Feitelson, D.G., Nitzberg, B.: Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860. In: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing. IPPS '95, London, UK, UK, Springer-Verlag (1995) 337–360
9. Sonmez, O., Yigitbasi, N., Iosup, A., Epema, D.: Trace-based evaluation of job runtime and queue wait time predictions in grids. In: Proceedings of the 18th ACM international symposium on High performance distributed computing. HPDC '09, New York, NY, USA, ACM (2009) 111–120
10. Bailey Lee, C., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? In: Proceedings of the 10th international conference on Job Scheduling Strategies for Parallel Processing. JSSPP'04, Berlin, Heidelberg, Springer-Verlag (2005) 253–263
11. Schoneveld, A., de Ronde, J.F., Sloot, P.M.A.: On the complexity of task allocation. Complex. **3**(2) (1997) 52–60
12. Navaridas, J., Miguel-Alonso, J., Ridruejo, F.J., Denzel, W.: Reducing complexity in tree-like computer interconnection networks. Parallel Computing **36**(2-3) (2010) 71 – 85
13. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction To Algorithms. MIT Press (2001)
14. Kwok, Y.K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing **59**(3) (1999) 381–422
15. Litchfield, J.A.: Inequality methods and tools. In: School of Economics. (1999)
16. Platform Computing Corporation: Fairshare scheduling. http://www.cisl.ucar.edu/docs/LSF/7.0.3/admin/fairshare.html (2008)