Stochastic Workflow Scheduling with QoS Guarantees in Grid Computing Environments

Ali Afzal, John Darlington, A. Stephen McGough
Department of Computing
Imperial College London
South Kensington Campus
London SW7 2AZ, UK
{aa202,jd,asm}@doc.ic.ac.uk

Abstract

Grid computing infrastructures embody a cost-effective computing paradigm that virtualises heterogenous system resources to meet the dynamic needs of critical business and scientific applications. These applications range from batch processes and long-running tasks to more real-time and even transactional applications. Grid schedulers aim to make efficient use of Grid resources in a cost-effective way, while satisfying the Quality-of-Service requirements of the applications. Scheduling in such a large-scale, dynamic and distributed environment is a complex undertaking. In this paper, we propose an approach to Grid scheduling which abstracts over the details of individual applications and aims to provide a globally optimal schedule, while having the ability to dynamically adjust to varying workload demands using various capacity planning techniques. Our model places particular emphasis on the stochastic and unpredictable nature of the Grid, leading to a more accurate reflection of the state of the Grid and hence more efficient and accurate scheduling decisions.

1 Introduction and Related Work

Grid schedulers have historically tended to focus on the scheduling and optimisation of single applications [1, 2, 3, 4, 5]. Given that these applications will have potentially conflicting requirements and the scheduler typically does not forecast future workloads and does not take the requirements of future jobs into account, it is unlikely that the global schedule will be optimal. Most existing work on Grid scheduling considers each application as it arrives, thus implicitly giving preference to the jobs that arrived first. Therefore the time when an application is submitted heavily influences the schedule: the resources available, the jobs ahead in the queue, advance reservations etc, all determine the state of the Grid at the time the application is scheduled, and hence the resulting schedule. This is often not the required behaviour. Moreover, most approaches fail to take account of the effects of heavy Grid workloads, which would be expected to swamp the scheduler, create conflicts between the various workflows and ultimately lead to the rejection of incoming jobs due to the inability to satisfy their QoS requirements.

Furthermore, performance prediction plays a pivotal role in most of the existing Grid schedulers. Performance prediction algorithms use a wide variety of techniques, from analytical modelling to statistical analysis of empirical and historical data, to estimate the task execution time on every Grid resource [6, 7, 8]. The accuracy of these performance prediction algorithms varies wildly under varying conditions, getting progressively worse if the performance characteristics of the task exhibit a degree of Data-dependence[9]. Hence, in most cases, the resulting schedule can only be considered to be best-effort, as deadlines, and hence Quality-of-Service in general, cannot be guaranteed. In such cases, some schedulers aim to schedule using conservative estimates of the performance of tasks [10, 11]. The scheduler then negotiates for the creation of advance reservations on Grid Resources so as to allow exclusive, uninterrupted access to the workflow in question. In cases where schedulers have aimed to take stochastic behaviour into consideration, models have typically been simplified in terms of normal distributions or the optimisation of rudimentary characteristics such as the mean service rates, without regard to the overall Quality-of-Service [10, 12, 13, 14].

Furthermore, in commercial Grids [15, 16], the job of the scheduler is extended to include that of the resource broker. In such scenarios, it is highly desirable to minimise the cost of resource usage on behalf of the Grid users. This applies regardless of whether the resources need to be procured from outside the enterprise Grid setting. Even if the organization-owned resources are capable of handling the workload, it may be desirable to minimise the number of resources allocated for enterprise use so that the surplus resources can be auctioned off for external use, generating revenue for the organization.

The aim of our work is to schedule Enterprise and Scientific Grid workloads so as to minimise the costs, while ensuring desired Quality-of-Service with a certain degree of confidence. We aim to move away from the existing paradigm of scheduling each application individually and focus instead on scheduling the various types of tasks across a heterogeneous set of resources such that applications comprising these tasks can be executed while satisfying their cost and performance constraints. In the OGSA model of
the Grid [17], workflow tasks are mapped onto Grid services, so we will use the terms workflow tasks and Grid services interchangeably throughout the rest of the paper, depending on whether we are talking about the workflow construct or the implementation.

Figure 1. The Grid as a Queueing Network: Applications are routed around the network in the order defined by the Workflows. To guarantee QoS, the 95% confidence limit of the response time distribution for the workflow should be less than the Application deadline.

We propose a scheduling architecture for Grid computing infrastructures that aims to minimise the cost of application execution while ensuring that the quality-of-service constraints are satisfied. The stochastic nature of the Grid is built into the scheduling formulation in terms of queueing theory. In modelling the Grid as a queueing network (Figure 1), we are able to use more accurate forecasting mechanisms which improve our ability to negotiate for advance reservations and Grid futures. Our treatment of the scheduling problem is similar to the work described in [13, 18, 19] in that resources are assigned to various types of tasks. However, it improves upon the work in that the allocation of resources to services is more deterministic and quality-of-service and cost play a pivotal role in determining the resource assignment.

Our scheduling approach is predictive in nature, in that it is able to use various forecasting mechanisms to predict the workload on the Grid at a certain point in the future and calculate an optimal resource assignment to the various Grid services. Therefore, any applications that are submitted for execution are channeled straight through to the appropriate resources without first undergoing a scheduling phase. This avoids the possibility of the scheduler becoming a performance bottleneck and reduces the application turn-around time.

In the following section, we describe our scheduling approach in detail. We verify our approach using simulation and provide performance results in section 3, before concluding.

Figure 2. Grid Scheduling Architecture

2 Quality-of-Service Constrainted Scheduling

A typical Grid consists of a number of services and a number of physical resources, including compute resources that are capable of hosting these services as well as storage resources, network resources etc. Enterprise and Scientific Grid applications are typically defined in terms of workflows, consisting of one or more tasks that may communicate and cooperate to achieve their objective. The job of the scheduler is to select a set of resources on which to schedule the tasks of an application, coordinate the execution of the tasks on the compute resources and manage the data distributions and communication between the tasks.

The Grid experiences a certain workload at any given time: Applications and workflows submitted by various users for execution. These applications are compositions of different tasks, that are ultimately translated into Grid service invocations. Therefore the overall workload translates into workload on each of these individual services (See figure 2). The scheduling problem can hence be re-formulated so that instead of assigning resources to the individual applications and their tasks, we assign resources to the various Grid services such that all of the requests received by the that service can be handled by the assigned compute resources, while meeting some pre-defined performance constraints. Assigning a resource to a service involves staging the service executable to the selected resource and setting up the execution environment. The workload to the service is then distributed proportionally across the assigned resources.

Our method of scheduling is particularly suited to Enterprise and Scientific Grids where there usually are a relatively small set of workflows that are executed repeatedly,
e.g. in Particle Physics experiments, Climate Modelling, Medical Image Analysis, Portfolio Optimisation in Financial Institutions [20, 21] etc, usually with tight performance constraints, e.g. in Image-Guided Neurosurgery. It is in these scenarios, where a small set of workflows constitute a large percentage of the workload that we intend to demonstrate the efficacy of our approach.

We consider workflows which have been defined as DAGs (Directed Acyclic Graphs) (see Figure 3), with deadlines that indicate the maximum time that the user is prepared to wait for the execution of these workflows to complete. We hence need to make sure that compute resources are allocated to the workflow tasks (Grid services) such that their execution in the sequence defined by the workflows completes within the given time (See figure 1). In the case of workflows with multiple execution paths, where we cannot determine the critical path through the workflow before scheduling, we need to ensure that all paths through the workflow satisfy the deadline constraints. For example, to guarantee that Workflow 2 (Figure 3) meets its deadline, we need to ensure that:

\[
P\{W_1 + W_2 + W_5 + W_6 + W_7 \leq D_2\} \geq \alpha_2
\]

\[
P\{W_1 + W_2 + W_3 + W_4 + W_7 \leq D_2\} \geq \alpha_2
\]

where \(W_i\) represents the sojourn time distribution of service \(i\), \(D_2\) is the workflow deadline and \(\alpha_2\) is the confidence with which we want the QoS constraint to hold.

The scheduling problem is hence to minimize the cost, given by the equation

\[
c^T x + k^T z(x_0, x)
\]

where the vector \(c\) represents the resource costs, the vector \(x\) represents the resources assignments, the function \(z\) represents the necessary switches and the vector \(k\) represents the associated switching costs, subject (as described earlier) to the constraints that all the paths through the various workflow \(i\) should satisfy the associated deadline \(D_i\). In addition, we define other constraints to ensure that the assigned service rates exceed the arrival rates for each Grid service and that the ratio of arrival rate to service rate, \(\rho\), at each resource is less than 1: a fundamental requirement for a stable queue.

In our work, we have assumed general service time distributions: All distributions with finite variance. Hence, assuming Poisson arrivals, for the M/G/1 queue we have well-known formulae (Pollaczek-Khinchin) for the calculation of the mean response time and the variance of the response time, at each service queue [22].

\[
E[r] = E[s] + \lambda E[s](1 + C^2_s)/2\mu(1 - \rho)
\]

\[
Var[r] = Var[s] + \lambda E[s^3]/3(1 - \rho) + \lambda^2 E[s^2]^2/4(1 - \rho)^2
\]

Given that the response time distribution of the entire workflow is the convolution of the response time distributions of the individual stages in the workflows execution path, assuming independently distributed services, we have the following equations for the mean and variance for the response time of the entire workflow:

\[
E[R_{\text{workflow}}] = \sum_{i} E[R_i]
\]

\[
\text{var}[R_{\text{workflow}}] = \sum_{i} \text{var}[R_i]
\]

for each of the execution paths through the workflow.

To guarantee that workflow deadlines will be satisfied with confidence \(\alpha\), we use the Vysochanski-Petunin inequality, which gives a lower bound for the probability that a random variable with finite variance lies within a certain number of standard deviations of the variable’s mean. We use the Vysochanski-Petunin inequality because it tends to give tighter bounds than the more well known Chebyshev’s inequality for unimodal distributions. From this inequality, we can calculate the upper bound on the number \(n\) of standard deviations within which lies the \(\alpha\) percentile. We then add the following constraint to our model to ensure that the deadline is met:

\[
E[R_{\text{workflow}}] + n \times \text{stdev}[R_{\text{workflow}}] \leq D_{\text{workflow}}
\]
$\textbf{Variable Definitions}$
Positive variable arrivals, subdeadline, stdev;
Binary variable alloc;

$\textbf{Minimise the cost of resource allocation and setup costs}$
cost.. \ Z = e= \sum((j,i), \ \text{costs}(j) \times \text{alloc}(j,i) + \text{setup\_costs}(j) \times \text{alloc}(j,i));

$\textbf{Rho}(j,i)$, the ratio of arrival rate to service rate should be less than one
utilisation(j,i).. arrivals(j,i) - mu(j,i) =l= 0;

$\textbf{The mean response time for the service calculated according to the formula for M/G/1}$
mean\_service\_time(j,i).. alloc(j,i) \times \text{e}\_s(j,i) + arrivals(j,i) \times \text{e}\_s(j,i) \times (1 + \sqrt{cs(j,i)})
/(2 \times (1 - \text{arrivals(j,i)/mu(j,i)})) =l= \text{subdeadline}(i);

$\textbf{The mean variance for the response time of the M/G/1 queue should be less than the}$
variance\_limit(j,i).. alloc(j,i) \times \text{var\_s}(j,i) + arrivals(j,i) \times \text{e}\_s(j,i) \times (1 + \sqrt{cs(j,i)})
/(3 \times (1 - \text{arrivals(j,i)/mu(j,i)})) + \sqrt{\text{arrivals(j,i)}} \times \sqrt{e\_s2(j,i)}
/(4 \times \sqrt{1 - \text{arrivals(j,i)/mu(j,i)}}) =l= \text{sqrt(stdev}(i));

$\textbf{Variance of response time of the workflow is equal to the sum of the variances of}$
sum\_of\_variances(k).. \ \sum(i, \ \text{workflow\_composition}(k,i) \times \text{sqrt(stdev}(i)))) =l= \text{sqrt(stdev}(k));

$\textbf{Vyochanskii-Petunin confidence summation. Confidence}(k)$ is number of standard
$\textbf{deviations that gives the required confidence for workflow k to meet deadline}(k)$
convolution(k).. \ \sum(i, \ \text{workflow\_composition}(k,i) \times \text{subdeadline}(i)) + \text{confidence}(k) \times \text{stdev}(k)
= l= \text{deadline}(k);

$\textbf{Sum of arrivals allocated across the various resources is greater than the arrival}$
arrival\_rates(i).. lambda(i) - \sum(j, \text{arrivals}(j,i)) = l= 0;

$\textbf{If allocated arrivals to the resource, set the binary allocation variable to 1. M}$
integrality(j,i).. arrivals(j,i) - M \times \text{alloc}(j,i) = l= 0;

$\textbf{Solve the above model using the MINLP algorithms, for the variables arrivals(j,i)}$
$\textbf{s and alloc(j,i)}$
solve scheduling using minlp minimizing Z;

\textbf{Figure 4. NLP Model}

The results obtained from the optimisation algorithm are
used to configure the workload distribution across the Grid
resources and the resource allocation tables in the Grid In-
frastucture. In our problem formulation, we are assigning a
certain proportion of \(n\) resources to each Grid service. The
workload on the Grid service is hence divided proportion-
ally across the \(n\) resources. Each resource is treated as an
M/G/1 queue, where \(G\) is an arbitrary distribution.

The proportional distribution of workload across the \(n\)
resources is vital to our scheduling architecture to ensure
that the queues operate as desired and satisfy the execution-
time constraints. In our work, we have implemented the
\textit{Weighted Random Routing Approach}, whereby the propor-
tion of jobs routed to a particular resource is determined by
the proportion of the total service rate that is provided by that resource. These service rates are deduced from the result of the optimisation process.

In a scheduling approach based on the current values of the parameters alone, there is a danger that the resulting schedule, though optimal with regards to the current parameter values, will be unsuitable for use soon after it is calculated. Therefore, our scheduling model aims to forecast the values of the parameters at some time in the future and calculate a schedule for those values. The parameters most likely to vary over time are the workflow arrival rates. At any point in time, the arrival rates in next scheduling window are likely to be closest to the most recent arrival rates observed. We assume that Grid workloads display seasonality and trend in a similar fashion to Web Server Workloads [24], therefore we use the Holt-Winter’s method [25] to forecast future arrival rates. The least mean squared error technique is used to discover the initial smoothing constants and then using these values, we predict the future workloads. We intend to further extend our approach to take account of more complex workload models and scenarios and capacity planning techniques (see section 4).

3 Simulation and Performance Results

Mixed-Integer Non-linear programming (MINLP) is computationally extremely challenging [26, 25]. MINLP programs are considered to be NP-Complete. Hence, in this paper, we have solved relatively small optimisation problems in order to demonstrate the efficacy of our scheduling algorithm. Figure 5 shows the solution times for various problem sizes, where the problem size is the product $mn$, where $m$ is the number of services and $n$ is the number of services. We have solved the programs on the NEOS Server, using the SBB MINLP optimiser [27, 28]. We are currently investigating various approximations and techniques to increase the scalability of our approach (see section 4).

We compare our scheduling approach, Queueing Scheduler, with three well-known techniques for Grid scheduling Simple Scheduler, Static Scheduler and Dynamic Scheduler. The Simple and Static schedulers are based on well-known advance reservation based co-allocation techniques [29, 5]. They differ in that the Static scheduler performs sub-deadline re-calculation and re-negotiation if the initial co-allocation request fails, whereas the Simple scheduler aims to negotiate only within the bounds of the deadlines calculated from the outset. Our Dynamic scheduler is a variant of work presented in [14, 11], where the tasks of an application are scheduled Just-in-time. All of the reservations-based schedulers implement backfilling and have been extended to make reservations for conservative estimates of the service runtimes, in the absence of accurate performance predictions [11].

We have simulated several common workflow structures in scientific and commercial workflows: sequential, parallel and hybrid. We have conducted experiments with different workloads (generated using a Poisson distribution with a constant mean throughout each experiment) and variances in task execution times over a fixed set of resources, with fixed deadlines for each of the workflows (see figure 3). The tasks of a workflow are defined in terms of MI (Millions of Instructions) and the resources in terms of MIPS (Millions of Instructions Per Second). We have simulated 8 types of services and 24 resources of 4 different speeds (see tables 1 and 2). In our current experiments, we consider the data transmission times and costs to be negligible. Furthermore, it is assumed that the setup times and costs, and the scheduling times are negligible.

Our Simulation Architecture (see figure 6) is based on...
the ICENI Grid Computing infrastructure [4] and the GridSim toolkit [30], which have been extended to include support for advance reservations and queueing.

We have conducted a large number of experiments with varying arrival rates and variances for each of the services. Experiments were conducted for 4000 workflows to ensure that system reached a steady state. Data for the first 2000 workflows was discarded. In particular, we collected data regarding the number of failures, the average cost of workflow execution, the average utilisation across the allocated resources and the mean and the variance of the application execution times. We present the most important of these results below.

### Table 1. Resources Speeds and Cost

<table>
<thead>
<tr>
<th>Resource ID</th>
<th>Speed (MIPS)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource1-6</td>
<td>200,000</td>
<td>24.0</td>
</tr>
<tr>
<td>Resource7-12</td>
<td>150,000</td>
<td>18.0</td>
</tr>
<tr>
<td>Resource13-18</td>
<td>100,000</td>
<td>15.0</td>
</tr>
<tr>
<td>Resource19-24</td>
<td>50,000</td>
<td>10.0</td>
</tr>
</tbody>
</table>

### Table 2. Task Sizes

<table>
<thead>
<tr>
<th>Workflow ID</th>
<th>Deadline(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow1</td>
<td>35.0</td>
</tr>
<tr>
<td>Workflow2</td>
<td>43.75</td>
</tr>
<tr>
<td>Workflow3</td>
<td>45.5</td>
</tr>
</tbody>
</table>

### Algorithm 1 Simple scheduling algorithm: Upon failure in creating advance reservations, the algorithm terminates

**Require:** A Workflow $W$ defined as a Directed Acyclic Graph  
**Ensure:** DAG is scheduled within deadline

1: distribute deadline proportionally over all tasks $T_i \in W$  
2: repeat  
3: $S \leftarrow$ all unscheduled tasks  
4: for all $i \in S$ do  
5: compute ready time for task $i$  
6: calculate reservation duration based on performance prediction or conservative estimate  
7: request processing time, price and available time slots from resources with ready-time and deadline constraints  
8: make advance reservations on desired resources for all tasks in $i$  
9: if advance reservations could not be created then  
10: flag failure to schedule, cancel reservations and terminate  
11: end if  
12: end for  
13: until all tasks have been scheduled

### Algorithm 2 Static scheduling algorithm: Upon failure in creating advance reservations, the schedule attempts to relax the subdeadlines and reschedule

**Require:** A Workflow $W$ defined as a Directed Acyclic Graph  
**Ensure:** DAG is scheduled within deadline

1: distribute deadline proportionally over all tasks $T_i \in W$  
2: repeat  
3: $S \leftarrow$ unscheduled tasks whose parent tasks have been scheduled  
4: for all $i \in S$ do  
5: compute ready time for task $i$  
6: calculate reservation duration based on performance prediction or conservative estimate  
7: request processing time, price and available time slots from resources with ready-time and deadline constraints  
8: make advance reservations on desired resources for all tasks in $i$  
9: if advance reservations could not be created then  
10: if attempts threshold not reached then  
11: re-calculate subdeadlines and attempt to reschedule  
12: else  
13: flag failure to schedule, cancel reservations and terminate  
14: end if  
15: else  
16: adjust subdeadline of $i$ and re-distribute deadline over remaining tasks  
17: end if  
18: end for  
19: until all tasks have been scheduled

### Algorithm 3 Dynamic scheduling algorithm: The algorithm performs Just-in-Time Scheduling

**Require:** A Workflow $W$ defined as a Directed Acyclic Graph  
**Ensure:** DAG is scheduled within deadline

1: distribute deadline proportionally over all tasks $T_i \in W$  
2: repeat  
3: $S \leftarrow$ unscheduled task whose parent tasks have completed execution  
4: for all $i \in S$ do  
5: set ready time to current time  
6: calculate reservation duration based on performance prediction or conservative estimate  
7: request processing time, price and available time slots from resources with subdeadline constraints  
8: make advance reservations on desired resources for all tasks in $i$  
9: if advance reservations could not be created then  
10: if attempts threshold not reached then  
11: re-calculate subdeadlines and attempt to reschedule  
12: else  
13: flag failure to schedule, cancel reservations and terminate  
14: end if  
15: else  
16: adjust subdeadline of $i$ and re-distribute deadline over remaining tasks  
17: end if  
18: end for  
19: until all tasks have completed execution
3.0.1 Failures

Figures 7 and 8 show the percentage of applications that failed to execute or meet their deadlines, as the variance of the task execution times was increased, under light and heavy loads respectively. The results indicate that the Dynamic scheduler performs better than its reservations-based counterparts, but was outperformed by the Queueing Scheduler which recorded the lowest number of failures as the variances were increased, in both light and heavy workloads. Because the deadlines (as calculated above) are quite tight, all of the schedulers registered 100% failures at maximum variance.

3.0.2 Cost

Figures 9 and 10 show the average cost per workflow execution. Once again, the Dynamic scheduler performance increasingly better than its co-allocation based counterparts as the variance is increased, however is outperformed by the Queueing scheduler. The costs are calculated based on the assumption that the resource providers, in a commercial Grid setting, will charge for the duration of the reservations created, regardless of the actual resource time used. The sum of the resource costs is averaged over the number of successful executions. An interesting thing to note w.r.t the reservations-based schedulers is the occasional decrease in average cost as the load is increased. This is because at higher arrival rates, we have a higher number of conflicts and thus fewer successful executions, which in turn means that slower and cheaper resources can be used more often. Furthermore, the reservations-based schedulers are using roughly the same number of resources as for lower workloads, however the resources are now being better utilised. The Queueing scheduler has the ability to optimise and al-
locate the least number of resources that would satisfy the QoS constraints, leading to better workload allocations, increased utilisation and hence lower average workflow costs even as the load is increased.

3.0.3 Utilisation

Figures 11 and 12 show the average utilisation across the allocated Grid resources. It shows that the Dynamic scheduler starts off having exactly the same utilisation as the Queueing scheduler but drops sharply as the variance increases and the scheduler has to make resource reservations for increasingly large chunks of time. However, as the workload is increased, the number of conflicts between reservations, and hence failures, increase and the average resource utilisation is much lower than the Queueing scheduler.

4 Conclusions and Further Work

In this paper, we proposed a new Grid scheduling algorithm that minimizes the cost of execution of workflows, while ensuring that their associated Quality-of-Service constraints are satisfied. We have described how the algorithm views the Grid as a queueing system, seamlessly routing the workflows through the network. We have demonstrated our algorithms ability to efficiently schedule applications without requiring performance prediction or negotiation for advance reservations for every stage of the workflow, which leads to significant performance gains. We have also shown our algorithm’s ability to schedule current and future workloads and not just individual applications, and the ability to guarantee QoS within required confidence bounds for the end-to-end execution of workflows. We have evaluated our algorithm using small problem sizes. The experimental results demonstrate that using the proposed scheduling algorithm can satisfy workflow deadline constraints with lower cost and higher utilisation of the underlying Grid resources and hence leads to reduced failure rates and turnaround times, especially in heavily-loaded Grids.

The results that we have presented in this paper assume negligible setup costs and times, and negligible scheduling times. As these costs and times become more significant, the advantage of the Queueing Scheduler over its reservations-based counterparts becomes even more apparent since the Queueing scheduler incurs this cost relatively infrequently as compared to the reservations-based schedulers.

We are currently investigating methods to approximate the performance equations and hence speed up the optimisation process. We have developed several preliminary solutions, that involve Quadratic Programming and Dynamic Programming. We are also investigating scheduling architectures where we have a network of schedulers, each optimising the schedule for a subset of the overall workload on the Grid.

We are currently extending our approach to optimise for network bandwidth and costs, which becomes a significant factor in Data Grids, and introducing schedule optimisation and recovery mechanisms for resource failures. We also aim to improve our forecasting mechanisms and capacity planning techniques to allow us to calculate more accurate schedules for varying workloads.

5 Acknowledgements

We would like to thank the Engineering and Physical Sciences Research Council (EPSRC) for funding this work. We would also like to thank Dr. William Knottenbelt and Prof. Berc Rustem for their help and advice regarding performance modelling and optimisation issues. Finally, we
would like to acknowledge the help and support of the members of the London e-Science Centre.

References


