Performance Analysis of Multiple Site Resource Provisioning: Effects of the Precision of Availability Information

Marcos Dias de Assunção and Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{marcosd, raj}@csse.unimelb.edu.au

Abstract. Emerging deadline-driven Grid applications require a number of computing resources to be available over a time frame, starting at a specific time in the future. To enable these applications, it is important to predict the resource availability and utilise this information during provisioning because it affects their performance. It is impractical to request the availability information upon the scheduling of every job due to communication overhead. However, existing work has not considered how the precision of availability information influences the provisioning. As a result, limitations exist in developing advanced resource provisioning and scheduling mechanisms. This work investigates how the precision of availability information affects resource provisioning in multiple site environments. Performance evaluation is conducted considering both multiple scheduling policies in resource providers and multiple provisioning policies in brokers, while varying the precision of availability information. Experimental results show that it is possible to avoid requesting availability information for every Grid job scheduled thus reducing the communication overhead. They also demonstrate that multiple resource partition policies improve the slowdown of Grid jobs.

1 Introduction

Advances in distributed computing have resulted in the creation of computational Grids. These Grids, composed of multiple resource providers, enable collaborative work and resource sharing amongst groups of individuals and organisations. These collaborations, widely known as Virtual Organisations (VOs) [1], require resources from multiple computing provider sites, which are generally clusters of computers managed by queueing-based Resource Management Systems (RMSs), such as PBS and Condor.

Emerging deadline-driven Grid applications require access to several resources and predictable Quality of Service (QoS). A given application may require a number of computing resources to be available over a time frame, starting at a specific time in the future. However, it is difficult to provision resources to these applications due to the complexity of providing guarantees about the start or

completion times of applications currently in execution or waiting in the queue. Current RMSs generally use optimisations to the first come first served policy such as backfilling to reduce the scheduling queue fragmentation, improve job response time and maximise resource utilisation. These optimisations make it difficult to predict the resource availability over a time frame as the jobs' start and completion times are dependent on resource workloads.

To complicate the scenario further, users may access resources via mediators such as brokers or gateways. The design of gateways that provision resources to deadline-driven applications relying on information given by current RMSs may be complex and prone to scheduling decisions that are far from optimal. Moreover, it is not clear how gateways can obtain information from current RMSs to provision resources to QoS demanding applications. Existing work on resource provisioning in Grid environments has used conservative backfilling wherein the fragments of the scheduling queue are given to be provisioned by a broker [2]. These fragments are also termed availability information or free time slots. We consider impractical to request the free time slots from providers upon the scheduling of every job due to potential communication overhead.

In this paper, we investigate how the precision of availability information affects resource provisioning in multiple site environments. In addition, we enhance traditional schedulers, allowing the obtention of availability information required for resource provisioning. We evaluate the reliability of the provided information under varying conditions by measuring the number of provisioning violations. A violation occurs when the information given by the resource provider turns out to be incorrect when it is used by the gateway. Additionally, we evaluate the impact of provisioning resources to Grid applications on providers' local requests by analysing the job bounded slowdown. We investigate whether EASY backfilling and multiple partition policies provide benefits over conservative backfilling if job backfilling is delayed, enabling large time slots to be provided to the gateway.

2 Related Work

The performance analysis and the policies proposed in this work are related to previous systems and techniques in several manners.

Modelling providers' resource availability: AuYoung *et al.* [3] consider a scenario wherein service providers establish contracts with resource providers. The availability information is modelled as ON/OFF intervals, which correspond to off-peak and peak periods respectively. However, they do not demonstrate in practice how this information can be obtained from RMSs.

Advance reservations and creation of alternatives to rejected requests: Mechanisms for elastic advance reservations and generation of alternative time slots for advance reservation requests have been proposed [4, 5]. These models can be incorporated in the provisioning scenario described in this work to improve resource utilisation and generate alternative offers for provisioning violations. However, we aim to reduce the interaction between resource providers and gateways by allowing the providers to inform the gateways about their spare

capacity. We focus on how the availability information can be obtained from RMSs and how reliable it is under different conditions.

Multiple resource partition policies: Work on multiple resource partitions and priority scheduling has shown to reduce the job slowdown compared to EASY backfilling policies [6]. We build on this effort and extend it to enable other multiple partition policies. We also propose a new multiple resource partition policy based on load forecasts for resource provisioning.

Resource allocation in consolidated centres: Padala *et al.* [7] apply control theory to address the provision of resources to multi-tier applications in a consolidated data centre. Garbacki and Naik [8] consider a scenario wherein customised services are deployed on virtual machines which in turn are placed into physical hosts. Although the provisioning of resources to applications in utility data centres is important, here we focus on traditional queue-based RMSs.

Shared spaces for collaborative scheduling: Ranjan *et al.* [9] use a P2P tuple space to co-ordinate the matching of user requests and providers' resource claims [9] in federated clusters. We will explore a shared space in future work with multiple gateways exchanging resource shares obtained from providers.

Resource provisioning: Singh *et al.* [2, 10] present a provisioning model where Grid sites provide information on the time slots over which sets of resources are available. The sites offer their resources to the Grid in return for payments, thus they present a cost structure consisting of fixed and variable costs over the resources provided. The main goal is to find a subset of the aggregated resource availability, termed as resource plan, such that both allocation costs and application makespan are minimised. Our work is different in the sense that we investigate multiple approaches to obtain availability information and how reliable this information can be in multiple site environments.

3 Multiple-Site Resource Provisioning

The multiple site scenario is depicted in Figure 1, which shows DAS-2's configuration used later in the experiments. A Resource Provider (RP) contributes a share of computational resources to a Grid in return for regular payments. An RP has local users whose resource demands need to be satisfied, yet it delegates provisioning rights over spare resources to an InterGrid Gateway (IGG) by providing information about the resources available in the form of free time slots. A free time slot describes the number of resources available, their configuration and time frame over which they will be available. The delegation can be made through a secure protocol such as SHARP [11].

A Grid can have peering arrangements with other Grids managed by IGGs and through which they co-ordinate the use of resources. This work does not address peering arrangements [12]. Here, we investigate how an IGG can provision resources to applications based on the availability information given by RPs.

Problem Description: An IGG attempts to provision resources to meet its users' QoS demands, improve the job slowdown and minimise the number of violations. A violation occurs when a user tries to use the resources allocated by

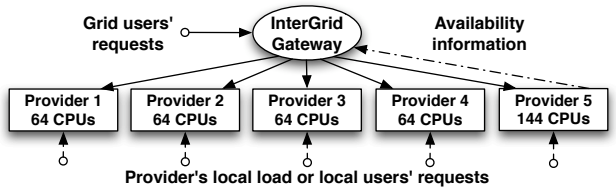

Fig. 1. Resource providers contribute to the Grid but have local users.

the IGG and they are no longer available due to wrong or imprecise availability information given by the RP. RPs, on the other hand, are willing to increase the resource utilisation without compromising their local users requests. IGG should achieve allocations that minimise the response time and slowdown of Grid users' requests without perceivable impact on the slowdown of the RPs' local requests.

Grid Requests: A request received by an IGG is contiguous and needs to be served with resources from a single resource provider. They contain a description of the required resources and the time duration over which they are required. A request can demand either QoS or a best effort service. A QoS constrained request has an execution estimate, a deadline and a ready time before which the request is not available for scheduling. A best effort job has an execution time estimate but does not have a deadline.

4 Policies Investigated

We have extended traditional scheduling policies in order to obtain the free time slots from resource providers. The policies utilise an 'availability profile' similar to that described by Mu'alem and Feitelson [13]. The availability profile is a list whose entries describe the CPUs available at particular times in the future. These entries correspond to the completion or start times of jobs and advance reservations. By scanning the availability profile and using other techniques described here, the resource providers inform the gateway about the free time slots; the gateway in turn can carry out provisioning decisions based on this information.

Conservative Backfilling Based Policies: Under conservative backfilling, a job is used to backfill and start execution earlier than expected, given that it does not delay any other job in the scheduling queue [13]. In order to reduce complexity, the schedule for the job is generally determined at its arrival and the availability profile is updated accordingly. Given those conditions, it is possible to obtain the free time slots by scanning the availability profile. This approach, depicted in Figure 2a, was also used by Singh *et. al* [2, 10]. In that case, the availability profile is scanned until a given time horizon thus creating windows of availability or free time slots; the finish time of a free time slot is either the finish time of a job in the waiting queue or the planning horizon. We have also implemented a conservative backfilling policy that uses multiple resource partitions based on the EASY backfilling proposed by Lawson and Smirni [6].

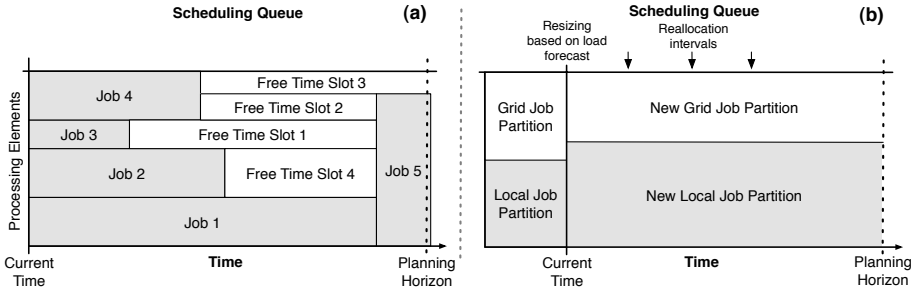

Fig. 2. Obtaining free time slots: (a) conservative backfilling, (b) multiple partitions.

Multiple Resource Partition Policies: We have implemented 3 policies based on multiple resource partitions. In our implementation, each policy divides the resources available in multiple partitions and assigns jobs to these partitions according to partition predicates. A partition can borrow resources from another when they are not in use by the latter and borrowing is allowed by the scheduler. One policy implements the EASY backfilling (aka aggressive backfilling) described by Lawson and Smirni [6]. In this case, each partition uses aggressive backfilling and has a pivot, which is the first job in the waiting queue for that partition. A job belonging to a given partition can start its execution if it does not delay the partition's pivot and the partition has enough resources. In case the partition does not have enough resources, the job can still start execution if additional resources can be borrowed from other partitions without delaying their pivots. Additionally, the policy uses priority scheduling wherein the waiting queue is ordered by priority when the scheduler is backfilling. In order to evaluate this policy, we attempt to maintain the configuration provided by Lawson and Smirni [6], which selects partitions according to the jobs' runtimes. The partition $p \in \{1, 2, 3\}$ for a job is selected according to Equation 1, where t_r is the job's runtime in seconds.

$$p = \begin{cases} 1, & 0 < t_r < 1000 \\ 2, & 1000 \leqslant t_r < 10000 \\ 3, & 10000 \leqslant t_r \end{cases} \tag{1}$$

We also introduce a new policy, depicted in Figure 2b, in which, the partitions are resized by the scheduler at time intervals based on a load forecast computed from information collected at previous intervals. As load forecasts are prone to be imprecise, when the scheduler resizes partitions, it also schedules reallocation events. At a reallocation event, the scheduler evaluates whether the load forecast has turned out to be an underestimation or not. If the load was underestimated, the policy resizes the partitions according to the load from the last resizing period until the current time and backfill the jobs, starting with the local jobs. We use EASY backfilling with configurable maximum number of pivots, similarly

to MAUI scheduler [14]. The policy can be converted to conservative backfilling by setting the number of pivots to a large value, here represented by ∞.

Algorithm 1 describes two procedures of the load forecast policy; *getFreeTimeSlots* is invoked when the provider needs to send the availability information to the IGG whereas *reallocationEvent* is triggered by *getFreeTimeSlots* to verify whether the previous forecast has turned out to be precise or if a reallocation is required. From line 3 to 4 the scheduler becomes conservative backfilling based by setting the number of pivots in each partition to ∞. It also schedules the jobs in the waiting queue. After that, the scheduler returns to EASY backfilling (line 5). Then, from line 6 to 10, the scheduler obtains the load forecast and the free time slots, which are resized by modifying the number of CPUs according to the number of resources expected to be available over the next interval. Next, the scheduler triggers a reallocation event. At line 19 the scheduler verifies whether the forecast was underestimated. If that is the case, it turns the policy to conservative backfilling and informs the gateway about the availability. We have also implemented a multiple resource partition policy based on conservative backfilling.

Algorithm 1: Provider's load forecasting policy.

```
 1  procedure getFreeTimeSlots()
 2  begin
 3      set number of pivots of local and Grid partitions to ∞
 4      schedule / backfill jobs in the waiting queue
 5      set number of pivots of local and Grid partitions to 1
 6      actualLoad ← load of waiting/running jobs
 7      forecast ← get the load forecast
 8      percToProvide ← min{0, 1 − actualLoad}
 9      slots ← obtain the free time slots
10      slots ← resize slots according to percToProvide
11      if percToProvide > 0 then
12          inform gateway about slots and schedule reallocation event
13      schedule next event to obtain free time slots
14  end
15  procedure reallocationEvent()
16  begin
17      localLoad ← obtain the local load
18      forecast ← get the previously computed forecast
19      if localLoad > forecast then
20          set number of pivots of local partition to ∞
21          schedule / backfill jobs in the waiting queue
22          set number of pivots of Grid partition to ∞
23          schedule / backfill jobs in the waiting queue
24          slots ← obtain the free time slots
25          inform gateway about slots
26      else
27          schedule next reallocation event
28  end
```

The policies we consider for the gateway are described as follows:

- **Least loaded resource:** The gateway submits a job to the least loaded resource based on utilisation information sent by the resource providers every ten minutes.
- **Earliest start time:** This policy is employed for best effort and deadline constrained requests when the resource providers are able to inform the gateway about the free time slots. When scheduling a job using this policy, the scheduler is given the provider's availability information and the job. If the providers send the information at regular time intervals, this information is already available at the gateway; otherwise, the gateway requests it from the resource providers. If the job is not deadline constrained, the gateway selects the first provider and submits the job to it. When the job is deadline constrained, the gateway attempts to make a reservation for it. If the reservation cannot be accepted by the provider, the provider updates its availability information at the gateway.

5 Performance Evaluation

5.1 Scenario Description

We have modelled DAS-2 Grid configuration because job traces collected from this Grid and its resources' configuration are publicly available and have been previously studied [15]. As depicted in Figure 1 beforehand, DAS-2 is a Grid infrastructure deployed in the Netherlands comprising 5 clusters. The evaluation of the proposed mechanism is performed through simulation by using a modified version of GridSim.[1] We resort to simulation as it provides a controllable environment and enables us to carry out repeatable experiments.

The resource providers' workloads have been generated using Lublin and Feitelson [16]'s model, here referred to as Lublin99. Lublin99 has been configured to generate four month long workloads of type-less jobs (i.e. we do make distinctions between batch and interactive jobs); the maximum number of CPUs used by the generated jobs is set to the number of nodes in the clusters. Grid jobs' arrival rate, number of processors required and execution times are modelled using DAS-2 job trace available at the Grid Workloads Archive.[2] We use the interval from the 9^{th} to the 12^{th} month. The jobs' runtimes are taken as runtime estimates. Although this generally does not reflect the reality, it provides the basis or bounds for comparison of scheduling approaches [17].

To eliminate the simulations' warm up and cool down phases from the results, the last simulated event is the arrival of the last job submitted in any of the workloads and we discard the first two weeks of the experiments. In the case of the forecast based policy, the second week is used as training period. We select randomly the requests that are deadline constrained. In order to generate the request deadlines we use a technique described by Islam *et al.* [18], which provides a feasible schedule for the jobs. To obtain the deadlines, we perform the

[1] More information available at: http://www.gridbus.org/intergrid/gridsim.html
[2] Grid Workloads Archive website: http://gwa.ewi.tudelft.nl/pmwiki/

experiments by using the same Grid environment using aggressive backfilling at the resource providers and 'submit to the least loaded resource' policy at the gateway. A request deadline is the job completion under this scenario multiplied by a *stringency factor*. The load forecasting uses a weighted exponential moving average [19], considering a window of 25 intervals.

Performance Metrics: One of the metrics considered is the bounded job slowdown (*bound*=10 seconds) hereafter referred to as job slowdown for short [17]. Specifically, we measure the bounded slowdown improvement ratio $\mathcal{R}$ given by Equation 2, where s_{base} is the job slowdown using a base policy used for comparison; and s_{new} is the job slowdown given by the policy being evaluated. We calculate the ratio $\mathcal{R}$ for each job and then take the average. The graphs presented in this section show average ratios.

$$\mathcal{R} = \frac{s_{base} - s_{new}}{min(s_{base}, s_{new})} \tag{2}$$

We also measure the number of violations and messages exchanged between providers and IGG to schedule Grid jobs. The reduction in the number of messages required is used for estimating the tradeoff between precision of information and communication overhead. A given job j faces a violation when the inequality $j_{pst} - j_{gst} > T$ is *true*, where j_{gst} is the job start time assigned by the gateway based on the free time slots given by providers; j_{pst} is the actual job start time set by the provider's scheduler; and T is a tolerance time. The experiments performed in this work use a T of 20 seconds. A violation also occurs when a resource provider cannot accept a reservation request made by the gateway.

Policy Acronyms: Due to space limitations, we abbreviate the name of the evaluated policies in the following manner. A policy name comprises two parts separated by +. The first part represents the policy employed by the provider whereas the second is the gateway policy. In the resource provider's side, **Ar** stands for Advance reservation, **Eb** for EASY backfilling, **Cb** for Conservative backfilling, **M** for Multiple partitions and **Mf** for Multiple partitions with load forecast. For the gateway's policy, **least-load** means 'submit to least loaded resource', **earliest** represents 'select the earliest start time', **partial** indicates that providers send free time slot information to the gateway on a periodical basis and **ask** means that the gateway requests the free time slot information before scheduling a job. For example, **ArEbMf+earliest-partial** indicates that providers support advance reservation, EASY backfilling, multiple partitions and load forecasts, whereas the gateway submits jobs selecting the earliest start time based on the availability information sent by providers at regular intervals.

5.2 Experimental Results

The first experiment measures the number of messages required by the policies supporting advance reservation and conservative backfilling (i.e. ArCb), both that request the free time slots and those in which the time slots are informed by providers at time intervals. We investigate whether we can reduce the number

of messages required by making the resource providers publish the availability information at gateways at time intervals. We vary the interval for providing the availability information; we also measure the number of violations and average job slowdown to check the tradeoff between the precision of scheduling decisions and the freshness of the information. The planning horizon is set to ∞, so that a provider always informs all the free time slots available. In addition, we consider a two phase commit protocol for advance reservations. The time interval for providing the time slots to the gateway is described in the last part of the name of the policies (e.g. 15 min., 30 min.). The stringency factor is 5 and around 20% of the Grid requests are deadline constrained.

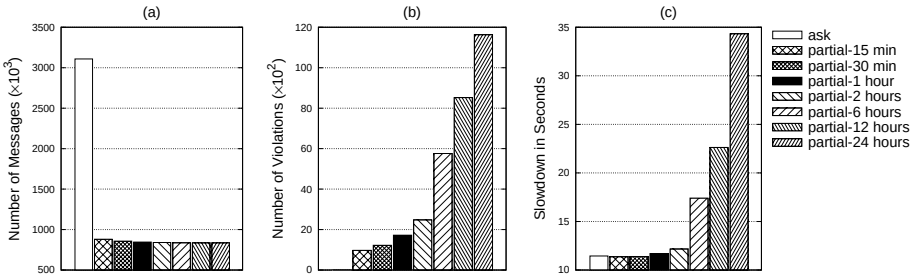

Fig. 3. ArCb+earliest-* policies: (a) number of messages; (b) number of violations; and (c) average job slowdown.

Figure 3a shows that the number of messages required by the policy in which the gateway asks for the time slots upon the schedule of every job (i.e. ArCb+earliest-ask) is larger compared to other policies. In contrast, policies that provide the free time slots at regular intervals or when an advance reservation request fails lead to a lower number of messages.

The number of violations increases as the providers send the availability information at larger intervals (Figure 3b). If the scheduling is made based on information provided every 15 minutes, the number of violations is 973, which accounts for 0.43% of the jobs scheduled. To evaluate whether these violations have an impact on the resource provisioning for Grid jobs, we measure the average bounded slowdown of Grid jobs (Figure 3c). As shown in the figure, there is an increase in the job slowdown as the interval for providing the free time slots increases. However, when the providers send availability information every 15 minutes, the average slowdown is improved. We can conclude that for a Grid like DAS-2 wherein providers send the availability information at intervals of 15 to 30 minutes resource provisioning can be possible using a simple policy supporting conservative backfilling.

The second experiment measures the average of jobs ratio $\mathcal{R}$ described in Equation 2 proposed by Lawson and Smirni [6]. The values presented in the graphs are averages of 5 simulation rounds, each with different workloads for

providers' local jobs. The set of policies used as basis for comparison are EASY backfilling in the providers and 'submit to the least loaded resource' in the gateway. This way, the experiment measures the average improvement ratio wherein the base policies are EASY backfilling and submit to the least loaded resource. The resource providers send the availability information to the gateway every two hours. In this experiment we do not consider deadline constrained requests, as they could lead to job rejections by some policies, which would then impact on the average bounded slowdown.

The results conservative backfilling and 'least loaded resource' policies (i.e. ArCb+least-load and ArCbM+least-load) tend to degrade the bounded slowdown of Grid jobs (Figure 4a). The reason is that submitting a job to the least loaded resource, wherein utilisation is computed by checking how many CPUs are in use at the current time, does not ensure immediate start of a job because other jobs in the waiting queue may have been already scheduled. Moreover, the gateway is not aware of the time slot the job will in fact utilise.

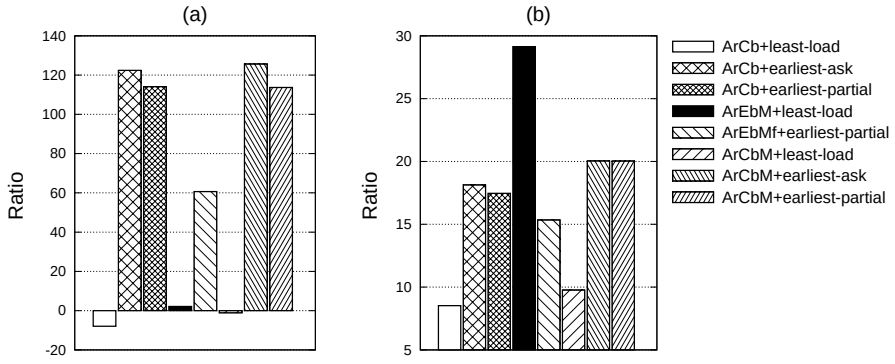

Fig. 4. Slowdown improvement ratio: (a) Grid jobs and (b) providers' local jobs.

The multiple resource partition policies with conservative backfilling without priorities and providing the free time slots to the gateway improve the average slowdown of both Grid jobs (Figure 4a) and providers' local jobs (Figure 5b). ArEbM+least-load, proposed by Lawson and Smirni [6], improves the slowdown of local jobs (Figure 4b) providing little changes in the slowdown of Grid jobs. That occurs because in the original implementation of this policy, higher priority is given to local jobs. The EASY backfilling policy that resizes the resource partitions according to load estimates improves the slowdown of both Grid jobs (Figure 4a) and providers' local jobs but not as much as that of the other multiple partition policies.

We also vary the intervals for providing the free time slots in the previous experiment. Figure 5a shows that for small planning horizons, the multiple resource partition policy with EASY backfilling and load estimates (i.e. ArEbMf+earliest-partial) improves the average ratio, but not as much as the other policies. How-

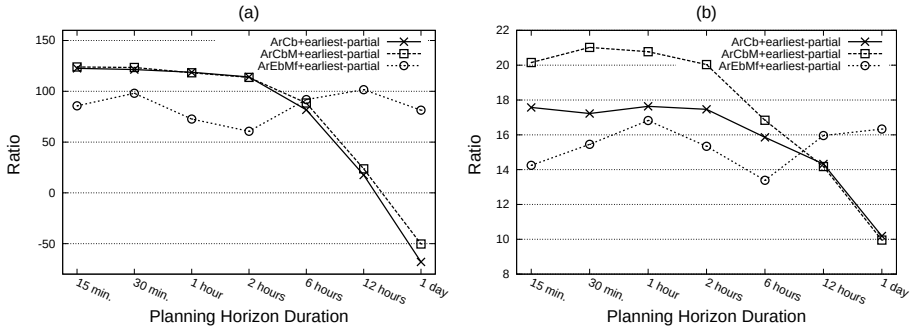

Fig. 5. Slowdown improvement ratio: (a) Grid jobs and (b) providers' local jobs.

ever, as the time interval for providing the availability information increases, the policy outperforms the other multiple partition policies. The slowdown improves compared to the other policies when the interval increases. The reason for the better performance under long intervals is probably because if a load estimate is wrong, the policy becomes a multiple partition conservative backfilling. When an incorrect estimate is identified in a long interval, it may take a while to approach the next interval when the policy will become EASY backfilling again. This conservative backfilling provides a better slowdown and updating the availability in the middle of an interval provides an advantage over the other policies. However, to confirm that, we require further investigation. Furthermore, we expect that better load forecast methods can improve the jobs slowdown under varying intervals.

6 Conclusions and Future Work

This work investigates resource provisioning in multiple site environments. It evaluates whether it is possible to provision resources for Grid applications based on availability information given by resource providers using existing resource management systems. We present empirical results that demonstrate that in an environment like DAS-2, a gateway can provision resources to Grid applications if the resource providers inform the available time slots between 15 and 30 minutes. Additionally, multiple resource partition policies can improve the slowdown of both local and Grid jobs if conservative backfilling is used. Future investigations include more sophisticated resource provisioning policies for the gateways and more accurate load forecasting techniques.

Acknowledgements

We thank the Grid Workloads Archive group for making the Grid workload traces available. This work is supported by DEST and ARC project grants. Marcos' PhD research is partially supported by National ICT Australia (NICTA).

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Int. J. High Perform. Comput. Appl. **15**(3) (2001) 200–222
2. Singh, G., Kesselman, C., Deelman, E.: A provisioning model and its comparison with best-effort for performance-cost optimization in Grids. In: 16th IEEE HPDC, Monterey, USA (2007) 117–126
3. AuYoung, A., Grit, L., Wiener, J., Wilkes, J.: Service contracts and aggregate utility functions. In: 15th IEEE HPDC, Paris, France (2006) 119–131
4. Röblitz, T., Schintke, F., Wendler, J.: Elastic Grid reservations with user-defined optimization policies. In: Workshop on Adaptive Grid Middleware, France (2004)
5. Wieczorek, M., Siddiqui, M., Villazon, A., Prodan, R., Fahringer, T.: Applying advance reservation to increase predictability of workflow execution on the Grid. In: 2nd IEEE e-Science, Washington DC, USA (2006) 82–82
6. Lawson, B.G., Smirni, E.: Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In: 8th JSSPP, London, UK (2002) 72–87
7. Padala, P., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K.: Adaptive control of virtualized resources in utility computing environments. In: 2007 Conference on EuroSys, Lisbon, Portugal (2007) 289–302
8. Garbacki, P., Naik, V.K.: Efficient resource virtualization and sharing strategies for heterogeneous Grid environments. In: 10th IFIP/IEEE IM, Munich, Germany (2007) 40–49
9. Ranjan, R., Harwood, A., Buyya, R.: Peer-to-peer tuple space: A novel protocol for coordinated resource provisioning. Technical Report GRIDS-TR-2007-14, Grid Computing and Distributed Systems (GRIDS) Laboratory, Australia (2007)
10. Singh, G., Kesselman, C., Deelman, E.: Application-level resource provisioning on the grid. In: 2nd IEEE e-Science, Amsterdam, The Netherlands (2006) 83–83
11. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: SHARP: An architecture for secure resource peering. In: 19th ACM Symposium on Operating Systems Principles, New York, NY, USA (2003) 133–148
12. de Assunção, M.D., Buyya, R., Venugopal, S.: InterGrid: A case for internetworking islands of Grids. Conc. and Comp.: Pr. and Exp. (CCPE) **20**(8) (2008) 997–1024
13. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Transactions on Parallel and Distributed Systems **12**(6) (2001) 529–543
14. Jackson, D.B., Snell, Q., Clement, M.J.: Core algorithms of the Maui scheduler. In: 7th JSSPP, London, UK (2001) 87–102
15. Iosup, A., Epema, D.H.J., Tannenbaum, T., Farrellee, M., Livny, M.: Interoperating Grids through delegated matchmaking. In: 2007 ACM/IEEE Conference on Supercomputing, Reno, USA (2007)
16. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. J. Par. and Dist. Comp. **63**(11) (2003) 1105–1122
17. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: Job Scheduling Strategies for Parallel Processing, London, UK, Springer-Verlag (1997) 1–34
18. Islam, M., Balaji, P., Sadayappan, P., Panda, D.K.: QoPS: A QoS based scheme for parallel job scheduling. In: 9th JSSPP, Seattle, USA (2003) 252–268
19. Hanke, J.E., Reitsch, A.G.: Business Forecasting. 5th edn. Prentice-Hall, Inc., Englewood Cliffs, USA (1995)