Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid *

Marek Wieczorek, Mumtaz Siddiqui, Alex Villazón, Radu Prodan, Thomas Fahringer
Institute of Computer Science, University of Innsbruck
Technikerstraße 21a, A-6020 Innsbruck, Austria
{marek,mumtaz,avt,radu,tf}@dps.uibk.ac.at

1 Abstract

In this paper we present an extension to devise and implement advance reservation as part of the scheduling and resource management services of the ASKALON Grid application development and runtime environment. The scheduling service has been enhanced to offer a list of resources that can execute a specific task and to negotiate with the resource manager about resources capable of processing tasks in the shortest possible time. We introduce progressive reservation approach which tries to allocate resources based on a fair-share principle. Experiments are shown that demonstrate the effectiveness of our approach, and that reflect different QoS parameters including performance, predictability, resource usage and resource fairness.

2 Introduction

Scheduling and resource management play a key role for effective execution of applications on the Grid. A scheduler is used to make a plan for the execution, determining which tasks should be executed on which resources and in which order. Resource management is responsible for services such as registration and provision of resources. The scheduler optimizes a goal function, usually execution time or economic cost. Created in this way schedules are subsequently executed.

Highly dynamic and unreliable Grid environments make any assumptions concerning resource availability and execution time extremely difficult. Therefore, creation of a reliable schedule and proper reservation of resources constitute one of the largest challenges in Grid computing. Grid environments cannot usually guarantee that execution requests will be fulfilled within expected time intervals. Moreover, requests coming from other users can always disrupt already taken scheduling decisions. Time-critical applications, which are an important class of Grid applications, cannot be effectively executed without any guarantee for the expected execution time. Advance reservation of resources provides a possible solution to this problem. A user can reserve resources, in order to be sure that within a requested period of time resources will be available.

This paper investigates the impact of resource reservations on different aspects of application execution, represented by different criteria comprising execution time, predictability, resource usage and fairness. Predictability can be considered as the most important criterion, because it has a substantial impact on the execution of time-constrained applications. We have chosen scientific grid workflow applications as the class of computational applications to validate our approach. Workflow applications consist of execution tasks, data transfer tasks, and dependencies between them.

The rest of the paper is organized as follows. In the next section we describe briefly the related work on advance reservations for the Grid. Later, we present the architecture of our Grid environment, focusing on scheduling and resource management. Section 5 and 6 provide details of our reservation and scheduling services respectively. Section 7 describes the applied experimental methodology, and the methods used to evaluate the results. Results and discussion are presented in the Section 8. The last section concludes the paper and discusses future work.

3 Related work

The Maui [8] scheduler is an advanced job scheduler for cluster systems in which an advance reservation scheme enables future allocation of local resources. On a Grid site, Maui can be used in combination with other Local Resource Managers (LRM), such as PBS [15], SGE [14], LSF [7]. Maui does not provide negotiation mechanism at resource and Grid level, and requires the Grid job submission service to be modified in order to be reservation aware.

GARA [6] provides an interesting proposal for an architecture supporting advance reservation and co-allocation of resources in large Grid environments incorporating resources of widely varying types (computers, networks, disk, memory, etc.). However, workflow applications are not covered by this work, nor does the system provide any practical solution to the problem of fairsharing.

Thomas Roebliz et al. present an elastic Grid reservation

*This work is partially funded by the European Union through the IST-2002-511385 KWf-Grid and IST-034601 Edutain@Grid projects.

with user defined optimization policies and co-reservation with a concept of virtual resources [11]. No negotiation or scheduling mechanisms are supported.

Elmroth and Tordsson [4] propose a resource broker supporting advance reservations, developed in the NorduGrid project [1]. They present a simple approach based on modified basic Grid middleware services (extended GridFTP client and server are used to make the reservations). This approach does not consider workflow applications but simple execution requests, and no special scheduling algorithm is applied.

Theoretical analysis and experimental results for advance reservation and scheduling of Grid workflows is presented in [9]. The authors prove usefullness of the reservations, and propose a reservation architecture based on "the predictability trinity": Grid Scheduling, Reservation Fabric, Performance Models. The problems of resource usage and fairness are not covered. The experiments are performed for simple workflows, composed of 3 sequential tasks only.

An interesting reservation approach and performance analysis for execution simulation of workflow DAGs on the Grid is presented in [13]. For scheduling, the authors applied a modified version of the Earliest Time First algorithm. The algorithm does not optimize the execution order of independent tasks within the same workflow.

In contrast to other existing approaches, we provide a flexible customizable negotiation mechanism for reservations. We compare two different reservation approaches, among them the *progressive* approach proposed by us, which shows promising results with respect to different evaluation metrics. We present an extensive experimental analysis of the workflow execution based on our own scheduling approach, applying diverse metrics (execution time, predictability, resource usage, fairness), and analyzing complex scientific workflows.

4 Architecture

In previous work we have developed the ASKALON Grid application development and computing environment [5] which provides high-level mechanisms for composing and executing scientific workflows, with the ultimate goal of providing the application developers with an invisible Grid by hiding as many low-level technology details as possible. The XML representation of a composed workflow is given to the ASKALON middleware services for scheduling and reliable execution on the Grid.

Our Grid middleware separates the planning of a workflow from its actual execution. The *workflow planning* is performed by a negotiation process between: (a) individual scheduler service instances (acting on behalf of the users who submit the workflows), and (b) the resource manager service which ensures the advance reservation of resources. The general architecture of our system is depicted in Fig. 1.

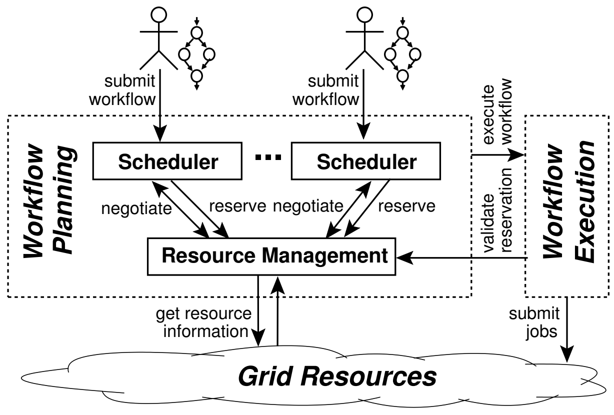


Figure 1. Architecture of the system

Both the scheduler instances and the resource manager try to maximize their own profit, which are potentially in conflict, i.e. each scheduler instance tries to get as many resources as possible to execute its own workflow (without taking into account the other users), and on the other hand, the resource manager tries to minimize the resource usage fairly distributed among the users. The negotiation leads to a final set of resources which are reserved for the execution of the workflow handled by the scheduler. The resource model applied by us assumes a distributed collection of parallel computers as Grid sites. Every Grid site is defined by a set of processors each of which is an atomic unit.

The *workflow execution* service is responsible for the actual execution of the planned workflow. It uses the reserved slots that were allocated by the resource manager during the planning phase. The workflow jobs submitted to the Grid through a job submission service such as Globus WS-GRAM [2] must be validated in order to check if the requested slots were actually reserved by the user.

5 Advance reservation

The Grid resource management mainly deals with on-demand provision of Grid resources, and the Quality-of-Service (QoS) can be improved significantly with the help of advance reservation. An advance reservation is a limited or restricted delegation of a particular resource capability over a certain time interval which is achieved through a negotiation process between requester (scheduler) and provider (resource manager). It enhances the predictability of the Grid and leads towards a Grid with smart middleware having better control over the underlying resources. Furthermore, advance reservation enables Grid resource management to optimize different QoS parameters and enhance resource utilization with better capacity planning [12].

We introduced two new algorithms i.e. *attentive* and *progressive* as an extention of our existing work about Grid capacity planning for optimized QoS [12], which provides a flexible mechanism to plug in different offer generation algorithms. In the offer generation process, the RM tries to maximize provider's interests while fulfilling client's requirements. Free slots which are in accordance to the reservation policy are offered on request.

The *attentive* algorithm offers requested slot only if it is available, otherwise it generates alternative offers according to the available slots closer to the requested QoS parameters, for instance, the timeframe. While generating alternative offers, it tries to keep the reserved segments as minimum as possible, by proposing alternative options which are overlapping or adjacent to the existing reserved slots, i.e. it tries to find slots during or immediately before or after an existing reservation for all the available processors (P_{total}) in a Grid site. Fig. 2 shows an example with a resource with 2 processors and 5 reservations (solid boxes) and what happens when another slot is requested. Dotted boxes show possible alternative options which are generated and offered to the client by the attentive algorithm (offers before the requested timeframe are also generated, if possible). It is then up to the client to select one of the offers or re-negotiate.

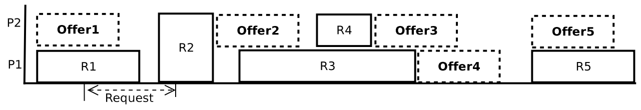


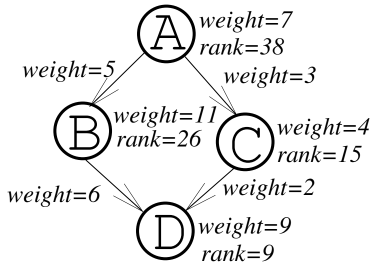
Figure 2. Possible options offered by attentive algorithm when a node with two processors was occupied by five confirmed reservations.

Offers can be generated by automatically adjusting the timeframe and other constraints (e.g. processors) in such a way that the overall requested QoS is optimized. For this purpose we propose adjustment like: change 10% of the requested duration by adding or removing one processor, or adjust the requested timeframe by scaling the attributes according to the speedup, which can be defined using different theoretical models or using a database with application benchmarks. The allocations for multiprocessor jobs is possible, though main focus of this paper is scheduling sequential jobs using advance reservations.

The *progressive* algorithm is an extension of the attentive algorithm, that considers fairness as well. It attempts to fairly distribute available resource capacity among competing clients instead of allowing a single client to reserve the entire capacity available on a Grid site. Having said that, the minimal time when the reservation can be established depends on the number and duration of reservations already made by the client. This is done by introducing a new restriction on the number of processors of a Grid site ($P_{allowed}$) which can be offered to a user at the same time. In the future we plan to cover other parameters like number of allowed reservations during a certain time period. The progressive approach provides a more fair distribution policy, so that several clients can get better chances to obtain offers that fit their requirements.

6 Reservation-based Scheduling

The ASKALON scheduler is a best-effort Grid workflow scheduler, adapted to apply different algorithms which can be used by the service as interchangeable plug-ins. To support advance reservation, we applied some modifications to the Heterogeneous Earliest Finish Time (HEFT) algorithm that proved to be an efficient heuristic for our real workflow's execution on the Grid [5].



	R1	R2	R3	*avg*
A	5	8	8	7
B	9	13	11	11
C	3	4	5	4
D	7	10	10	9

execution times on different resources

	R1–>R2	R1–>R3	R2–>R3	*avg*
A–>B	6	4	5	5
A–>C	4	2	3	3
B–>D	7	4	7	6
C–>D	1	1	4	2

data transfer time between different resources

Figure 3. Weights and ranks calculated with HEFT algorithm

HEFT is a list scheduling algorithm adjusted for heterogeneous environments. The algorithm consists of three distinct phases: the *weighting phase*, the *ranking phase*, and the *mapping phase*. During the weighting phase, which is adjusted to heterogeneous Grid environments, weights are assigned to the tasks and dependencies of the workflow that estimate the time cost of individual operations (i.e., task execution and data transfer time). If there is no data transfer time associated with a dependency, the weight assigned to the dependency is equal to zero. Let $w = (AS, DS)$ denote a workflow application, where AS represents the set of tasks, and DS the set of dependencies. The weight associated to a task $N \in AS$ is calculated as the average value of the predicted execution times $T_N(R)$ on every individual resource R (i.e. processor) available on the Grid:

$$\overline{W}(N) = \underset{\forall R \in Grid}{avg} \{T_N(R)\}, \forall N \in AS.$$

In the example depicted in Fig. 3, the Grid consists of three processors and, therefore, the weight of the task A is calculated as:

$$\overline{W}(A) = \frac{T_A(R1) + T_A(R2) + T_A(R3)}{3} = \frac{5+8+8}{3} = 7.$$

Similarly, the weight associated to a data link is calculated as the average of the predicted data transfer time on all interconnection links available between Grid sites. The ranking phase is performed by traversing the workflow graph upwards, and assigning a rank value to each task. The rank value of a task is equal to the weight of the task plus the maximum rank value of the successors, including the communication to the current task (if any):

$$\overline{R}(N) = \max_{\forall(N,Succ)\in DS}\{\overline{W}(N)+\overline{W}(N,Succ)+\overline{R}(Succ)\}.$$

For example, the rank of the task A is calculated as:

$$\overline{R}(A) = max\{\overline{W}(A)+\overline{W}(A,B)+\overline{R}(B), \overline{W}(A)+ \overline{W}(A,C)+\overline{R}(C)\} = max\{7+5+26, 7+3+15\} = 38.$$

The list of workflow tasks is then sorted in a descending order according to their ranks, i.e. A, B, C, and D. Finally in the mapping phase, the ranked tasks are mapped onto the processors that deliver the earliest completion time, i.e. A onto $R1$, B onto $R1$, C onto $R3$, and D onto $R1$.

In our model, we assume that data transfers are considerably less time consuming than execution tasks, and therefore as data transfer time estimations for scheduling we can simply apply default constant values (of the order of seconds or few minutes) rather than any reliable predictions. This approach is clearly not sufficient for many data intensive applications, but it covers a large class of computationally intensive applications with considerably small amounts of data being transferred (few megabytes per one data transfer), executed in fast network environments.

To support advance reservation, we modified the *mapping* phase by assigning to each task a list of resources instead of a single resource only. The list is ordered according to the increasing completion time, so that the first element on the list is always the most optimal one. For example, the resource order for the first task A is: $\{R1, R2, R3\}$.

We introduced to the algorithm a new fourth phase which we called the *reservation phase*. In this phase the scheduler negotiates with the resource manager in order to reserve exactly one resource from the list for each task (see Alg. 1). The reservations are performed in a retry loop which handles situations when a negotiation fails for some reason (e.g., the offer is timed out or taken by another user). By invoking `negotiate()`, the scheduler obtains initial reservations, from which it accepts one by invoking `reserve()`. The `negotiate()` function represents one of two alternative reservation strategies described in Section 5.

As the output, the scheduler returns an execution plan (schedule) and reservations performed for each of the tasks. The reservations are assigned to the users and not to the tasks, which means that the user can execute any of his tasks in any of the timeslots reserved for him. Such execution will still be valid from the viewpoint of the reservation service.

Algorithm 1 Scheduling algorithm: reservation phase

```
Input:
  $T$ - set of all workflow tasks
  $L_t$ - ordered lists of resources for all tasks $t \in T$
Output: $R$ - list of reserved resources

for $t \in T$ do
  $best\_time_E := \infty$;
  $time_B := earliest\_begin\_time(t)$;
  {Negotiate with Resource Manager}
  $is\_reserved := false$;
  while ! $is\_reserved$ do
    for $r \in L_t$ do
      $time := execution\_time(t, r)$;
      $time_E := time_B + time$;
      $(time'_B, time'_E) := negotiate(time_B, time_E)$;
      if $time'_E < best\_time_E$ then
        $best\_time_B := time'_B$;
        $best\_time_E := time'_E$;
        $best\_r := r$;
      end if
    end for
    {Reserve the resource}
    if $best\_time_E <> \infty$ then
      $is\_reserved :=$
       $reserve(t, best\_r, best\_time_B, best\_time_E)$;
    end if
  end while
  $R := R \cup \{(t, best\_r)\}$;
end for
```

7 Experimental methodology

We base the performance prediction of workflow tasks on trace data originated in previous real-world experiments performed on the Austrian Grid testbed [10] which consists of a set of Grid sites, each one being a parallel computer or a cluster. For our experiments we considered 5 grid sites distributed over the state of Austria (Table 1), comprised of 80 CPUs in total.

Table 1. The Austrian Grid testbed.

Site	*# CPUs*	*Machine Type*
agrid1 (Innsbruck)	16	Intel Xeon cluster
hydra (Linz)	16	AMD 2000 cluster
agrid (Innsbruck)	16	P4 cluster
altix1.jku (Linz)	16	Itanium SMP
altix1.uibk (Innsbruck)	16	Itanium SMP
Total:	80	

In our scheduling experiments, the execution of the workflows is done by increasing or decreasing the actual execution time randomly up to 50% from the predicted values (for possible inaccurate predictions, job queuing system delays, security latencies, external load on parallel computers, etc.). Therefore, the workflow execution does not necessarily follow the schedule and the reservations. We simulated the execution considering all reservations performed by all users, not allowing any task to use a resource if it is reserved by another user. An execution can be performed by a user on a resource at a time only when one of the following conditions is true: the resource is reserved by the user, or the resource is neither reserved nor used at all at this time. We applied a far-sighted policy, trying to predict in advance if the tasks can be submitted. A task cannot be submitted if it has no chance to finish within the time available for the user. An alternative approach would be to submit tasks whenever a user is allowed to do it, and kill them and migrate when a foreign reservation starts. It would mean different execution conditions, because the same task would execute many times, and could consume more resource time than in the case of the far-sighted approach applied in our experiments.

We performed several experiments, in each of which we submitted workflow applications in sequence at the interval of 60 seconds. Reservations were performed for individual CPUs which gave the user exclusive access over the allocated time periods. The execution engine processed the workflow tasks according to the schedule, and simulated their execution, if all sufficient conditions were fulfilled. If a task could not be executed immediately, it was postponed a certain period of time (30 seconds) for later execution.

We compared execution times, resource usage, predictability and fairness measured for different executions, and for different scheduling and reservation strategies, using some metrics defined by us. We measured the resource usage U_R as a ratio between the aggregated time when resources were effectively used and the overall available execution time (number of available resources multiplied by the total time when all experiments were performed):

$$U_R = \frac{\sum_{A \in AS} t_A}{t_{total} \cdot |R|},$$

where $|R|$ represents the number of resources available.

As a measure for predictability, we calculated the relative overhead O_{pred} derived from inaccurate prediction as the difference between the actual execution time t_{act} and the predicted execution time t_{pred} divided by the predicted time. If the actual execution time is shorter, then O_{pred} is equal to zero:

$$O_{pred}(w) = \begin{cases} \frac{t_{act}(w) - t_{pred}(w)}{t_{pred}(w)} & t_{pred}(w) < t_{act}(w), \\ 0 & t_{pred}(w) \geq t_{act}(w) \end{cases}$$

The predicted workflow execution time t_{pred} is equal to the difference between the end time of the last reservation performed for the given workflow, and the end time of the scheduling algorithm that precedes the execution. If no reservations were made for the workflow (scheduling without reservations), then we use the predicted end time of the last task instead of the end time of the last reservation .

We evaluate the fairness calculating the standard deviation ρ of execution times over all workflows scheduled within an experiment, denoted W.

$$t_{avg}(W) = \frac{\sum_{w \in W} t_{act}(w)}{|W|}$$

$$\rho(W) = \sqrt{\frac{1}{|W|} \sum_{w \in W} (t_{act}(w) - t_{avg}(W))^2}$$

We will compare the results achieved with and without reservations. Different strategies will be examined for the scheduler and the resource manager. The scheduler can request for shorter or for longer periods of time. A *short request* reservation for a task is 20% longer than the predicted execution time for this task. A *long request* reservation for a task is 2 times longer than the predicted execution time. The resource manager applies either an *attentive* or a *progressive* reservation strategy (see Section 5), with $P_{allowed} = \frac{1}{4} P_{total}$, i.e. 25%. We assume that the Grid does not change during a whole experiment, which is quite realistic for applications with low communication rate, executed with advance reservations.

8 Results and Discussion

In this section, we discuss the results obtained when executing several workflows scheduled on the Grid with and without reservations. We use in our experiments a real-world material science application called WIEN2k [3]. The workflow of WIEN2k (Fig. 4) has 2 parallel sections with equal number of tasks determined by the parameter called *kpoints*, and an external while-loop. In our experiments we

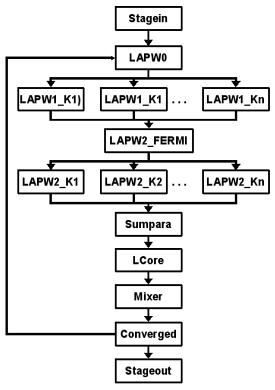
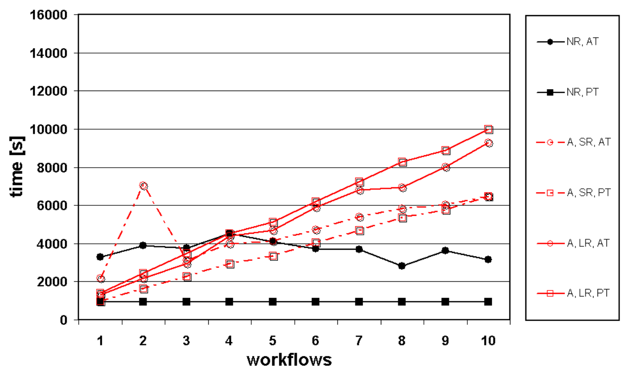


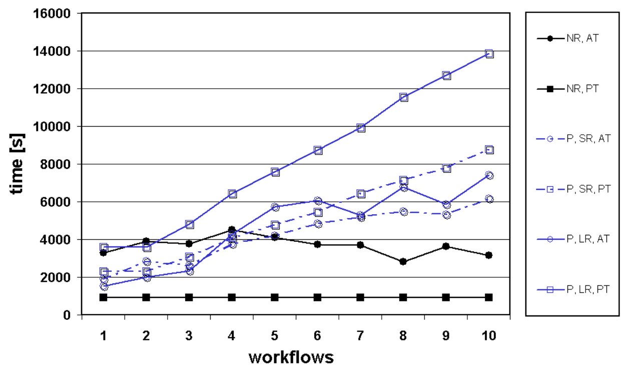
Figure 4. The WIEN2k workflow.

use a workflow with exactly 1 iteration of the while-loop and 80 parallel tasks in each parallel section.

Fig. 5-6 show the execution times of 10 workflows and compare the predicted and actual execution time. Fig. 5(a) compares the results achieved without reservations with the results achieved with attentive reservations, and Fig. 5(b) compares the results achieved without reservations with the

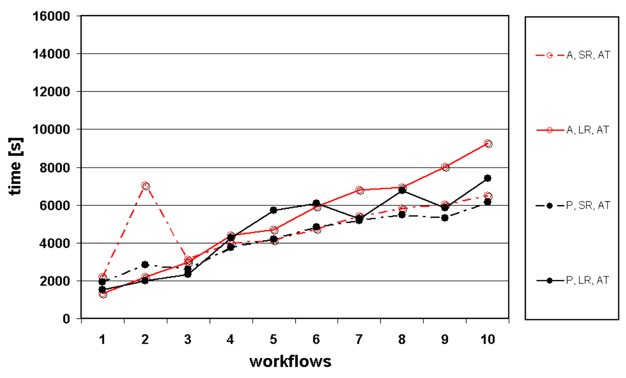


(a) Experiments without reservations compared with the experiments with attentive reservations

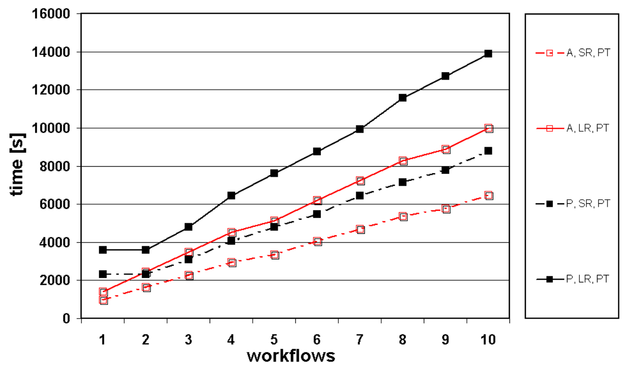


(b) Experiments without reservations compared with the experiments with progressive reservations

Figure 5. Comparison of execution times with and without reservations. [NR = No reservations, A = Attentive, P = Progressive, SR = Short requests, LR = Long requests, AT = Actual time, PT = Predicted time]



(a) Experiments with attentive reservations compared with the experiments with progressive reservations. Actual execution times



(b) Experiments with attentive reservations compared with the experiments with progressive reservations. Predicted execution times

Figure 6. Comparison of execution times with attentive and progressive reservations. [NR = No reservations, A = Attentive, P = Progressive, SR = Short requests, LR = Long requests, AT = Actual time, PT = Predicted time]

results achieved with progressive reservations. The workflows were scheduled using the different reservation approaches (no reservation, attentive and progressive), and with *short* and *long* requests, i.e. 20% and 100% longer than the predicted execution time, as described in Section 7.

This means that the first workflow could be scheduled in a basically empty Grid and get more chances to get all the resources necessary for its execution. For the subsequent workflows the environment becomes more and more *saturated*, i.e. fewer resources remain available for execution and reservations. The workflows executed at the end are scheduled in the most saturated environment. However, the last part of their execution can be performed in an almost empty environment, which results in an improved performance.

It can also be observed that up to the fourth workflow the execution time is shorter for the workflows scheduled with reservations (the exception for the attentive reservations with short requests is discussed later). This means that from low to moderately loaded environments it makes sense to introduce reservations to achieve better performance. In addition, without reservations the execution time is relatively high for the first workflows, but then it falls down for the workflows scheduled at the end. At the same time, the execution time for the workflows without reservation mostly increases when new workflows are started. This occurs because without reservations the execution is sensitive to the state of the environment at any moment between the begin and the end of execution, while in the case with reservations the most important is the moment when the reservations are performed. Therefore, with reservations the workflows scheduled at the beginning of an experiment have the best situation, while without reservations the workflows scheduled at the end of the experiment can achieve the best performance.

When comparing the attentive and the progressive reservation strategy (Fig. 6), we can see that in most of the cases the progressive strategy provides shorter execution times, even though the predictions for the progressive reservations are usually much larger. This can be explained because the progressive approach provides many gaps in the reservation schedule, allowing the executor to apply the backfilling strategy and execute many tasks out of the timeslots reserved for them. The outcome would be different if a less intelligent executor were applied.

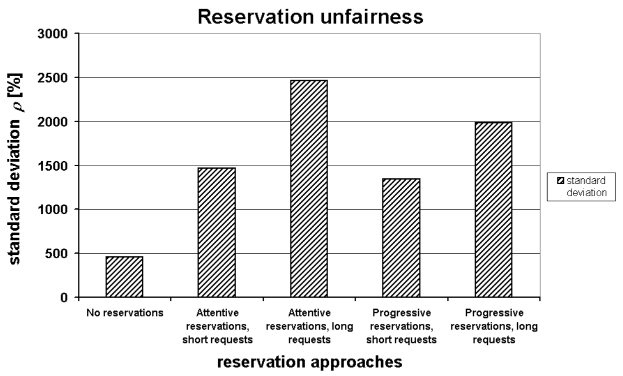


Figure 7. Standard deviation of execution time for different reservation approaches

Considering the fairness (see Fig. 7), as expected the progressive approach gives some benefits over the attentive approach. The value for ρ is smaller in all of the cases for the progressive strategy. It can be also observed in the curves in Fig. 6 that for the progressive strategy both the predicted and the actual execution times do not increase significantly for the first three scheduled workflows. It means that before the environment becomes saturated with reservations, the reservation service is able to allocate fair amounts of resources to all users.

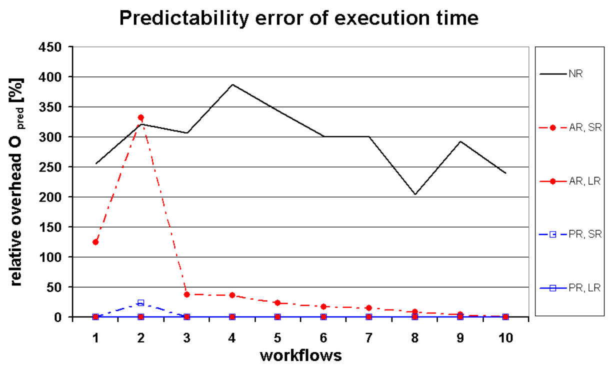


Figure 8. Comparison of predictability for different reservation approaches. [NR = No reservations, A = Attentive, P = Progressive, SR = Short requests, LR = Long requests]

Figure 8 compares the predictability of different reservation approaches, expressed using the notion of relative overhead O_{pred} defined in Section 7. The highest prediction error (200-400%) is obtained by the scheduling without reservations. The error could be lower if we calculated the prediction in a different way (not as the shortest possible execution time, as we do currently). It would be however very difficult to develop any reliable prediction strategy, since any expectations for the execution time depend strongly on the execution service (mostly on the queuing system). The attentive reservations with short reservation requests do not provide good predictability either. Attentive reservations fill the resource time tightly with reservations, and the margin of error is very small because of short requested times, therefore any exceeding of a timeslot reserved can be very costly in terms of the finish time of a workflow (see workflow 2). The progressive approach provides much better predictability, because the reservations do not fill the time space too tightly so that a delayed task can easily find an "emergency" slot. Long reservation requests eliminate entirely the problem of wrong predictability, and no tasks are delayed anymore. However, they cause very low resource usage, since a lot of reserved resource time is unused.

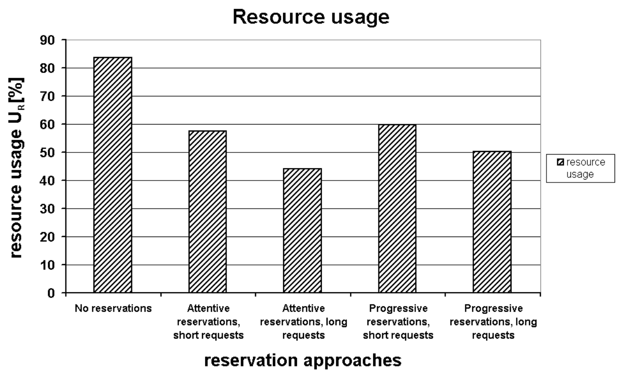


Figure 9. Resource usage for different reservation approaches. [NR = No reservations, A = Attentive, P = Progressive, SR = Short requests, LR = Long requests]

Resource usage (U_R) also differs significantly between different reservation strategies (see Fig. 9). The best usage (more than 80%) can be achieved for the approach without reservations. For the reservations with short reservation requests, the usage is more than 25% lower, both for the attentive and the progressive reservation policies. Applying the long reservation requests provides an additional loss of more than 10%, which decreases the resource usage to less than 50%.

Finally, we made a mixed experiment (see Table 2), comparing the scheduling performed without reservations (workflows 1, 3, 5, 7 and 9, first column in Table 2) with the scheduling performed with reservations (workflow 2, 4, 6, 8 and 10, progressive reservations with short reservation requests applied). As expected, every workflow with an even number $2i$ shows shorter execution time than its neighboring workflows with odd numbers $2i - 1$ and $2i + 1$ (except for the workflow 2), and much higher predictability. Comparing the results of the mixed experiment with the results of the previous experiments (see Fig. 5-9), we notice that the workflows scheduled with reservations improved their performance (again, except for workflow 2) and the workflows scheduled without reservations worsened their performance up to 50%. The mixed strategy presents a good fair-sharing, which is about 2 times better than for strategies with reservations only.

Table 2. Execution without reservations - odd numbers, and with progressive reservations with short reservation requests (20% longer than the predictions) - even numbers

Workflow	$t_{act}[s]$	$t_{pred}[s]$	$O_{pred}[\%]$
1	*4438*	*927*	*378.75*
2	**4464**	**2311**	**93.17**
3	*4993*	*927*	*438.57*
4	**2731**	**2907**	**0.0**
5	*5098*	*927*	*449.92*
6	**3020**	**2892**	**4.41**
7	*4415*	*927*	*376.23*
8	**3934**	**4081**	**0.0**
9	*4182*	*927*	*351.09*
10	**4354**	**4721**	**0.0**
U_R [%]	73.70	ρ	723.84

9 Conclusions and Future Work

This paper presents a scheduling and an advance reservation service for execution of scientific workflow applications on the Grid. We developed a resource management service supporting advance reservations, based on customizable negotiation mechanism. We proposed the *progressive* reservation approach which appears to be a good solution for Grid environments dedicated for execution of scientific workflows. We have implemented our approach in the ASKALON application development and execution environment. We demonstrated that advance reservation can have a major impact on execution time and can increase considerably predictability of a Grid environment. We also showed the importance of requesting for longer reservation time periods, because of the low reliability of execution time predictions. This may provide very high probability that the execution will finish within the reserved time. However, this happens at the cost of lower resource usage and fairness of resource distribution. Fairness can be increased by employing a special reservation strategy, for instance the progressive reservation strategy proposed in this paper, which gives the best results when the Grid is not saturated. Finally, we showed that the strategies without reservations and with progressive reservations can be applied together, and result in good performance and fairness.

In the future, we plan to investigate more dynamic reservation algorithms for heavily loaded Grid environments. We plan to study how to increase the resource usage and consider network traffic as an important aspect of workflow scheduling and execution.

References

[1] NorduGrid project. http://www.nordugrid.org/.

[2] WS-GRAM. http://www.globus.org/toolkits/docs/ 4.0/execution/wsgram.

[3] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. *WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties*. Institute of Physical and Theoretical Chemistry, TU Vienna, 2001.

[4] E. Elmroth and J. Tordsson. A Grid Resource Broker Supporting Advance Reservations and Benchmark-based Resource Selection. In *State-of-the-art in Scientific Computing.*, volume 3732 of *Lecture Notes in Computer Science*, pages 1077–1085. Springer Verlag, Oct. 2005.

[5] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. ASKALON: A Grid Application Development and Computing Environment. In *6th International Workshop on Grid Computing (Grid 2005)*, Seattle, USA, Nov. 2005. IEEE Computer Society Press.

[6] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36, London, UK, June 1999. IEEE.

[7] Load Sharing Facility. http://www.platform.com/Products/Platform.LSF.Family/.

[8] Maui Scheduler. http://www.supercluster.org/maui/.

[9] A. S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young. Making the Grid Predictable through Reservation and Performance Modelling. *The Computer Journal*, 48(3):358–368, 2005.

[10] A. G. project. http://www.austriangrid.at.

[11] T. Roeblitz, F. Schintke, and Wendler. Elastic grid reservations with user-defined optimization policies. In *Proceedings of the Workshop on on Adaptive Grid Middleware*, 2004.

[12] M. Siddiqui, A. Villazon, and T. Fahringer. Grid capacity planning with negotiation-based advance reservation for optimized qos. In ACM, editor, *International Conference for High Performance Computing, Networking and Storage (SuperComputing), SC06*, Tampa, Florida, USA, November 11-17 2006. ACM/IEEE.

[13] G. Singh, C. Kesselman, and E. Deelman. Technical report, University of Southern California, 2005.

[14] Sun Grid Engine. http://gridengine.sunsource.net/.

[15] The Portable Batch System. http://www.openpbs.org.