In Proceedings of 18th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2004), Santa Fe, New Mexico, USA, April 2004, 8 pages.

Queue Scheduling and Advance Reservations with COSY

Junwei Cao and Falk Zimmermann
C&C Research Laboratories, NEC Europe Ltd., Sankt Augustin, Germany
{cao, falk}@ccrl-nece.de

Abstract

Most of current job scheduling systems for supercomputers and clusters provide batch queuing support. With the development of metacomputing and grid computing, users require resources managed by multiple local job schedulers. Advance reservations are becoming essential for job scheduling systems to be utilized within a large-scale computing environment with geographically distributed resources. COSY is a lightweight implementation of such a local job scheduler with support for both queue scheduling and advance reservations. COSY queue scheduling utilizes the FCFS algorithm with backfilling mechanisms and priority management. Advance reservations with COSY can provide effective QoS support for exact start time and latest completion time. Scheduling polices are defined to reject reservations with too short notice time so that there is no start time advantage to making a reservation over submitting to a queue. Further experimental results show that as a larger percentage of reservation requests are involved, a longer mandatory shortest notice time for advance reservations must be applied in order not to sacrifice queue scheduling efficiency.

1. Introduction

Scheduling of parallel jobs on supercomputers and clusters has been an active research topic in the high performance computing community for over ten years [5]. Most current scheduling systems provide batch queuing support. One of the basic queue orders is first-come-first-served (FCFS), in which jobs are ordered by the arrival time. The backfilling technique is proven to be effective to improve scheduling performance with minor drawbacks and thus widely used.

With the development of metacomputing [11] and grid computing [7], users require access to multiple resources that may be distributed geographically. In general, a user would like to reserve all of the resources in advance for a distributed application so that corresponding quality of service (QoS) requirements can be guaranteed, e.g. the execution time.

COSY is a lightweight implementation of such a local job scheduler with plugs into SCore [19] and NEC MPI [12] environments. The current COSY implementation supports both queue scheduling and advance reservations. The COSY queue scheduling is accompanied by an aggressive backfilling mechanism that attempts to allocate currently unutilized nodes to jobs behind in the priority queue of waiting jobs without possible delaying the head of waiting jobs. Advance reservations with COSY can be used to guarantee an exact job start time or latest completion time. Users can query about a reservation and confirm it later within a period of time. The two phase commitment is designed for COSY to be utilized in a metacomputing or grid computing environment with QoS negotiation requirements.

When queue scheduling is combined with advance reservations, users especially with lower priorities may utilize advance reservations to gain start time advantages. If a user finds that an advance reservation can lead to an earlier job start time than submitting to the queue, he may choose to make reservations even though there is no explicit QoS requirements associated to the job. In a commercial environment, this can be prevented by charging more to advance reservations. In an academic / research environment, this has to be solved by applying proper scheduling policies.

In the COSY scheduler described in this work, the issue is addressed by applying a shortest notice time for each reservation. Only if the notice time is longer than the threshold can a reservation be accepted. The shortest notice time for a reservation is defined using the predictive wait time as if the reservation were submitted as a queue job. The prediction is based on historical information and defined using the mean wait time of queued jobs.

Experiments are designed in this work using a representative workload, which is generated using the Cirne and Berman archive [3] of parallel workload models included in [4]. Experimental results show that the existence of advance reservations still prolongs the queue wait time even though no start time advantages are taken. A longer shortest notice time must be applied for advance reservations in order not to sacrifice the queue efficiency. The more advance reservations are involved, the longer notice time constraint should be applied.

There are many other job scheduling systems, including Catalina [2], Condor [17], EASY [16], LoadLeveler [20], LSF [25], Maui [14], PBS [13], SGE [24], STORM [8] and Titan [23]. A good survey on the status of support for advance reservations within existing scheduling systems can be found in [18]. Features of several current systems that support advance reservations are summarized in this document, such as LSF, PBSPro, Maui, Catalina and SCAI/EASY. COSY is a lightweight implementation with essential job scheduling functionalities. COSY is most similar to the SCAI/EASY scheduler and currently does not have some advanced features, e.g. re-negotiation in Maui and web service interfaces in LSF.

Two most related work can be found in [21] and [22] in which the impact of advance reservations on queue scheduling is also investigated. In the simulation study described in [21], the percentage of queue jobs that can be delayed by a reservation request is taken as the parameter to decide whether the reservation can be accepted, while the mean wait time of queued jobs is utilized in our work. An advance reservation that can not be satisfied at the submission is arranged at the closest available time in [21] and rejected in our work. There is no system implementation associated with the simulation study included in [21], while our experiments are all carried out using the COSY implementation. The simulation study included in [22] is carried out using the Maui implementation. The work concludes that as the percentage of advance reservations increases, the overall resource utilization declines, which is also observed in our work. COSY provides solutions to satisfy advance reservations as far as possible without taking start time advantages and reducing queue scheduling efficiency, which is not focused in other work.

The rest of the paper is organised as follows: Section 2 introduces the COSY implementation briefly; in Section 3, the COSY scheduling issues are discussed in detail; experimental results are included in Section 4 and the paper concludes in Section 5.

2. COSY Structure

The COSY system implementation includes a front-end scheduling daemon and node daemons. Daemons are implemented using C++ that perform job scheduling and execution. Both command line user interfaces (in C++) and web-based GUIs (in Java) are provided. The COSY functional structure are illustrated in Figure 1 and described below respectively.

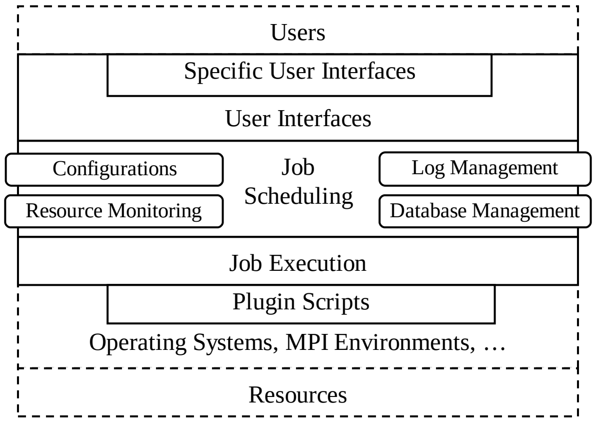

Figure 1. The COSY Structure

- *User Interfaces*. COSY provides several generic user interfaces for submitting a job, releasing a job and querying about job status and information. For those users who do not want to interact directly with the scheduling system, the COSY structure allows an implicit scheduling implementation. In this situation, users run parallel programs using specific user interfaces as if no scheduler were sitting in the middle. Generic user interfaces are called implicitly so that the execution can go through COSY and finally reach the corresponding plugin script. Currently COSY only provides implicit job scheduling for SCore and NEC MPI runs, and the structure allows straightforward integration with other environments in future.
- *Job Scheduling*. COSY scheduling algorithms and polices are described in detail in Section 3. The scheduling is performed with support of several other modules. When the scheduling daemon is started up, basic configurations on queues and resources are loaded from a predefined XML file. Resource monitoring is carried out periodically to check node availability so that the scheduler can adapt to the scheduling space dynamically. Scheduling processes are logged and all information is restored in database. In case the scheduling daemon fails accidentally, errors can be traced and user requests can be recovered with log and database support.
- *Job Execution*. COSY supports both batch and interactive jobs. COSY job execution is guided by the scheduling information and responsible for actual assigning / releasing node access to the corresponding user when a job is started / finished. For batch jobs, COSY calls batch scripts on behalf of the user. COSY takes care of the batch job execution by releasing resources for early completed jobs and

terminating uncompleted jobs when scheduled end time arrives.

- *Plugin Scripts*. COSY has to adapt to different parallel program execution environments. This is implemented by writing specific shell scripts to interface the COSY job execution to different environments. These scripts are specified implicitly in corresponding specific user interfaces as parameters of generic COSY submission.

While the COSY system implementation is introduced briefly above, this work focuses more on the scheduling algorithms and policies, especially when both queue scheduling and advance reservations are involved.

3. COSY Scheduling

COSY is a lightweight implementation of job scheduling systems with essential functionalities. Features of queue scheduling and supports for advance reservations in COSY are summarized in this section respectively.

3.1 Queue Scheduling

Queue scheduling with COSY is based on the first-come-first-served algorithm with aggressive backfilling mechanisms and includes following features:

- *Flexible node selection.* Apart from the number of nodes, COSY provides users with additional ways for node selection. Users can provide the list of preferred or disallowed nodes when submitting a job. In case nodes are heterogeneous they can be classified in COSY to different node sets. COSY allows users to specify preferred node sets so that nodes are firstly selected from given node sets. The XML configuration file includes all information of nodes and node sets. Flexible node selection is also applied to advance reservations.
- *Multiple queues & queue priority*. COSY job scheduling can support multiple queues with different priorities. A job submitted to a queue with a higher priority will be executed first (without backfilling). Job priority, resource limitation and access control are all defined within each queue and specified in the XML configuration file. For example, if a COSY job scheduler is configured with a *day* queue and a *night* queue and *day* has a higher priority than *night*, only after all jobs submitted to *day* are finished can jobs at *night* be executed. COSY allows different policies for different queues.
- *User priority*. Job priorities are associated with corresponding users. User priorities include four levels: *prefer*, *allow*, *defer*, and *deny*. Jobs are ordered within each queue according to user priorities. A user can have different priorities in different queues. For example, an external user may be *deferred* in *day* and *preferred* at *night*.
- *Resource limitation.* Resources allocated to one job can be limited in each queue. For example, the maximum time allocated to one job can be 1 hour in the *day* queue and 4 hours in the *night* queue so that users are encouraged to submit large batch jobs to *night* and allow more small and emergent jobs to be executed in *day*. Resource limitation also indicates memory and disk sizes of nodes so that a job that requires too much memory and disk can be rejected immediately.
- *Access control list.* Each queue is configured with a list of users or groups with different priorities. If a user submits a job to a queue without the access control authorization, the job will be rejected immediately.
- *Batch and interactive support*. COSY allows users to submit unlimited number of batch jobs but only one interactive job at one time. Advance reservations are only allowed to be submitted as batch jobs currently in COSY.

3.2 Advance Reservations

Advance reservations with COSY provide QoS supports of job execution. This is especially useful in a metacomputing or grid computing environment, where users or applications require coordinating multiple related jobs at multiple sites.

- *Exact start time*. Advance reservations in COSY are currently not associated with any priority management. COSY may accept a reservation or reject it immediately. COSY can guarantee an accepted advance reservation to be started exactly at the scheduled start time. This means a scheduled reservation will not be changed due to queue jobs.
- *Latest end time*. Sometimes users may require COSY to meet a deadline before which the job must be finished. The job is scheduled to be completed exactly at the deadline (rejected if not possible) and later may be rescheduled to earlier possible nodes.
- *Two phase commitment*. Some applications in a metacomputing or grid computing environment include QoS negotiation processes. An application may book resources for a job at multiple sites simultaneously and finally confirm one of them with

the best QoS support. In this situation, local job schedulers are required to support the two phase commitment, which is implemented in COSY by introducing an additional parameter when a reservation request is submitted. In a default mode, an advance reservation is booked and confirmed immediately if accepted. With the additional parameter, users can choose to book a reservation and confirm it later. If a reservation is not confirmed within a certain period of time, say 5 minutes, COSY will release the schedule automatically without actual execution.

When advance reservations are implemented in a queue scheduling system, it is obvious that users will soon find that advance reservations can be utilized to take start time advantages. This could result that advance reservations are abused and all policies for queue scheduling would make no sense. In a commercial environment, this can be avoided by charging more to advance reservations. In an academic / research environment, necessary scheduling polices should be applied. We currently focus on the latter situation.

4. Performance Evaluation

Experiments introduced in this section for performance evaluation of different scheduling policies are driven by solving the following problems:

- *Problem I*. How to prevent advance reservations from taking start time advantages over queued jobs in an academic / research environment?
- *Problem II*. How to satisfy advance reservations as far as possible without sacrificing queue efficiency, e.g. increasing wait time of queued jobs?

4.1 Performance Metrics

COSY scheduling performance is evaluated using the following metrics in our experiments:

- *Mean wait time*. The average amount of time that jobs have to wait before being executed.
- *Resource utilization*. The average percent of node time that is utilized by jobs.
- *Rejection rate*. The ratio of the number of advance reservations that are rejected to that of all jobs involved during a certain period of time.

The mean wait time can be applied to both queued jobs and advance reservations and we are particularly interested in the interaction between them. Resource utilization is a common performance metrics that is also used in other job scheduling research, e.g. [21] and [22]. The advance reservation model in our work is different from that described in [21]. Since COSY allows rejection of advance reservations, the rejection rate has to be considered in performance evaluation of the COSY scheduling.

4.2 Workload Model

A representative workload is important to evaluate job scheduling algorithms and policies. Too light or heavy workload may result into a situation where impact of different policies cannot be observed.

In this work, the workload used for all the experiments is the same and generated by the Cirne Comprehensive Model [3] using the Argonne National Laboratory (ANL) arrival pattern. The resource model is a 32-node cluster. For each experiment, 100 requests are generated that represent over one week’s actual operation time. The submission time, required time and number of nodes are retrieved from the generated workload file for each COSY request. COSY runs in normal mode with job execution disabled. In order to reduce experiment time, both request intervals and required time are reduced by 180 so that experiments can be accelerated without destroying the workload pattern and COSY can still support the reduced time granularity. Experiments are also simplified by assuming that only one queue is supported, all users have the same priority, two phase commitment for advance reservations is not involved, all nodes are homogeneous and belong to one node set.

4.3 Experiment Design

The scheduling policy investigated to solve *Problem I* in COSY is to apply a shortest notice time for each reservation request. Only if the notice time is longer than the threshold can a reservation be accepted. Since the impact of advance reservations on queue efficiency is evaluated using the mean wait time of queued jobs, this statistics can be feedback to COSY for the runtime configuration of the notice time constraint for each reservation. The main advantage is the decision making whether a reservation should be accepted or not can adapt to the workload of queued jobs in real time. For example, if the system load is very low, i.e. there are practically no jobs waiting in the queue, it is meaningless to make notice time constraints for reservation requests. On the other hand, if the queue is very long, advance reservations should be also postponed accordingly. The drawback is that the statistics have to be recalculated every time the COSY scheduling daemon receives a reservation request.

The implementation of the scheduling policy described above is straightforward in COSY. During each experiment, we randomly turn a certain percent of jobs into advance reservations. The exact start time of each reservation request equals to the shortest notice time plus the submission time. Note that the configuration guarantees that a reservation request will not be rejected directly due to the COSY shortest notice time constraint, but still cannot guarantee that all advance reservations can be accepted and scheduled, since resource requirements of different advance reservations are still possible to conflict with each other, which would result in rejections of later arriving requests.

Experimental results described later in this paper and those included in [21] show that though advance reservations do not take start time advantages, the existence of advance reservations still influences the queue efficiency. Further scheduling policies are required to solve *Problem II*. More experiments are designed to investigate whether the mean wait time of queued jobs can remain the same as if there were no advance reservations by applying a longer shortest notice time for each reservation request. This results that the mean wait time of advance reservations will be longer than that of queued jobs so that postponing advance reservations can compensate the loss of queue efficiency and result in a shorter queue wait time.

In summary, each experiment is carried out using the same workload with two parameters configured:

- *Percentage of advance reservations (p).* The percentage of advance reservations is increased from 5% to 20%.
- *Shortest notice time of advance reservations (n).* This is represented as times of the mean wait time of queue jobs. This can also be calculated as the ratio of mean wait time of advance reservations (w_a) to that of queue jobs (w_q) after experiments, because all advance reservations are configured with an exact start time that just meets the notice time constraint. During each experiment, this parameter is predefined approximately as 1, 2, 3, 4 … and refined using statistical results. Given a certain *p*, *n* is increased to reach a value when the corresponding notice time constraint can lead to the mean wait time of queued jobs that is as short as if there were no advance reservations.

After each experiment, three metrics are evaluated:

- *Relative mean wait time of queued jobs (q).* This is calculated using the mean wait time of queued jobs (w_q) divided by the value when no advance reservations are involved (w_q^0).
- *Resource utilization (u).*
- *Percentage of rejected advance reservations (r).*

4.4 Experimental Results

All experiment parameters and results are summarized in Table 1.

Table 1. Experimental Results

No.	*p*(%)	*n*	*q*	*u*(%)	*r*(%)
1	0	-	1	74	-
2	5	1.30	1.19	70	0
3	5	2.18	0.99	68	0
4	10	0.91	1.44	58	1
5	10	2.39	1.10	57	1
6	10	4.20	0.94	41	0
7	15	0.84	1.76	52	2
8	15	3.10	1.19	50	1
9	15	3.9	1.08	39	0
10	20	1.17	2.10	45	4
11	20	3.33	1.18	39	2
12	20	5.59	0.93	37	1

The first experiment is carried out without advance reservations. Since in this situation w_q equals to w_q^0, the relative mean wait time of queued jobs is 1. The resource utilization is 74% when no advance reservations are involved. Further results when different percentages of advance reservations are involved will be compared with these values. Statistical results are also illustrated in Figures 2, 4 and 5, respectively. Below we discuss the results in detail.

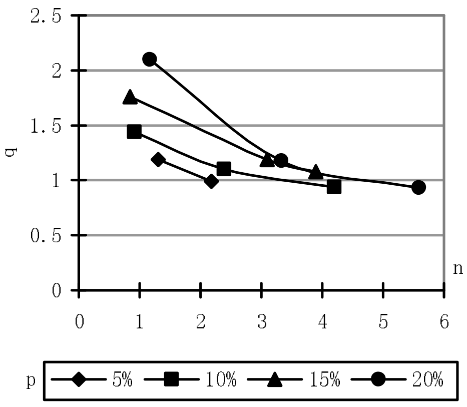

Figure 2. Experimental Results I: Relative Mean Wait Time of Queued Jobs (*q*) against Shortest Notice Time of Advance Reservations (*n*) with different Percentages of Advance Reservations (*p*) involved

Mean Wait Time

The mean wait time of queued job for each experiment is illustrated in Figure 2. It is apparent that the queue wait time increases with the number of involved advance reservations. This effect is strongly related to the added constraints at the time dimension so that the queue scheduling efficiency is decreased. It can be also noticed that in the case where *n* equals to 1, which means advance reservations do not take start time advantages, the *q* value is greater than 1 given any percentage of advance reservations involved, which means queue scheduling efficiency is decreased in any case. For example, the mean wait time of queued jobs is increased by 19% in exp. No. 2 when 5% of requests are advance reservations and 76% in exp. No. 7 when the percentage of advance reservations is 15%.

We are also interested in whether applying a longer shortest notice time to advance reservations can lead to better queue efficiency. It is apparent that as *n* increases, *q* does decline in any case. On the one hand, when advance reservations are configured with longer notice time constraints and, thus, have to be started later, queued jobs may start earlier. On the other hand, longer notice time constraints somehow enforce that advance reservations are scheduled with larger intervals, which provides more opportunities for queued jobs to be scheduled between these advance reservation intervals. When more advance reservations are involved, it will be more difficult to improve the queue efficiency. For example, in exp. No. 3 when 5% requests are advance reservations and the notice time constraint is configured with 2.18 times of mean wait time of queued jobs, queue scheduling is already as efficient as if there were no advance reservations. But as shown in exp. No. 9, in order to reach a similar efficiency when 15% requests are advance reservations, the notice time constraint has to be configured as long as 3.9 times of mean wait time of queued jobs.

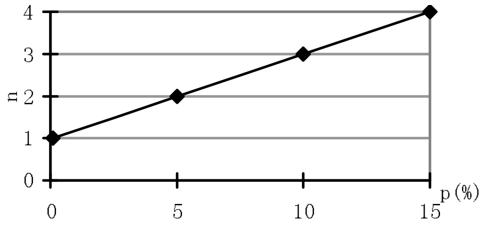

Figure 3. COSY Choice of Shortest Notice Time Constraints for Advance Reservations (*n*) when different Percentages of Advance Reservations (*p*) involved

In the COSY implementation, when a reservation request arrives, in order not to influence the queue scheduling, COSY does not only calculate the current mean wait time of queued jobs but also the current percentage of advance reservations. As described in Figure 3, shortest notice time constraints for advance reservations are represented as times of mean wait time of queued jobs and increase linearly from 1 to 4 as the percentage of advance reservations increases from 0% (not inclusive) to 15%.

In general, when the percentage of advance reservations is higher than 15%, it will be very difficult to carry out queue scheduling properly. This can be investigated by looking into the raw data of exp. No. 10. In this experiment, reservation requests arrive so frequently that some large queue jobs cannot be scheduled in any interval of these advance reservations. In real world systems, these jobs would have no chance to be scheduled and will wait for ever. If many queued jobs starve, it means the queue scheduling does not work at all. The aim of the COSY implementation is to satisfy advance reservations best possible without affecting queue scheduling efficiency. Consequently, if COSY detects a percentage of advance reservations of more than 15%, it will set an unlimited shortest notice time to advance reservations, stop receiving further reservation requests temporarily, and thus, guarantee a proper queue scheduling.

Note that the scheduling polices for not increasing queue wait time discussed above for advance reservations are effective only if a representative workload is utilized. When the workload is lighter or heavier, the result achieved in Figure 3 has to be adapted accordingly. This is beyond the scope of this paper.

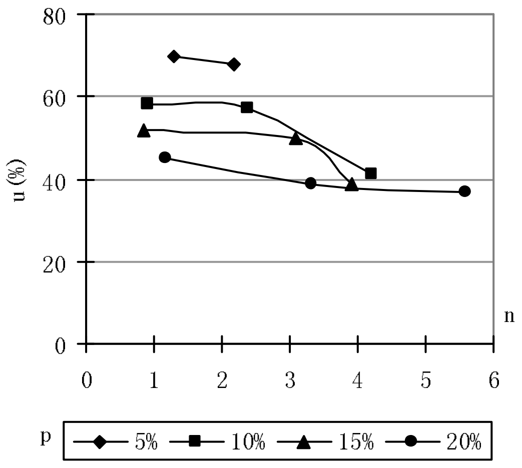

Figure 4. Experimental Results II: Resource Utilization (*u*) against Shortest Notice Time of Advance Reservations (*n*) with different Percentages of Advance Reservations (*p*) involved

Resource Utilization

As illustrated in Figure 4, when more advance reservations are involved, the average resource utilization rate decreases. This is also observed in the work described in [22]. It seems postponing advance reservations does not lead to an improvement of resource utilization but slightly reduces it, especially in the situation when advance reservations are largely postponed (e.g. exp. No. 6, 9 and 12). However further investigations of the raw data of these experiments prove that the statistics cannot reflect the situations of real world systems. Since experiments have to an explicit end point in time, in situations when advance reservations are postponing largely (e.g. 4-5 times of the mean wait time of queued jobs), some of the advance reservations are scheduled to be started very late when all queued jobs are already finished. Thus no queue jobs are left that could utilize the last idle resources caused by advance reservations. Since real world systems are continuously operational, there will be more requests that can continue to utilize these last idle resources. This issue is also discussed in the simulation study described in [21].

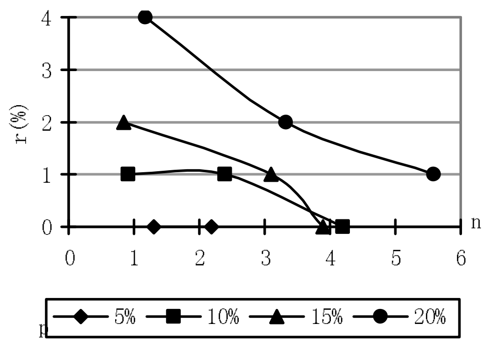

Figure 5. Experimental Results III: Percentage of Rejected Advance Reservations (*r*) against Shortest Notice Time of Advance Reservations (*n*) with different Percentages of Advance Reservations (*p*) involved

Rejection Rate

As illustrated in Figure 5, a very small amount of advance reservations are rejected. This reduces the comparability of experimental results because rejections lead that workloads of different experiments become different. However, since the percentage of rejected requests is relatively small, we assume that the impact is very limited.

We can also observe the fact that the more advance reservations are involved, the more are rejected due to an increase of conflicting (advance reservation) requirements. As the notice time constraint of advance reservations increases, the rejection rate decreases. This is because a longer notice time somehow enlarges the intervals of advance reservations, which therefore reduces likelihood of conflicts. Please note that the before mentioned decrease of the rejection rate is bound to the condition that all advance reservations satisfy the notice time constraint. In a real world system, longer notice time constraints lead to fewer accepted advance reservations, however, their chances for getting scheduled and not causing conflicts are quite good.

5. Conclusions

The COSY job scheduling system presented in this paper provides ordinary users with both queue scheduling and advance reservations. Queue scheduling is implemented with a backfilling mechanism. Advance reservations provide QoS support and two phase commitment. One of the interesting features is that COSY can be applied implicitly for those users who do not want to interact directly with the scheduling system.

In the work described in this paper, main research focuses on the performance impact of advance reservations on queue scheduling efficiency. Performance evaluation and experimental results included in this work conclude two solutions to the problems described at the beginning of Section 4:

- *Solution I.* COSY takes the current mean wait time of queued jobs as the shortest notice time requirement for reservation requests so that advance reservations cannot take start time advantages.
- *Solution II.* The mandatory shortest notice time of advance reservations is applied in COSY as 1 time of current mean wait time of queued jobs and increases to 4 linearly as the percentage of advance reservations increases from 0% (not inclusive) to 15% so that advance reservations can be satisfied as far as possible without queue wait time penalty. When the percentage of advance reservations is higher than 15%, COSY will reject reservation requests temporarily in order to guarantee a proper queue scheduling.

More experiments will be carried out with more realistic workload to verify above policies further. Future work on the COSY implementation will focus on the use of the COSY job scheduling system in a grid computing environment. These may include:

- Extension of COSY generic user interfaces with additional supports for standard languages (e.g. JSDL [15]) and protocols (e.g. GRAAP [10]) currently being defined in the Global Grid Forum (GGF).

- Integration of COSY with existing grid infrastructure software (e.g. GARA [6] and ARMS [1]) to enable grid level resource management.
- Implementation of COSY for ongoing grid testbeds to drive domain specific grid applications (e.g. GEMSS [9]).

The COSY support for two phase commitment is especially driven by the QoS negotiation requirement of medical simulation applications of the GEMSS project. More flexible and dynamic QoS supports for advance reservations with COSY will be implemented in near future.

Acknowledgements

The COSY development is partly funded by the European GEMSS project (EC/IST FP5 Project No. IST-2001-37153).

References

[1] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, "ARMS: an Agent-based Resource Management System for Grid Computing", Scientific Programming, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 135-148, 2002.

[2] Catalina, http://www.sdsc.edu/catalina.

[3] W. Cirne and F. Berman, "A Comprehensive Model of the Supercomputer Workload", in Proc. of 4th IEEE Annual Workshop on Workload Characterization, 2001.

[4] D. Feitelson, Parallel Workload Models, http://www.cs.huji.ac.il/labs/parallel/workload/models.html.

[5] D. Feitelson and L. Rudolph, Procs. of Workshops on Job Scheduling Strategies for Parallel Processing, http://www.cs.huji.ac.il/~feit/parsched/.

[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 Proc. of 7th IEEE Int. Workshop on Quality of Service, 1999.

[7] I. Foster and C. Kesselman, The GRID: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.

[8] E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin and S. Coll, "STORM: Lightning-Fast Resource Management", in Proc. of ACM/IEEE Supercomputing 2002.

[9] GEMSS, Grid Enabled Medical Simulation Services, http://www.ccrl-nece.de/gemss/.

[10] GRAAP-WG, Grid Resource Allocation Agreement Protocol, Working Group, Global Grid Forum, http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html.

[11] A. Grimshaw, J. Weissman, E. West, and E. Lyot, Jr., "Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems", J. of Parallel and Distributed Computing, Vol. 21, No. 3, pp. 257-270, 1994.

[12] R. Hempel, H. Ritzdorf, and F. Zimmermann, "Efficient Message Passing Interface Implementation for NEC Parallel Computers", NEC Research & Development, Special Issue on High Performance Computing, Vol. 39, No. 4, pp. 408-412, 1998.

[13] R. L. Henderson, "Job Scheduling Under the Portable Batch System", in Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing, 9th IEEE Int. Parallel Processing Symp., Santa Barbara, California, USA, Lecture Notes in Computer Science Vol. 949, pp. 279-294, 1995.

[14] D. Jackson, Q. Snell, and M. Clement, "Core Algorithms of the Maui Scheduler", in Proc. of 7th Job Scheduling Strategies for Parallel Processing, ACM SIGMETRICS 2001, Cambridge, Massachusetts, USA, Lecture Notes Computer Science Vol. 2221, pp 87-102, 2001.

[15] JSDL-WG, Job Submission Description Language, Proposed Working Group, Global Grid Forum, http://www.epcc.ed.ac.uk/~ali/WORK/GGF/JSDL-WG/.

[16] D. Lifka, "The ANL/IBM SP Scheduling System", in Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing, 9th IEEE Int. Parallel Processing Symp., Santa Barbara, California, USA, Lecture Notes in Computer Science Vol. 949, pp. 187-191, 1995.

[17] M. Litzkow, M. Livny, and M. Mutka, "Condor – a Hunter of Idle Workstations", in Proc. of 8th IEEE Int. Conf. on Distributed Computing Systems", San Jose, CA, USA, pp. 104-111, 1988.

[18] J. MacLaren, "Advance Reservations: State of the Art", Working Draft, Global Grid Forum, 2003, http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html.

[19] SCore. http://www.pccluster.org/.

[20] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The EASY-Loadleveler API Project", in Proc. of 2nd Workshop on Job Scheduling Strategies for Parallel Processing, 10th IEEE Int. Parallel Processing Symp., Honolulu, Hawaii, USA, Lecture Notes in Computer Science Vol. 1162, pp. 41-47, 1996.

[21] W. Smith, I. Foster, and V. Taylor, "Scheduling with Advanced Reservations", in Proc. of 14th IEEE Int. Parallel and Distributed Processing Symp., Cancun, Mexico, 2000.

[22] Q. Snell, M. Clement, D. Jackson, and C. Gregory, "The Performance Impact of Advance Reservation Meta-scheduling", in Proc. of 6th Job Scheduling Strategies for Parallel Processing, 14th IEEE Int. Parallel and Distributed Processing Symp., Cancun, Mexico, Lecture Notes Computer Science Vol. 1911, pp 137-153, 2000.

[23] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini and G. R. Nudd, "Local Grid Scheduling Techniques Using Performance Prediction", IEE Proceedings - Computers and Digital Techniques, Vol. 150, No. 2, pp. 87-96, 2003.

[24] Sun ONE Grid Engine Software (SGE), http://wwws.sun.com/software/gridware/.

[25] S. Zhou, "LSF: Load Sharing in Large-scale Heterogeneous Distributed Systems", in Proc. of Workshop on Cluster Computing, 1992.