A Simulation Infrastructure for Resource Allocation with Advanced Reservation in Global Grids

Anthony Sulistio Rajkumar Buyya

Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory,
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
ICT Building, 111 Barry Street, Carlton, VIC 3053
E-mail: {anthony, raj}@cs.mu.oz.au

Abstract

Grid applications can have large requirements and they need concurrent access to resources in order to perform their operations successfully. Grid resources also need to ensure to users that they can provide guaranteed access according to service level agreements between them. Advance Reservation (AR) for global grids solves the above problems by allowing users to gain concurrent access for their applications to be executed in parallel. AR also guarantees the availability of resources to users at the specified future times.

Evaluating various usage scenarios involving AR are not feasible to carry out on a real grid environment due to its dynamic nature. Therefore, in this paper, we present GridSim that provides a simulation infrastructure for repeatable and controllable evaluations. This paper discusses the design and implementation of AR within GridSim, together with experimental results to demonstrate its suitability and validity.

Keywords: advance reservation, grid simulation, and grid computing

1 Introduction

Grid computing has emerged as the next-generation parallel and distributed computing that aggregates dispersed heterogeneous resources for solving a range of large-scale parallel applications in science, engineering and commerce [1]. In most Grid scheduling systems, submitted jobs are initially placed into a queue if there are no available resources. Therefore, there is no guarantee as to when these jobs will be executed. This causes problems in parallel applications, where most of them have dependencies among each other.

Advance Reservation (AR) is a process of requesting resources for use at a specific time in the future [3]. Common resources who usage can be reserved or requested are CPUs, memory, disk space and network bandwidth. AR for a grid resource solves the above problem by allowing users to gain *concurrent access* to adequate resources for applications to be executed. AR also guarantees the availability of resources to users and applications at the required times.

AR is useful in eScience and eBusiness applications, such as Drug Design [4], Brain Analysis [5] and in a stock market environment. These applications share certain characteristics that require AR, such as generated data is available for only at certain periods, huge size of data and need for applications to be run in parallel.

A general scenario of using AR for these applications is described in Figure 1. A resource broker or a program on behalf of these applications can determine the patterns and frequencies of when these data are available from an instrument or a stock market, e.g. the data are ready by time T_5 (step 1). The broker reserves processing power at a supercomputing center at time T_{12} by taking into account the time at which the data is available, and the transfer time to move the data into the center (step 2). Next, the broker reserves network bandwidth between the supercomputing center and a data center to transport the results (step 3). Then reserves storage facility at the data center at time T_{40} (step 4), since the deadline is at time T_{45}. Finally, an analyst or a specialist can analyse them later on, e.g. at time T_{50} (step 5).

There are some systems that support AR capability, such as STARS [10], GARA [11] and Maui Scheduler [12]. However, to ensure the effectiveness of these scheduling system, all possible scenarios need to be evaluated. Given the inherent heterogeneity of a Grid environment, it is difficult to produce performance evaluation in a *repeatable* and *controlled* manner. In addition, Grid testbeds are limited

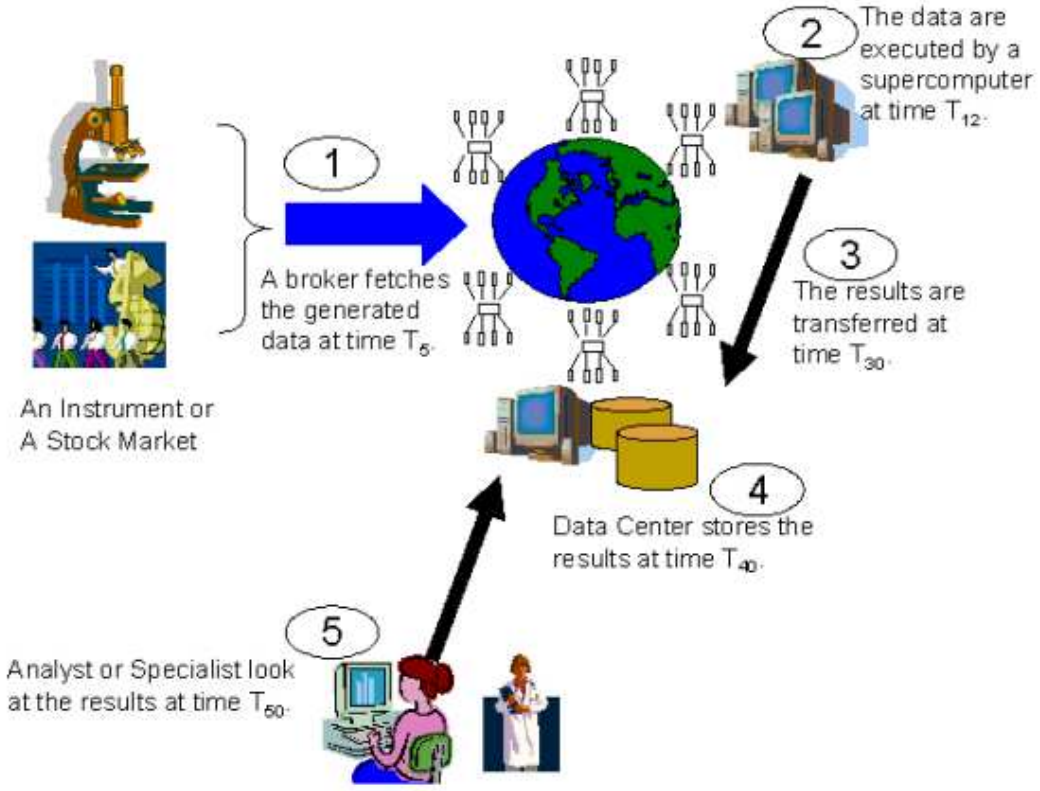

Figure 1. Scenario of using AR in different types of resources

and creating an adequately-sized testbed is expensive and time consuming. Morever, it needs to handle different administration policies of each resource. Therefore, it is easier to use simulation as a means of studying complex scenarios.

Some simulation tools available for application scheduling simulation in Grid computing environment, are Bricks [6], MicroGrid [7], SimGrid [8], and OptorSim [9]. However, none of them have the capability of simulating reservation-based systems. To address this weakness, we have developed AR mechanisms and incorporated them into the GridSim toolkit.

GridSim [2] is a Java-based discrete-event Grid simulation package that provides features for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. GridSim also has the ability to model heterogeneous computational resources of variable performance.

In this work, GridSim has been enhanced with the ability to handle: (1) creation or request of a new reservation for one or more CPUs; (2) commitment of a newly-created reservation; (3) activation of a reservation once the current simulation time is the start time; (4) modification of an existing reservation; and (5) cancellation and query of an existing resevation.

The rest of this paper is organized as follows: Section 2 presents the architecture of GridSim for AR. Section 3 discusses the implementation of GridSim. Section 4 shows the experiment of GridSim for simulating a Grid computing environment with AR functionality. Section 5 concludes the paper and suggests further work.

2 GridSim Design for Advance Reservation

2.1 Architecture Design

As mentioned earlier, GridSim is a Java-based discrete-event simulation package. To support AR capability, GridSim leverages Java's concepts, such as polymorphism and inheritance, to build a high-cohesion, low-coupling and extensible architecture.

Figure 2 shows GridSim's class diagram that relates to a reservation-based system. From this diagram, a Grid resource can be categorized into a reservation-based system (by creating `ARGridResource` object) or a non-reservation system (by creating `GridResource` object).

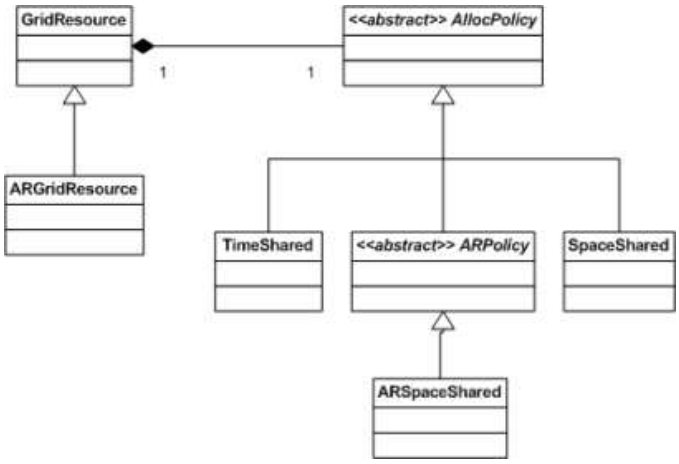

Figure 2. GridSim class diagram

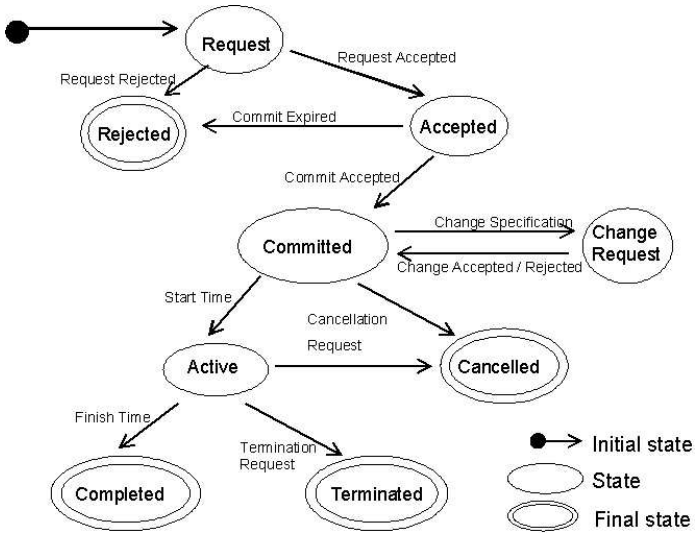

Figure 3. State transition diagram for advance reservation

As a design constraint, `GridResource` object contains only one scheduler of the type `AllocPolicy`. `GridResource` only acts as an interface between users and the local scheduler, and it is up to the scheduler to handle and to process submitted jobs. This approach gives the flexibility to implement various scheduling algorithms for a specific resource-based system.

Another advantage of this design is that adding a new scheduler does not require modification of existing resource and/or other scheduling classes. Creating a new scheduler is as simple as extending the `AllocPolicy` abstract class and implement the required abstract methods. If a scheduler needs to be able to handle AR functionalities, then it must extends from `ARPolicy` abstract class, and implement the required methods from both the `ARPolicy` and `AllocPolicy` classes. Abstract classes in this design are more appropriate than Java's interface class because there are common methods and attributes used by the child classes.

Currently, GridSim supports `TimeShared` (Round Robin) and `SpaceShared` (First Come First Serve (FCFS)) scheduling for a non-reservation resource. A reservation-based resource that uses FCFS scheduling will be discussed in more detail in Section 3.

2.2 States of AR

The design of AR in GridSim is influenced by recommendations from Global Grid Forum (GGF) draft [13] and Application Programming Interface (API) [14]. As a result, a reservation can be in one of several states during its lifetime as shown in Figure 3. Transitions between the states are defined by the operations that a user performs on the reservation. These states are defined as follows:

- **Requested:** Initial state of the reservation, when a request for a reservation is first made.
- **Rejected:** The reservation is not successfully allocated due to lack of enough free slots or an existing reservation has expired.
- **Accepted:** A request for a new reservation has been approved.
- **Committed:** A reservation has been confirmed by a user before the expiry time, and will be honoured by the resource scheduler.
- **Change Requested:** A user is trying to alter the requirements for the reservation prior to its starting. If it is successful, then the reservation is committed with the new requirements, otherwise the values remain the same.
- **Active:** The reservation's start time has been reached. The scheduler is now running or executing within the reservation slot.
- **Cancelled:** A user no longer requires a reservation and requests that it be cancelled.
- **Completed:** The reservation's end time has been reached.
- **Terminated:** A user terminates an active reservation before the end time.

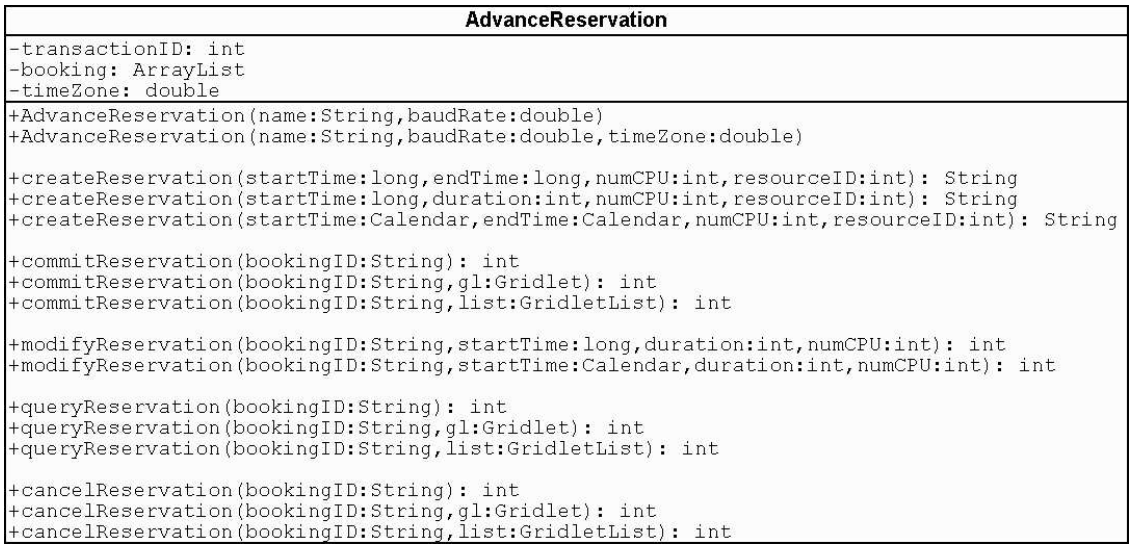

Figure 4. AdvanceReservation class diagram

2.3 GridSim API

The GridSim client-side API for AR is encoded in the method calls of the `AdvanceReservation` class as shown in Figure 4. In this class diagram, attributes and methods are prefixed with characters $+$, $-$, and $\#$, indicating access modifiers public, private and protected respectively.

The `transactionID` attribute is an unique identifier for a reservation, and is used to keep track of each transaction or method call associates with the reservation. The `booking` list is an important attribute to store reservations that have been accepted and/or committed. In the `AdvanceReservation` object, `timeZone` is a very important attribute as different resources are located geographically with different time zone. Hence, user's local time will be converted into a resource's local time when the resource receives a reservation.

For creating or requesting a new reservation, a user needs to invoke `createReservation()` method. Before running a GridSim program, an initialization of some parameters are required. One of the parameter is the simulation start time T_s. T_s can be a current clock time represented by a Java's `Calendar` object. Therefore, a reservation's start time needs to be ahead of T_s in either `Calendar` object or `long` representing time in milli seconds. Reservations can also be done immediately, i.e. the current time is being used as the start time with or without specifying a duration time.

If a new reservation has been accepted, then the `createReservation()` method will return an unique booking id in `String` object. Otherwise, it will return an approximate busy time in the interval of 5, 10, 15, 30 and 45 in time unit. The time unit can be in second or minutes or hours. If a request gets rejected, the broker, on behalf of the user, can negotiate with the resource by modifying the requirements, such as reducing number of CPUs needed or shortening the duration time.

Once a request for a new reservation has been accepted, the user must confirm it before the expiry time of this reservation by invoking the `commitReservation()` method. The expiry time is determined by the resource or its scheduler. The `commitReservation()` method returns an integer value representing error or success code.

Committing a reservation acts as a contract for both the resource and the user. By committing, the resource is obliged to provide CPUs at the specified time for a certain period. A reservation confirmation, as depicted in Figure 4, can be done in one of the following ways:

- committing first before the expiry time by sending a booking id. Once a job is ready, then committing it again with the job attached before the reservation's start time.
- committing before the expiry time together with a job if reserving only one CPU. In GridSim, a job or an application is represented by a `Gridlet` object.
- committing before the expiry time together with a list of jobs if reserving two or more CPUs. A `GridletList` object is a linked-list of Gridlet objects. This approach is highly-desirable for parallel applications as mentioned in Section 1.

According to the states of AR in Figure 3, a reservation that has been committed successfully, could be modified before its start time. This can done by invoking

`modifyReservation()` method, which returns an integer value representing error or success code. This method has similar parameters to `createReservation()` method, with the only difference is without the need to specify a resource id. Each booking id is unique to all resources and reservations.

The `queryReservation()` method aims to find out the current status of a reservation for one or more jobs, with one of the following result:

- `not committed`: reservation has not been committed.
- `not started`: reservation has been committed and not yet begun.
- `expired`: reservation has not been confirmed or committed within a specified expiry time.
- `active`: reservation has begun and is currently executing.
- `completed`: reservation is over, i.e. current time is more than the reservation finish time.
- `cancelled`: reservation has been cancelled.
- `terminated`: reservation has been terminated during execution.

Cancellation of a reservation can be done anytime before the completion time. The `cancelReservation()` method requires only booking id and returns an integer value representing error or success code. As with commitment and query of a reservation, cancellation can be done for one or more jobs.

3 Advance Reservation Implementation within GridSim

3.1 Queueing Model

As mentioned in Section 1, common resources that can be reserved are CPUs, memory, disk space and network bandwidth. In this paper, GridSim only focuses on reserving CPUs. Hence, an open queueing network model of a scheduling system is considered for implementing AR as in Figure 5. There is a finite buffer with size S to store objects waiting to be processed by one of P independent CPUs. m is the CPU speed, measured in the form of Million Instructions Per Second (MIPS) rating as per SPEC (Standard Performance Evaluation Corporation) [15] benchmarks. CPUs can be homogeneous or heterogeneous. For homogeneous ones, all m have the same MIPS rating. Different m exist for heterogeneous CPUs.

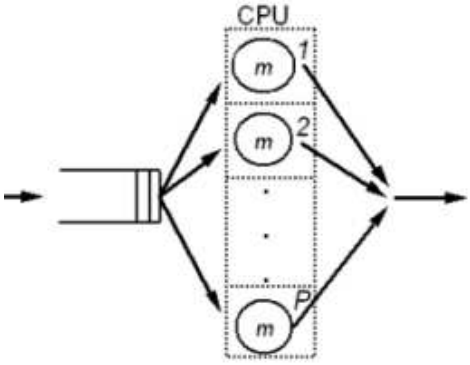

Figure 5. Queueing network model for a GridSim scheduling system

3.2 FCFS Scheduling System

GridSim uses FCFS approach for a reservation-based system for the queueing model as discussed earlier. For requesting a new reservation, Figure 6 shows the steps needed. A scheduler must find any potential conflicts with existing reservations or an empty slot for determining to accept/reject a new request. Before further explanation into the details of the algorithm for finding an empty reservation slot, the following parameters are defined:

- $List$: A reservation list. An accepted reservation is stored and sorted into this list based on its start time.
- $tempList$: A temporary list to store $List$ indexes.
- R_i: the i-th reservation object in $List$.
- R_{new}: A new reservation.
- $start_i$: Start time for R_i.
- $start_{new}$: Start time for R_{new}.
- $finish_i$: Finish time for R_i.
- $finish_{new}$: Finish time for R_{new}.
- CPU_i: Number of CPU required by R_i.
- CPU_{reserv}: Total number of CPU reserved by other reservations.
- CPU_{new}: Number of CPU required by a new request.
- $CPU_{resource}$: Total number of CPU owned by a resource.

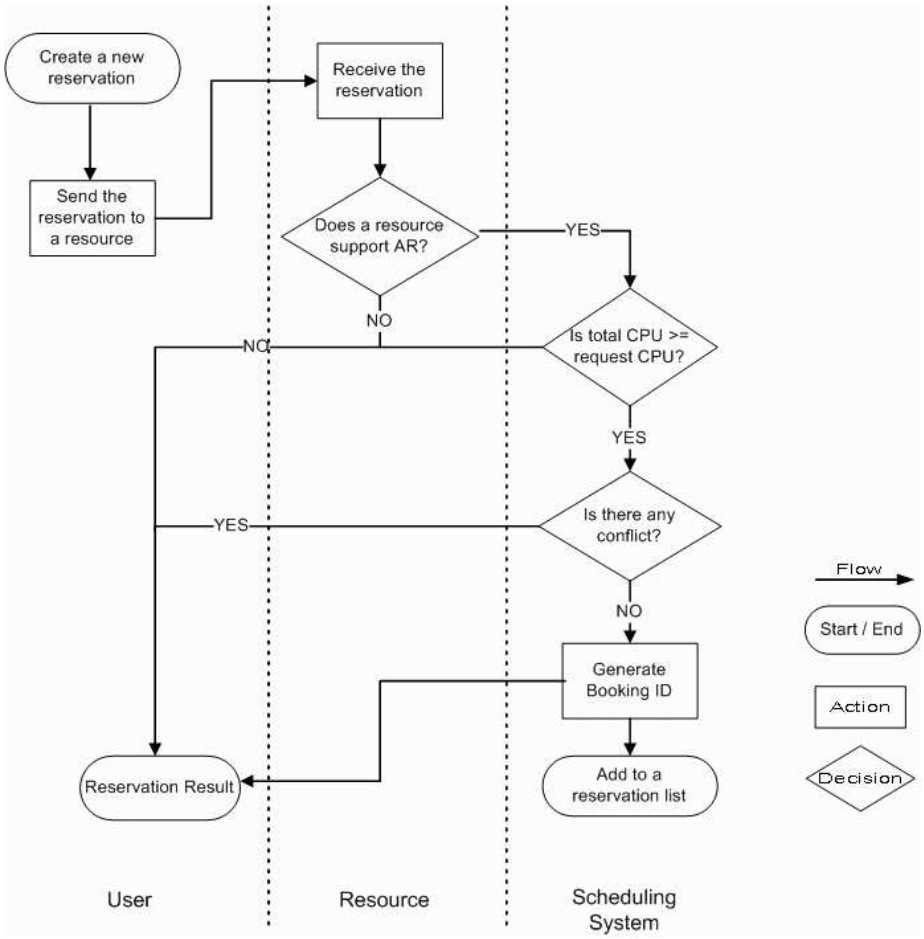

Figure 6. Flowchart for requesting a new reservation

Algorithm 1 explains on how GridSim's scheduler finds an empty reservation slot. Lines (1) to (2) are for a simple case, where no reservations exist. Hence, the scheduler will accept a new request straightaway. If $List$ is not empty, then line (4) to (9) finds reservations that might be conflicting with a new request. Line (6) handles a case where a reservation is within $start_{new} \longrightarrow finish_{new}$ interval. By storing indexes of all reservations that lay within this time interval, CPU_{reserv} can be calculated as in line (13).

Once a reservation has been accepted, it needs to be committed before an expiry time, as depicted in Figure 7. The overall sequence from a new reservation until the job completion is described in Figure 8. A `User` object communicates to `ARGridResource` object by sending a message. The same thing applies for returning a result or a value from `ARSpaceShared` object to `User` object. `ARGridResource` and `ARSpaceShared` objects are communicating through method calls since their relationship is a composition as mentioned in Figure 2.

4 Experimentation and Results

4.1 Experiment Setups

The experiment conducted models and simulates four resources with different characteristics, configurations and capabilities. Selected four resources mentioned in [2] are included in this experiment based on their location. A new cluster resource in our University is also modeled. Table 1 summarizes all the resource relevant information. The processing ability of these CPUs in simulation time units is modeled after the base value of SPEC CPU (INT) 2000 benchmark ratings published in [16].

This experiment simulates a scenario demonstrating GridSim's ability to handle AR functionalities. The following simulation setups are carried out:

- Created 5 resources, each is able to handle AR functionalities.
- Created 50 users, each with 100 Jobs.

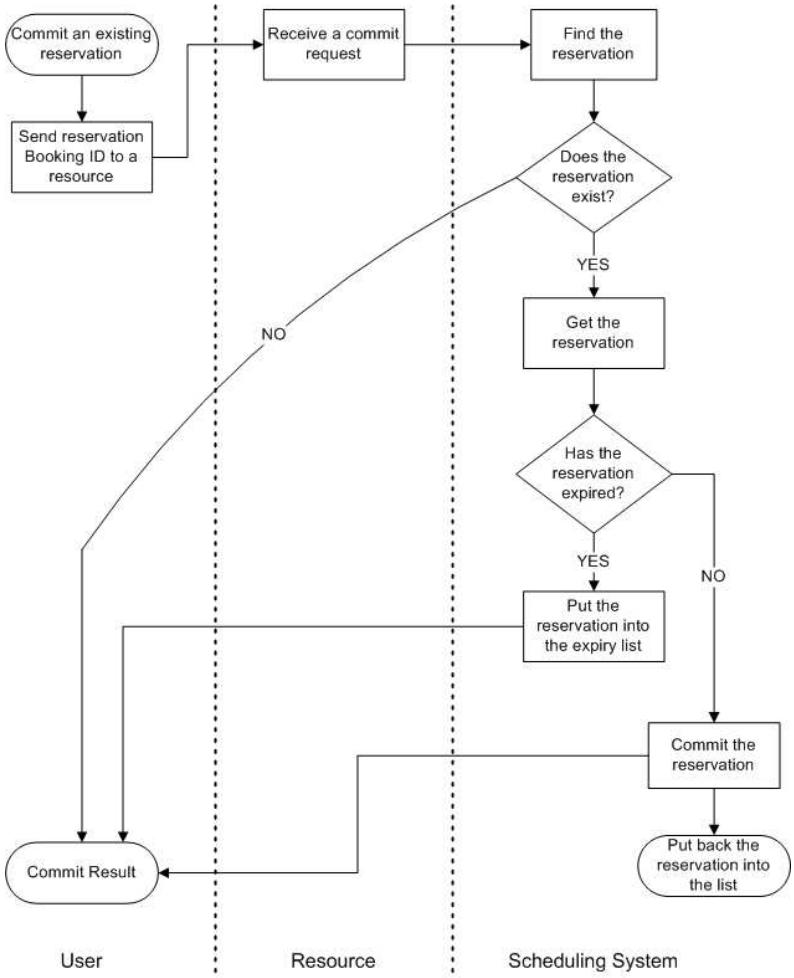

Figure 7. Flowchart for committing a reservation

- Job size is uniformly distributed in [500,000 ... 8,000,000] in Millions Instructions (MI) unit.
- Poisson distribution is used for job arrival time with average jobs for all users per time unit is 5.
- Assuming 1 simulation time is equivalent to 1 minute, maximum arrival time is 500 minutes.
- Sending jobs are uniformly distributed in $[R0 \ldots R4]$.
- Only concerns about scheduling jobs to empty CPUs. All resources and users are assumed to have same network bandwidth.
- A scheduler from each resource will start the job at the requested start time.

In addition, a reservation job can be `restartable` (paused if running longer, then resume execution from the point where it was paused if there is an empty CPU), and `non-restartable` (executing from start until finish or being pre-empted by a scheduler, and straightaway sending back to a user the finished or partially completed job). For this experiment, only 0%, 10% and 20% of total jobs using reservations and it follows similar approach done by [3].

The metrics used in this experiment are the Mean Waiting Time (MWT) and Mean Offset from requested reservation Time (MOT). MWT is the average amount of time that jobs have to wait before receiving resources. It examines the effect of a reservation-based system on a scheduler's performance. MOT is the average difference between users' initial start time to the actual guaranteed or obtained start time. It measures how well the scheduler performs at satisfying reservation requests.

In addition, this experiment is trying to determine the effect of estimating job's completion time on resource utilization and number of rejections to request a new reservation. For a non-restartable job, a reservation must have an

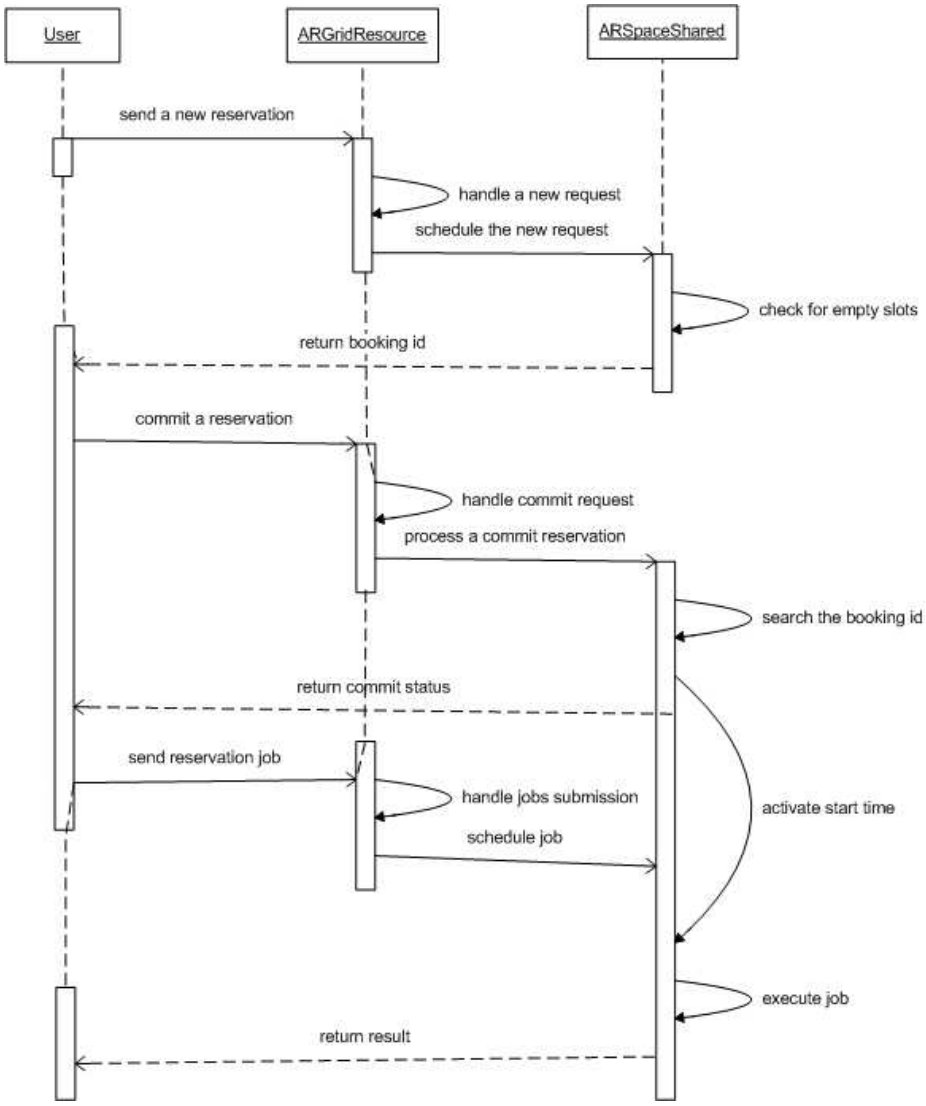

Figure 8. Sequence diagram

overestimate completion time so a job can finish executing without worry running out of time and being pre-empted. For a restartable job, underestimate completion time is allowed as the scheduler will put it into a queue if there is not enough time.

4.2 Mean Waiting Time

Figure 9 displays the impact on MWT of queued jobs when reservations are enabled in all resources. For all the jobs in five resources, queue wait times increase an average of 20% when 10% of the jobs are reservations and 71% when 20% of the jobs are reservations. In [3], MWT for 10% and 20% jobs are 13% and 62% respectively. Both experiments show a similar percentage increase of MWT by $\pm$50%.

4.3 Mean Offset Time

Figure 10 displays MOT of 0%, 10% and 20% jobs using reservations for all five resources. The figure shows that the offset is larger when there are more reservations. 0% reservations mean that they are immediate reservations with current time as the start time. These immediate reservations also have duration and number of CPUs requested. When comparing between immediate and advanced reservations, the result benefits significantly to jobs that requests resources ahead of time. For 10% reservations, the mean

Table 1. Testbed resources simulated using GridSim.

Resource Name in simulation	Simulated resource characteristics: vendor, type, OS	host name and location	A SPEC Rating (m)	Num CPU (P)	Time zone (GMT)
R0	Compaq, AlphaServer, Tru64 UNIX	grendel.vpac.org VPAC, Melbourne, Australia	515	4	$+10$
R1	Intel, Pentium4 2GHz, Linux	manjra.cs.mu.oz.au Melbourne Univ., Australia	684	13	$+10$
R2	SGI, Origin 3200, IRIX	onyx3.zib.de ZIB, Berlin, Germany	410	16	$+2^*$
R3	SGI, Origin 3200, IRIX	mat.ruk.cuni.cz Charles Univ., Prague, Czech Republic	410	6	$+2^*$
R4	Sun, Ultra, Solaris	pitcairn.mcs.anl.gov ANL, Chichago, USA	377	8	-5^*

* denotes daylight saving time ($+1$ hour) occurred in this area at the time of writing this paper.

Algorithm 1 Finding an empty slot

1: **if** $List$ is empty **then**
2: no conflict found. Hence, accepts a new reservation.
3: **else**
4: **for** $i = 0$ to $List\ size - 1$ **do**
5: put i into $tempList$ if one of the following properties are true:
6: $-start_{new} \leq start_i$ && $finish_{new} \geq finish_i$
7: $-start_{new} \leq finish_i$
8: $-finish_{new} \leq finish_i$
9: **end for**
10: **if** $tempList$ is empty **then**
11: no conflict found. Hence, accepts this reservation.
12: **else**
13: calculate CPU_{reserv} if $start \longrightarrow end$ time interval for a reservation overlaps with others
14: **if** $CPU_{reserv} + CPU_{new} \leq CPU_{resource}$ **then**
15: empty CPUs found. Hence, accepts this reservation.
16: **else**
17: no empty CPUs found. Hence, rejects this reservation
18: **end if**
19: **end if**
20: **end if**

Figure 9. Mean Waiting Time

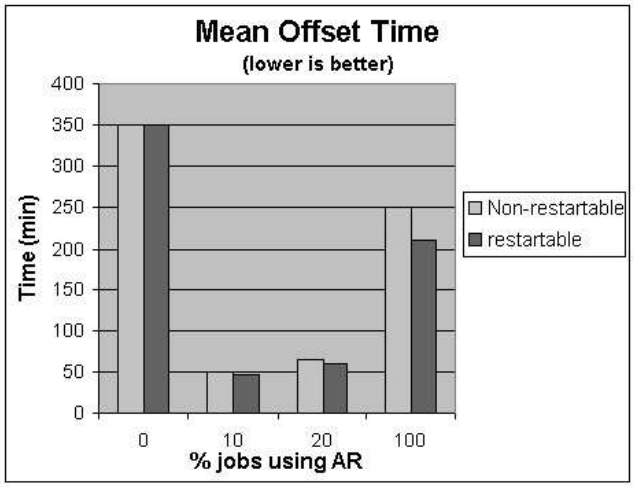

Figure 10. Mean Offset Time

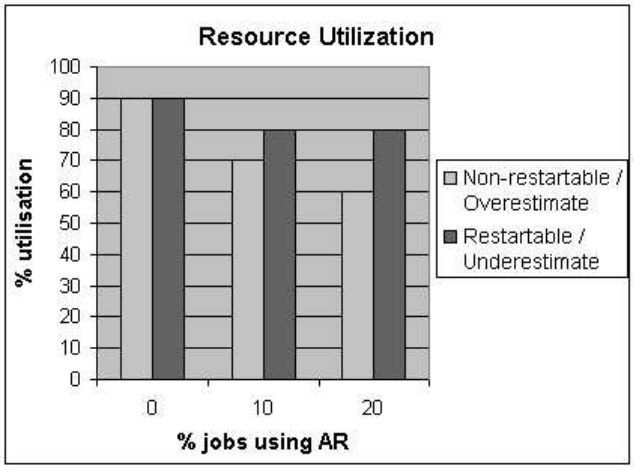

Figure 11. Number of utilization

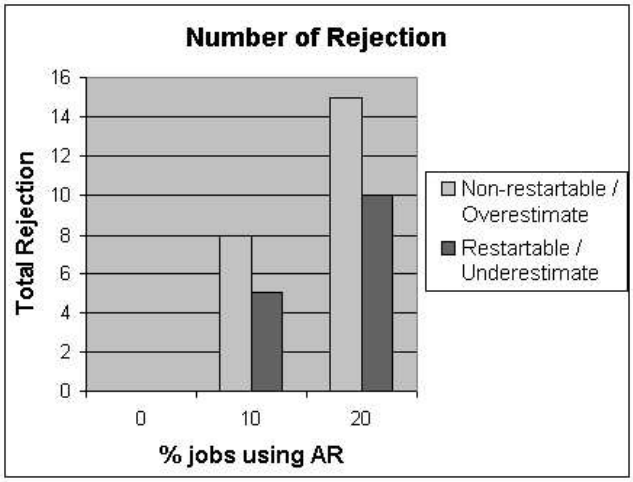

Figure 12. Number of rejections

difference from requested reservation time is 48 minutes. For 20% reservations, the mean difference is 62 minutes. This is an increase of 29% over the mean difference from requested reservation time when 10% of the jobs are reservations. In [3], MOT increase from 10% to 20% reservations is 32%.

Experiments conducted in [3] use workload traces taken from supercomputers in Argonne National Laboratory (ANL) [17], the Cornell Theory Center (CTC) [18], the San Diego Supercomputer Center (SDSC) [19]. Both MWT and MOT results between this experiment and in [3] are quite similar. This proves the validity and realistically of this experiment.

4.4 Resource Utilization

Figure 11 displays the utilization of resources being simulated in this experiment. When reservations are supported, resource utilization drops to between 60% and 80%. The result finds that utilization does not change for restartable jobs. This is because when they are running longer, they will be put into a queue if a slot is taken and will be processed later. In this experiment, restartable jobs are giving an underestimate completion time. In constrast, non-restartable jobs are giving an overestimate completion time to prevent from being pre-empted by other reservations. As a result, resource utilization is lower as most of non-restartable jobs are finishing earlier.

4.5 Number of Rejections

Figure 12 displays the number of rejections for requesting new reservations in this experiment. It is expected that as more reservations are requested, the number of rejections will be higher. Non-restartable jobs have higher rejection rate since they request longer duration time.

4.6 Parallel Jobs

The results seemed to disadvantage a system using AR. However, for parallel applications, as depicted in Figure 13, AR is very beneficial. For this experiment, all the jobs are parallel and they required 4 CPUs for executing simultaneously. Sending parallel jobs without reserving beforehand have the lowest percentage of successful completion. 10% jobs using AR perform better than 20% jobs as they have less competition to access the same resource.

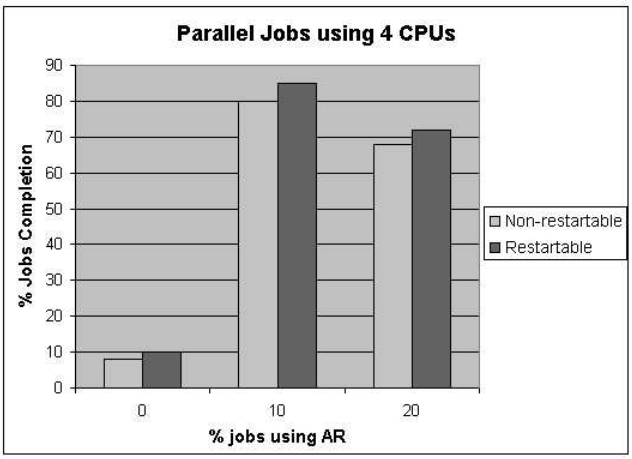

Figure 13. Parallel Jobs using AR

5 Conclusion and Future Work

Advance reservation of grid resources allow users to gain *concurrent access* to adequate resources for applications to be executed in parallel. Advance reservation also guarantees the availability of resources to users and applications at the specified times in the future.

This paper introduced advance reservation mechanisms incorporated into GridSim, a grid simulation tool. The design and implementation for supporting advance reservation in GridSim are also discussed. Experimental results have shown that a reservation-based system benefits advanced reservations from immediate ones and parallel jobs significantly, as the resources are guaranteed to be available at the specified time.

Future work on GridSim will look into various ways to prioritize reservations, and to evaluate the relative importance of different applications, in order to reject less important applications for others. In addition, policies and penalties for cancellation and modification of reservations need to be introduced.

References

[1] Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, USA, 1999.

[2] Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience, Vol. 14, Issue 13–15. Wiley Press, 2002.

[3] Smith, W., Foster, I., and Taylor, V.: Scheduling with Advanced Reservations. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) Conference, May 2000.

[4] Buyya, R., Branson, K., Giddy, J., and Abramson, D.: The Virtual Laboratory: Enabling Molecular Modeling for Drug Design on the World Wide Grid. The Journal of Concurrency and Computation: Practice and Experience (CCPE), 15(1): 1–25, Wiley Press, Jannuary 2003.

[5] Buyya, R., Date, S., Mizuno-Matsumoto, Y., Venugopal, S. and Abramson, D.: Neuroscience Instrumentation and Distributed Analysis of Brain Activity Data: A Case for eScience on Global Grids. Journal of Concurrency and Computation: Practice and Experience, Wiley Press, USA (accepted in Jan. 2004 and in print).

[6] Aida, K., Takefusa, A., Nakada, H., Matsuoka, S., Sekiguchi, S., Nagashima, U.: Performance Evaluation Model for Scheduling in a Global Computing System. The International Journal of High Performance Computing Applications, Vol. 14, No. 3. Sage Publications, USA, 2000.

[7] Song, X., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., Chien, A.: The MicroGrid: A Scientific Tool for Modelling Computational Grids. Proceedings of IEEE Supercomputing (SC 2000). Dallas, USA, Nov 4–10, 2000.

[8] Casanova, H.: Simgrid: A Toolkit for the Simulation of Application Scheduling. Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001). IEEE Computer Society Press, Brisbane, Australia, May 15–18, 2001.

[9] Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K. and Zini, F.: OptorSim – A Grid Simulator for Studying Dynamic Data Replication Strategies. International Journal of High Performance Computing Applications, 17(4), 2003.

[10] Hoo, G., Jackson, K. and Johnston, W.: Design of the STARS Network QoS Reservation System, Journal of Communications and Networking (JCN) - special issue on QoS in IP networks, June 2000.

[11] Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K., and Roy, A.: A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation, International Workshop on Quality of Service, 1999.

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

[13] MacLaren, J. (ed.): Advance Reservations: State of the Art (draft). GWD-I, Global Grid Forum (GGF), June 2003
http://www.ggf.org

[14] Roy, A. and Sander, V.: Advance Reservation API, GFD-E.5, Scheduling Working Group, Global Grid Forum (GGF), May 2002

[15] Standard Performance Evaluation Corporation. http://www.spec.org

[16] SPEC. SPEC CINT2000 Results.
http://www.specbench.org/cpu2000/

[17] Argonne National Laboratory.
http://www.anl.gov

[18] Cornell Theory Center.
http://www.tc.cornell.edu

[19] San Diego Supercomputer Center.
http://www.sdsc.edu