A GRID SIMULATION INFRASTRUCTURE SUPPORTING ADVANCE RESERVATION

Anthony Sulistio and Rajkumar Buyya
GRIDS Laboratory and NICTA Victoria Laboratory,
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
ICT Building, 111 Barry Street, Carlton, VIC 3053
email: {anthony, raj}@cs.mu.oz.au

ABSTRACT
Advance Reservation (AR) for global grids becomes an important research area as it allows users to gain concurrent access for their applications to be executed in parallel, and guarantees the availability of resources at specified future times. Evaluating various AR scenarios can not feasibly be carried out on a real grid environment due to its dynamic nature. Therefore, we extend a GridSim simulation package to support AR for repeatable and controlled evaluations. This paper discusses the design and implementation of AR within GridSim, together with the effects of AR from users' and resources' point of view in the experiments.

KEY WORDS
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 [2]. Common resources whose 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.

There are some systems that support AR capability, such as Globus Architecture for Reservation and Allocation (GARA) [3] and Maui Scheduler [4]. However, to validate the effectiveness of these scheduling systems, 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, and creating an adequately-sized testbed is expensive and time consuming. Therefore, it is easier to use simulation as a means of studying complex scenarios.

Some tools are available for application scheduling simulation in the Grid computing environment, such as Bricks [5], SimGrid [6] and OptorSim [7]. However, none of them have the capability of simulating reservation-based systems. To address this weakness, we extend GridSim to support AR mechanisms.

GridSim [8] is a Java-based grid simulation package that provides features for application composition, information services for resource discovery, and interfaces for assigning applications. GridSim also has the ability to model heterogeneous computational resources of variable performance.

In this work, GridSim has been extended 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 describes the general life-cycle of a reservation, while Section 3 presents the architecture of GridSim for AR. Section 4 describes how a user can use GridSim's functionalities. Section 5 conducts an experiment to show the effects of AR from users' and resources' point of view. Section 6 concludes the paper and suggests further work.

2 States of Advance Reservation

A reservation can be in one of several states during its lifetime as shown in Figure 1. The life-cycle of a reservation in GridSim is influenced by recommendations from the Global Grid Forum (GGF) draft [9] and the Application Programming Interface (API) [10]. 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 re-

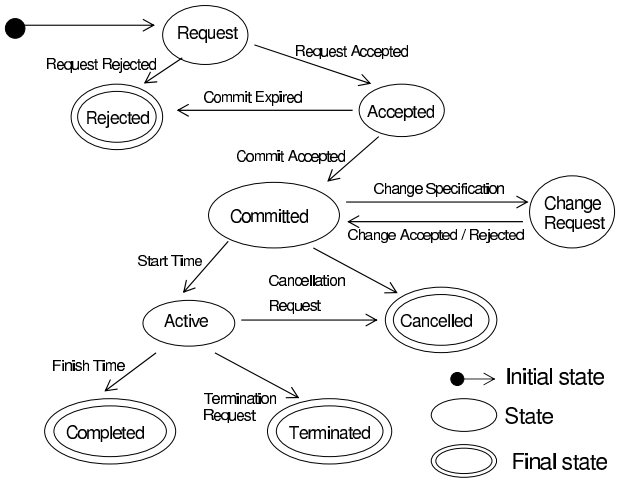

Figure 1. A state transition diagram for advance reservation

quest for a reservation is first made.

- **Rejected:** The reservation is not successfully allocated due to full 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 a resource.
- **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 resource now executes the reservation.
- **Cancelled:** A user no longer requires a reservation and requests that it is to be cancelled.
- **Completed:** The reservation's end time has been reached.
- **Terminated:** A user terminates an active reservation before the end time.

The following sections describe the implementation and usage of these states into GridSim.

3 GridSim Resource Design

In GridSim, a grid resource is represented by a `GridResource` object. Each `GridResource` object contains only one scheduler of type `AllocPolicy` class. In this case, the `GridResource` only acts as an interface

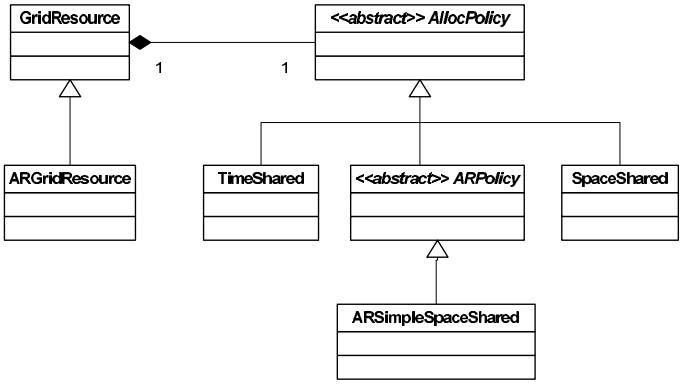

Figure 2. A GridSim resource class diagram

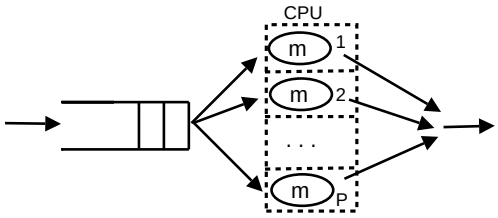

Figure 3. A queueing network model for the GridSim scheduling system

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. Currently, GridSim has `TimeShared` and `SpaceShared` objects that use Round Robin and First Come First Serve (FCFS) approaches respectively.

To support AR in GridSim, a new type of resource entity named `ARGridResource` inherited from `GridResource` is added as shown in Figure 2. Similarly, a new abstract scheduler class called `ARPolicy` inherited from `AllocPolicy` is also added. The extension to the current grid resource architecture is needed to incorporate the states of AR as discussed earlier.

The 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` or `ARPolicy` class, and implementing the required abstract methods. For an example, `ARSimpleSpaceShared` as shown in Figure 2, is a child of `ARPolicy` class, that uses FCFS approach to schedule reserved jobs.

3.1 System Model

GridSim mainly focuses on simulating computational grids. In this context, `GridResource` entities in GridSim

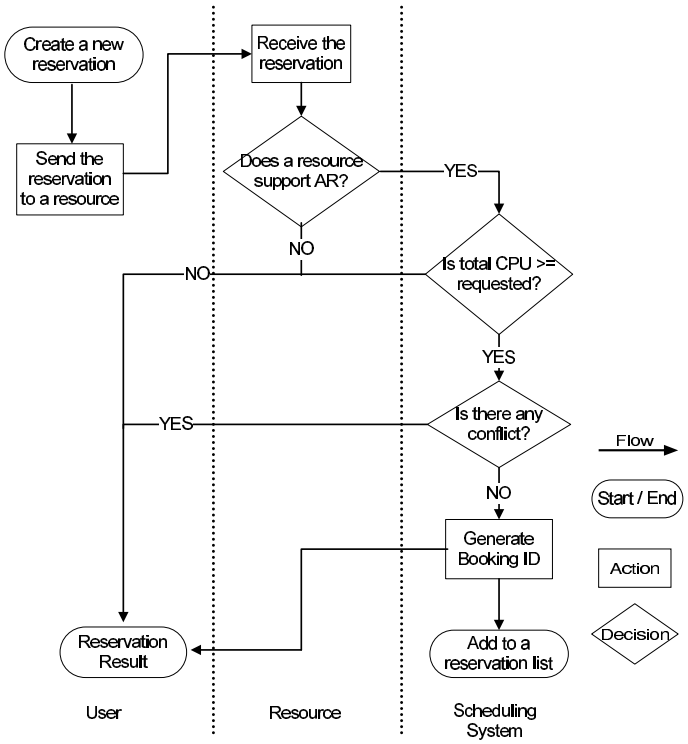

Figure 4. Flowchart for requesting a new reservation

are mainly related to processing capability and the cost of processing. Currently, a scheduler within a grid resource focuses on reserving CPUs. An open queueing network model of a resource is considered as seen in Figure 3. There is a finite buffer with size S to store objects waiting to be processed by one of P independent CPUs. They are connected by a high-speed network with negligible communication delays. m is the CPU speed, measured in the form of Million Instructions Per Second (MIPS) rating as per SPEC (Standard Performance Evaluation Corporation) [11] benchmarks. CPUs can be homogeneous or heterogeneous. For homogeneous ones, all m have the same MIPS rating. Different m exist for heterogeneous CPUs.

3.2 Scheduling System for Advance Reservation

Currently, GridSim uses FCFS approach for the `ARSimpleSpaceShared` object for a reservation-based system. For requesting a new reservation, Figure 4 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 of 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.

Algorithm 1 explains 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 straight away. 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 the $start_{new} \longrightarrow finish_{new}$ interval. By storing indexes of all reservations that lay within this time interval, CPU_{reserv} can be calculated as in line (13).

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**

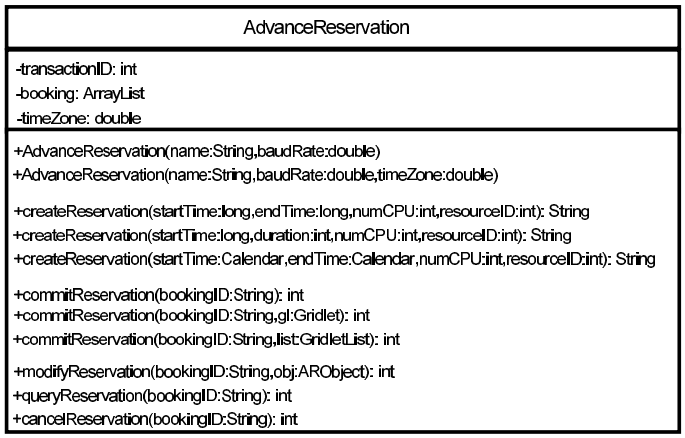

Figure 5. AdvanceReservation class diagram

4 GridSim API

The GridSim client-side API for AR is encoded in the method calls of the `AdvanceReservation` class as shown in Figure 5. In this class diagram, attributes and methods are prefixed with characters $+$ and $-$ indicating access modifiers public and private respectively. Due to space constraints, only few methods are drawn and discussed in this paper. Detailed API of this class can be found on the GridSim website [12].

The `transactionID` attribute is a 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 resources are located geographically in different time zone. Hence, a 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 the `createReservation()` method. Before running a GridSim program, an initialization of some parameters is required. One of the parameters 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. The start time can be of type `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 a unique booking id as a `String` object. Otherwise, it will return an approximate busy time in the interval of 5, 10, 15, 30 and 45 in time units. The time unit can be in seconds or minutes or hours. If a request is rejected, the user can negotiate with the resource by modifying the requirements, such as reducing the 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 5, can be done in one of the following ways:

- committing first before the expiry time by sending a booking id. Then, once a job is ready, 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.

The `queryReservation()` method aims to find out the current status of the overall reservation. Cancellation and modification of the reservation can be done by invoking `cancelReservation()` and `modifyReservation()` respectively.

The overall sequence from a new reservation until the completion of jobs is captured in Figure 6.

5 Simulation Results and Discussions

5.1 Experiment Setups

The experiment models and simulates four resources with different characteristics, configurations and capabilities. The four selected resources mentioned in [8] 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 [13].

This experiment simulates a scenario demonstrating GridSim's ability to handle AR functionalities. In addition, this experiment aims to find out the effects of AR from users' and resources' point of view. The following simulation setups are carried out:

- Five created resources, each able to handle AR functionalities.

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.

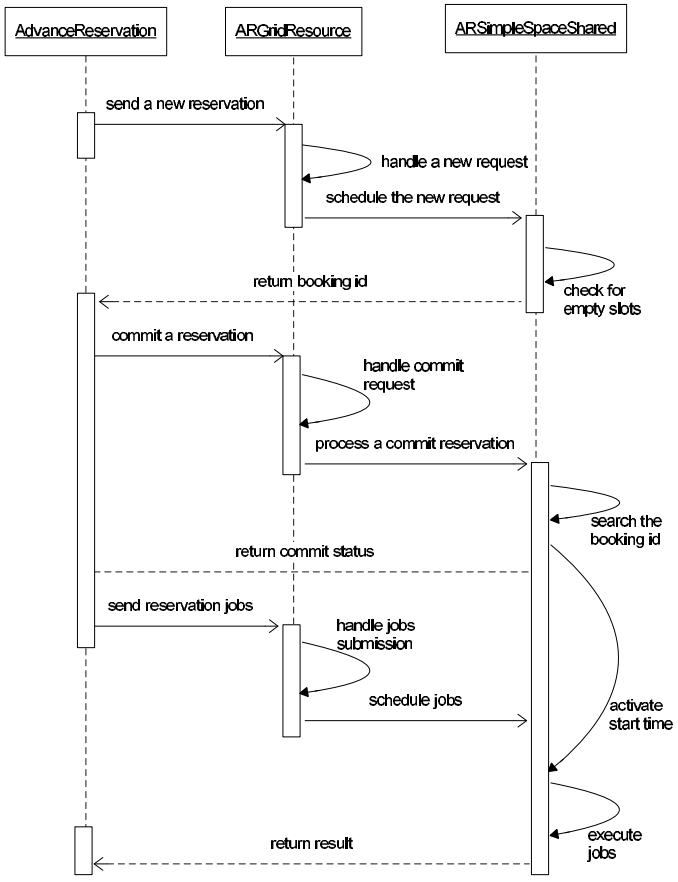

Figure 6. A sequence diagram for performing a new reservation in GridSim

- 50 created users, each with 100 Jobs.
- Job size is uniformly distributed in [500,000 ... 8,000,000] in Millions Instructions (MI) unit.
- Poisson distribution is used for job arrival time. Average jobs for all users per time unit is 5.
- Assuming 1 simulation time is equivalent to 1 minute, maximum arrival time for this experiment is 500 minutes.
- Sending jobs are uniformly distributed among the five resources as mentioned in Table 1.
- Only concerns scheduling jobs to empty CPUs. All resources and users are assumed to have same network bandwidth and I/O operations.
- A scheduler from each resource will start the job at the requested start time.

In addition, a reservation job can be `restartable` or `non-restartable`. A restartable job means a job will be paused if running longer than allocated, then resume execution from the point where it was paused if there is an empty CPU. A non-restartable job means a job will be executed from start until finish or be pre-empted by a scheduler, and straight away the finished or partially completed job sent back to a user. For this experiment, only 0%, 10% and 20% of total jobs using reservations and it follows similar approach done by [2]. However, experiments conducted in [2] use workload traces taken from supercomputers in Argonne National Laboratory (ANL) [14], the Cornell Theory Center (CTC) [15], and the San Diego Supercomputer Center (SDSC) [16].

5.2 Advance Reservation Analysis

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

Figure 7. Mean Waiting Time

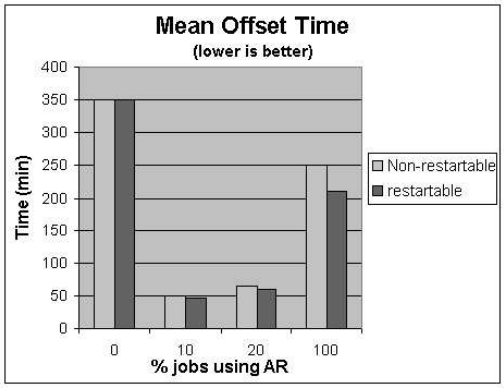

Figure 8. Mean Offset Time

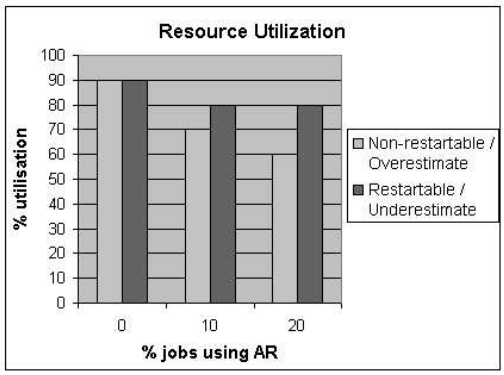

Figure 9. Number of utilization

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.

Achieving accurate prediction of a job's execution time is a difficult problem for AR. Therefore, the effects of the job's estimation on resource utilization and on the number of rejections are presented. For a non-restartable job, a reservation must have an overestimated completion time so a job can finish executing without the risk of running out of time and being pre-empted. For a restartable job, underestimated completion time is allowed as the scheduler will put it into a queue if completion time takes longer than allocated.

5.2.1 Mean Waiting Time

Figure 7 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.

5.2.2 Mean Offset Time

Figure 8 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 request resources ahead of time. For 10% reservations, the mean 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.

5.2.3 Resource Utilization

Figure 9 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 given an underestimated completion time. In constrast, non-restartable jobs are given an overestimated completion time to prevent from being pre-empted by other reservations. As a result, resource utilization is lower as most non-restartable jobs are finished earlier.

5.2.4 Number of Rejections

Figure 10 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 a higher rejection rate since they request a longer duration time.

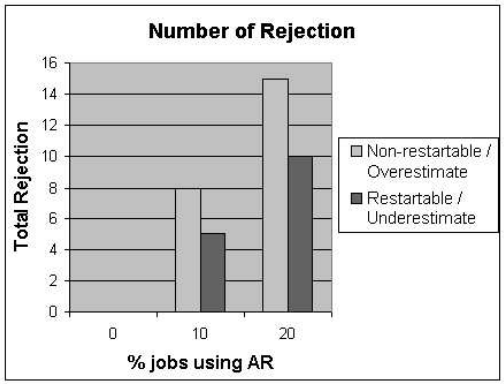

Figure 10. Number of rejections

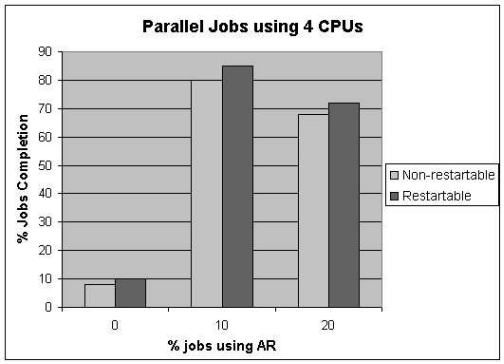

Figure 11. Parallel Jobs using AR

5.2.5 Parallel Jobs

The results seemed to disadvantage a system using AR. However, for parallel applications, as depicted in Figure 11, AR is very beneficial. For this experiment, all the jobs are parallel and required 4 CPUs for being executed simultaneously. Sending parallel jobs without reserving beforehand has the lowest percentage of successful completion. 10% of jobs using AR perform better than 20% of jobs as they have less competition to access the same resource.

6 Conclusion and Future Work

Advance reservation for grid resources allows users to gain *concurrent access* for their applications to be executed in parallel. It also guarantees the availability of resources 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 parallel and reserved jobs significantly, as the resources are guaranteed to be available at the specified time. Also, the mean offset from requested reservation time is lower for reserved jobs. However, the effects of advance reservation resulted in lower CPU utilization and a higher number of rejections in accepting new jobs.

Future work on GridSim will look into introducing policies and penalties for canceling and modifying reservations. In addition, a priority system for reserving resources needs to be introduced.

References

[1] I. Foster and C. Kesselman (Ed.), *The Grid: Blueprint for a Future Computing Infrastructure*, (San Mateo: Morgan Kaufmann Publishers, 1999).

[2] W. Smith, I. Foster, and V. Taylor, Scheduling with Advanced Reservations, *Proc. of the Int. Parallel and Distributed Processing Symposium (IPDPS) Conf.*, Cancun, Mexico, 2000, 127–132.

[3] 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, *Int. Workshop on Quality of Service*, London, U.K., 1999, 27–36.

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

[5] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima, Performance Evaluation Model for Scheduling in a Global Computing System, *Int. J. of High Performance Computing Applications*, 14(3), 2000, 268–279.

[6] H. Casanova, Simgrid: A Toolkit for the Simulation of Application Scheduling. *Proc. of the First IEEE/ACM Int. Symposium on Cluster Computing and the Grid*, Brisbane, Australia, 2001, 430–437.

[7] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini, OptorSim – A Grid Simulator for Studying Dynamic Data Replication Strategies, *Int. J. of High Performance Computing Applications*, 17(4), 2003, 403–416.

[8] R. Buyya and M. Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, *J. of Concurrency and Computation: Practice and Experience*, 14(13–15), 2002, 1175–1220.

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

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

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

[12] GridSim website. `http://www.gridbus.org/gridsim`

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

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

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

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