Chapter IV

Proposed Advance Reservation Model
CHAPTER 4

PROPOSED ADVANCE RESERVATION MODEL

4.1 Introduction

The chapter\(^2\) presents a grid framework for e-governance applications. Advance reservation is the concept used to reserve the resources in advance. The grid architecture is designed and it provides such a wide service to both the resource owner and the user by using the advance reservation algorithm. The chapter also presents a grid simulator called GridSim to support advance reservation of compute nodes.

4.2 Proposed Design of Advance Reservation

Advance reservation for global grids becomes an important concept as it allows users to gain concurrent access for their applications. It executes parallel and guarantees the availability of resources at specified future times.

4.2.1 Grid framework for an e-governance applications

The figure 4.1 shows the distributed grid framework for e-governance application [96]. This architecture is generic enough to accommodate different models used for e-governance applications.

\(^2\)This chapter is an extended version of the article Efficient Advance Reservation Algorithm for the Exchange of Data in E-Governance Applications. Journal of Convergence Information Technology, ISSN: 1975-9320. - Communicated.
The key components of the grid include,

- Grid user with applications
- Grid resource broker
- Grid middleware services and tools
- Grid service providers

Figure 4.1 Proposed grid framework for an e-governance applications

The vital players in data grids are resource providers and resource consumers. Both are inevitable parts of the grid. In the grid resource, consumers adopt the strategy of solving their problems within a required timeframe. Resource providers adopt the strategy of obtaining best possible usage of grids. The resource owners try to maximize their resource utilization by adopting the algorithm.
Unlike other grid systems, it does not need to offer tools and mechanisms that allow both resource providers and consumers to express their requirements. They are capable to deal between these requirements and steer the computations accordingly. This architecture provides such a wide service to both the resource owner and the user by using the algorithm.

The resource providers can contribute their resources to the grid and charge for services. They can use the payment server to define the charge for the resources used by the user. The users interact with the grid by defining their requirements through high level tools like resource brokers. The resource brokers work for the consumers and attempt to maximize user utility.

The meta scheduler has three components viz., the process server, intermediate queue and the application queue. The process server in the meta scheduler consists of the decision-making algorithm. This is used to process the entire request made to it for the applications. The intermediate queue uses a buffer which stores the jobs or the applications. The intermediate queue is used especially when they are switched between the servers. The application queue is the initiator of the meta scheduler. This is used vitally to sort all the stored jobs or applications using the sorting algorithm. This plays the vital role in deciding which job is to be executed first based on priority.

As mentioned earlier, the goal is to enhance the grid architecture by leveraging existing technologies using the advance reservation algorithm. Therefore it mainly focuses on two things such as to develop resource utilization
and data grids. The data grids use these services in developing advanced user-centric grid resource brokers. The highlighting feature of the model is that it allows advanced reservation which enables user to reserve resources in advance.

4.2.2 Algorithm for Advance Reservation

Advance reservation task in grid environments are difficult because resource availability often changes during workflow execution. The main idea of the algorithm is to take advantage of the environmental changes without the need to probe the remote sites.

A Task Graph (TG) is a model of a parallel program that consists of many subtasks that can be executed simultaneously on different compute nodes or processing elements. Subtasks exchange data through an interconnection network. The dependencies between subtasks are described by means of a Directed Acyclic Graph (DAG). Executing a task graph is determined by two factors:

(i) A node weight that denotes the computation time of each subtask

(ii) An edge weight that corresponds to the communication time between dependent subtasks [124].

Thus, to run these task graphs, a target system that is tightly coupled by fast interconnection networks is needed. Typically, systems like cluster computing provide an appropriate infrastructure for running parallel programs. The figure 4.2 illustrates that each task graph can be represented in a Standard Task Graph (STG)
format [124]. The first row of the STG format consists of two integer values, representing the total subtasks and the target processing elements.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td># total subtasks and target PEs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0 # subtask index, node weight, and num of parents</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2 # parent index and edge weight</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2 # subtask index, node weight, and num of parents</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>5 # parent index and edge weight</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2 # parent index and edge weight</td>
</tr>
</tbody>
</table>

Figure 4.2  Standard Task Graph (STG) Format

The target processing element is the number of processing elements required or requested by a user to execute one task graph. In this figure, a task graph consists of 9 subtasks (T0 to T8) that require 3 target processing elements. Then, a specification of individual subtask is described in a new row. As shown in figure 4.2 each row consists of three integers, denoting the subtask index or id, its node weight and number of parents.

If the subtask has a dependency, the row contains two numbers specifying its parent id and the edge weight. In addition, the STG format is similar to the one
proposed by Kasahara et al. [116]. Finally, # denotes a single line comment in the STG format. Figure 4.3 shows the structure of task graph. In this figure, a subtask's edge weight is represented by a number next to the arrow line.

Figure 4.3 Structure of a task graph

Scheduling the task graph in a non-dedicated environment is a challenging process because of the following constraints:

- The task graph requires a fixed number of processors for execution. Hence, a user needs to reserve the exact number of compute nodes.

- Due to communication overhead between the subtasks on different processing elements, each subtask must be completed within a specific time of period.

- Each subtask needs to wait for its parent subtask to finish executing in order to satisfy the required dependencies.
The figure 4.3 depicts the same. Therefore the advance reservation is needed to secure or guarantee resources prior to the execution of the subtasks.

Figure 4.4 shows the schedule of a task graph. It is derived from the STG format and structure of a task graph. In order to minimize the schedule length and the communication costs of a task graph, its subtask must be assigned to appropriate processing elements and it must be started after their parent subtasks.

![Figure 4.4 Schedule of a task graph](image)

As shown in the figure 4.3, T6 depends on T4 and T5. Thus, T6 must wait for both subtasks to finish, and it will be scheduled on the same processing element as T5. This is because executing T6 on PE1 and PE2 incurs a communication time of 7 (addition of 5 with 2) and 5 time units respectively. In contrast, running T6 on PE0 after T5 have a penalty of 2 time units as shown in figure 4.3. In this context the efficiency is measured by the ratio of the total node weight in relation to the overall processing time provided for the task graphs. Figure 4.4 shows the efficiency of this TG schedule as 9/18 or 50%. If there are no idle processing elements at all time then the efficiency can be said to be optimal.
The efficiency of a task graph can be increased by rearranging and moving subtasks, interweaving with other task graphs, and backfilling with other independent jobs.

To interweave all the subtasks in different task graphs, it is needed to first rearrange the subtasks of task graph, then move the subtasks and then interweave the subtask. As interweaving is a fragile process all the subtasks in the task graphs must be aligned accordingly.

Rearrangement of subtasks is based on the total number of subtasks. The figure 4.5 describes the execution of each processing elements.

**Algorithm for Rearranging subtasks of task graph**

**Input:** The task graph and the number of compute nodes

```plaintext
foreach number of compute nodes
    Calculate the number of subtasks for the entire index in a task graph.
    Initialize the index with the value for number of compute nodes to be reserved.
    At the end update the index with the number of subtask
    Update the schedule with the task graph and index.
end
return.
```

**Figure 4.5 Algorithm: Rearranging subtasks**

In this algorithm the index can be denoted as an indexing array. Thus it is necessary to store the total number of subtasks running on each processing elements. Then, it must be sorted from the lowest to the highest number of
subtasks. Finally the update schedule function is used to update the schedules of task graph, since each subtask may now be executing on a different processing elements.

The figure 4.6 shows the movement of one or more subtasks from one processing element to another. In this algorithm it is important to find a list of subtasks that run on particular processing elements and if there are two or more subtasks that depend on each other must be tagged or grouped as a whole.

```
Algorithm for moving subtasks of task graph to different processing elements

Input the task graph and the number of compute nodes

foreach number of compute nodes
    Collect the subtask of the entire task graph and update it to the subtask list.
    Group the subtask on dependencies and edge weight.
    Add subtasks into the list of the entire processing elements
end ;

foreach number of compute nodes to be reserved
    Assign another variable to compare between processing elements.
    If the value of new variable is greater than or equal to the number of compute nodes to be reserved, break the loop.
    Merge the respective subtasks
end

Update the schedule with the task graph and index.

return.
```

Figure 4.6 Algorithm: Moving subtasks
The tagging or grouping is needed to prevent them from executing into different processing elements, which may incur hefty communication costs. Finally, a loop is needed to merge the two processing element-id arrays into one, provided that there are empty slots that fit one or more subtasks.

Interweaving can be done by combining two or more task graphs from the list and by keeping the original allocation and dependencies untouched. For each reserved PE on both task graphs, it is necessary to find a list of subtasks and its starting and ending time. Then it must be checked whether both task graphs are suitable for each other.

The matching criteria should posses different starting time, ending time, or a combination of one or more reserved processing elements from the same task graph. If there is no significant increase in the average efficiency of Schedule Processing Elements (SPEs), the given task graphs cannot be interlocked properly.

The figure 4.7 is an algorithm to determine the scheduling order of the two task graphs, where it also depends on the matching criteria. Then the reserved processing elements of each task graph will be sorted accordingly. This algorithm shows how to interweave two task graphs with the use of an algorithm of rearranging subtasks.

Finally both task graphs should be interweaved as one. Interweaving can be done by delaying or modifying the starting time of subtasks. Though this process may create fragmentations or time gaps of idle processor-cycles on the right, it can be hopefully closed.
Algorithm for interweaving two task graphs

**Input** TG1, TG2 and the number of compute nodes

**Foreach** number of compute nodes

- Collect the subtask of respective task graphs to be Interweaved.
- Add the subtask for every processing element respectively
- Assign the start time of the respective task graphs
- Calculate the end time of the respective task graphs

**end:**

Check if the given two task graphs

Match each other, if not exit

Determine the scheduling order of the two TG's if TG1 is to proceed TG1 else TG2 to proceed TG1.

Sort the task graphs that run the TG

Index the last PE.

**foreach** number of compute nodes

- Calculate the gap time with the schedules last PE's start time and the current PE's start time
- Calculate the current schedules start time with the finish time of the schedule and gap time

Update the schedule.

Calculate the new PE id with the schedule of the appended TG and halt.

Calculate the new TG with the PE id and TG1 and TG2.

**end**

Return the new TG

---

**Figure 4.7 Algorithm: Interweaving subtasks**
If there are smaller independent jobs, then fit it in and execute without delaying any of the subtasks of task graph. In contrast to the interweaving step the best fitting jobs should only be selected.

It starts with the first gap and looks for a job that has an estimated schedule length lower or (best) equal to the gap's length. As an example there is enough gap on PE0 in figure 4.8 (on the right) to put two small independent jobs (each runs for 1 time unit) or one bigger job that needs to be scheduled for 2 time units.

**Figure 4.8 Combining the execution of two TGs by interweaving**

All the three algorithms together give an effective reduction in the time consumed by advance reservation. By interweaving a set of task graphs the advance reservation scheduler manages to reduce the overall reservation duration time.

### 4.3 Implementation of Advance Reservation

Advance reservation algorithm is implemented through a simulator. The system architecture of the gridsim is explained with their different states.
4.3.1 Prototype Simulation

Different scenarios need to be evaluated to ensure the effectiveness of advance reservation and scheduling techniques. As given in 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. Moreover, the testbed requires the handling of different administration policies at each resource. Due to these reasons, the thesis proposes modeling and simulation of different scenarios without a full-scale implementation of grids.

For simulating a grid, a tool needs to be modeled for the interaction of users, resource brokers (on behalf of the users), resources and the network. For these purposes, a grid simulation tool must have at least the following functionalities:

1. Able to model heterogeneous resources, for computational and/or data grids.
2. Extensible and modifiable, so that various scheduling systems can be implemented and analyzed.
3. Able to store and query information about resource properties and/or data files. This is achieved by using an indexing or catalog service.
4. Able to specify an arbitrary network topology in the simulated grid environment.

Based on the above requirements, GridSim [101] is chosen as the preferred simulation tool. GridSim is an open-source software platform, which provides
features for application composition, information services for resource discovery, and interfaces for assigning applications to resources. GridSim also has the ability to model heterogeneous resources of various configurations.

4.3.2 Grid Simulator

Simulation has been used extensively for modeling and evaluating the real world systems in business process and factory assembly line. Consequently, modeling and simulation has emerged as an important discipline in many standard application-specific tools and technologies. It include simulation languages (e.g. Simscript [115]), simulation environment (e.g. Parsec [97]), simulation libraries (e.g. SimJava2 [126]), and application specific simulators (e.g. NS-2 network simulator [90]). While there exists a large body of knowledge and tools, there are very few well-maintained tools available for application scheduling in grid computing environments.

OptorSim [129] is being developed as part of the European (EU) DataGrid project. It aims to mimic the structure of an EU DataGrid Project and to study the effectiveness of several grid replication strategies. It is quite a complete package as it incorporates few auction protocols and economic models for replica optimization. However, it mainly focuses more on the issue of data replication and optimization.

The SimGrid toolkit [47], developed at the University of California at San Diego (UCSD), is a C language based toolkit for the simulation of application
scheduling. It supports modeling of resources that are time-shared. The load can be injected as constants or real traces. It is a powerful system that allows creation of tasks in terms of their execution time and resources, with respect to a standard machine capability.

The MicroGrid emulator [44] is modeled after Globus [48] is a software toolkit used for building grid systems. It allows execution of applications constructed using the Globus toolkit in a controlled virtual grid resource environment. MicroGrid is actually an emulator. It executes application code on the virtual grid. Thus, the results produced by MicroGrid are much closer to the real world and it is a real implementation. However, using MicroGrid requires knowledge of Globus and implementation of a real system/application to study.

GangSim [15], developed at the University of Chicago, is targeted towards a study of scheduling policies in a multi-site and multi-virtual organization environment. It combines discrete simulation techniques and modeling of real grid components in order to achieve scalability to grids of substantial size.

Finally, GridSim [40] supports simulation of various types of grids and application models scheduling.

4.3.2.1 Key Features

GridSim is an open-source software platform, written in Java, which provides features for application composition, information services for resource discovery, and interfaces for assigning applications to resources. The GridSim
toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domain(s) distributed computing systems such as clusters and grids. Application schedulers in grid environment are called resource brokers. It performs resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user. Each user has their own private resource broker and hence, it can be targeted to optimize for the requirements and objectives of its owner.

Schedulers and managing resources such as clusters in a single administrative domain have complete control over the policy. It is used for the allocation of resources. Here all users need to submit their jobs to the central scheduler. It can be targeted to perform global optimization such as higher system utilization, overall user satisfaction depending on resource allocation policy or optimize for high priority users.

In order to prove the effectiveness of resource brokers and associated scheduling algorithms, their performance needs to be evaluated under different scenarios such as varying the number of resources and users with different requirements. In a real grid environment, it is hard and perhaps even impossible, to perform scheduler performance evaluation in a repeatable and controllable manner for different scenarios. The availability of resources and their load continuously varies with time. It is impossible for an individual user/domain to control activities
of other users in different administrative domains. To overcome this limitation, the experiment is simulated using grid simulation toolkit called GridSim.

This toolkit supports modeling and simulation of heterogeneous grid resources (both time- and space-shared), users, brokers, and application models. It provides primitives for the creation of application tasks, the mapping of tasks to resources. The simulator implements various scheduling algorithms. The performance of the reservation algorithms can be evaluated by varying the number of users, deadlines, and optimization strategies.

Salient features of the GridSim toolkit include the following:

It allows modeling of heterogeneous types of resources.

- Resources can be modeled operating under space- or time-shared mode.
- Resource capability can be defined (in the form of MIPS as per SPEC benchmark).
- Resources can be located at any time zone.
- Weekends and holidays can be mapped depending on resource’s local time.
- Resources can be booked in advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.

- Network speed between resources can be specified.

- It supports simulation of both static and dynamic schedulers.

- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

### 4.3.2.2 System Architecture of Gridsim

The figure 4.9 shows a multi-layer architecture and abstraction for the GridSim platform and its applications.

- The first layer is concerned with the scalable Java's interface and the runtime machinery, called JVM (Java Virtual Machine), whose implementation is available for single and multiprocessor systems including clusters [9].

- The second layer is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer. One of the popular discrete-event infrastructure implementations available in Java is SimJava [98].

- The third layer is concerned with modeling and simulation of core Grid entities such as resources, information services, and so on; application
model, uniform access interface, primitive application modeling and framework for creating higher level entities. The GridSim toolkit focuses on this layer that simulates system entities using the discrete-event services offered by the lower-level infrastructure.

Figure 4.9 Architecture for GridSim platform and components

- The fourth layer is concerned with the simulation of resource aggregators called Grid resource brokers or schedulers.
- The final layer focuses on application and resource modeling with different scenarios using the services provided by the two lower-level layers for evaluating scheduling and resource management policies and algorithms.
4.3.3 States of Advance Reservation

This section discusses the functionalities of advance reservation in GridSim. The figure 4.10 shows that a reservation can be one of several states during its lifetime. The life-cycle of a reservation in GridSim is influenced by recommendations from the Global Grid Forum (GGF) draft [60] and the Application Programming Interface (API) [7].

Transitions between the states are defined by the operations that a user performs on the reservation. These states are defined as follows:

1. **Requested**: Initial state of the reservation, when a request for a reservation is first made.
2. **Rejected**: The reservation is not successfully allocated due to full slots, or an existing reservation has expired.
3. **Accepted**: A request for a new reservation has been approved.
4. **Committed**: A reservation has been confirmed by a user before the expiry time, and will be honored by a resource.
5. **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.
6. **Active**: The reservation's start time has been reached. The resource now executes the reservation.
7. **Cancelled**: A user no longer requires a reservation and requests it to be cancelled.

8. **Completed**: The reservation's end time has been reached.

9. **Terminated**: A user terminates an active reservation before the end time.

![State Transition Diagram for Advance Reservation](image)

**Figure 4.10** A state transition diagram for advance reservation
From the above states, GridSim uses a two-phase commit, where a user requests for a new reservation first. If the request is accepted then the user needs to commit the reservation within a specified time limit.

If the request gets rejected, then the user needs to negotiate until it succeeds. 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 grid resource acts only as an interface between users and the local scheduler. The scheduler has to process the submitted jobs. This approach gives the flexibility to implement various scheduling algorithms for a specific resource-based system.

4.4 Summary

An algorithm for advance reservation to reserve resources for jobs in grid environments is described in this chapter. The advance reservation scheduler manages to interweave a set of task graphs. Often, the evaluation of complex scenarios cannot feasibly be carried out on a real grid environment due to its dynamic nature. In addition, grid testbeds are limited and creating an adequately-sized testbed is expensive and time consuming. Moreover the testbed requires the handling of different administration policies at each resource. Therefore it is easier to use simulation as a mean of studying these complex scenarios.