Chapter VI

Proposed Hybrid Architecture
CHAPTER 6

PROPOSED HYBRID ARCHITECTURE

6.1 Introduction

This chapter presents the hybrid architecture with the advance reservation and the scheduling strategies. The new algorithm called hybrid-core for the hybrid architecture for e-governance applications is described in grid environment. This includes the advance reservation and the scheduling algorithms.

6.2 Design of Hybrid Architecture

The hybrid architecture is designed to implement the e-governance model in the grid environment. In this architecture, the hybrid-core algorithm is proposed, which includes the advance reservation and scheduling.

6.2.1 Proposed Hybrid Architecture

The figure 6.1 shows the design of the proposed hybrid architecture. This presents the implementation of a grid resource broker that mediates access to distribute computational and data resources running in diversified middleware. The broker is able to discover remote data repositories, interface with various middleware services and select suitable resources in order to meet the

---

1This chapter is an extended version of the article The Hybrid Architecture for the Secure Exchange of Data in E-Governance Applications, Published in the CiiT International Journal of Networking and Communication Engineering. ISSN: 0974-9713. DOI: NCE052009005.
application requirements. The grid environment consists of EB server, CWSS server and the corporation server. The meta scheduler consists of process server, intermediate queue and the application queue.

![Diagram of Grid Environment and Internal Process Flow]

**Figure 6.1** The hybrid architecture and the internal process flow in grid environment

The process server in the meta scheduler consists of the decision-making algorithm, which is used to process the entire request made 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 scheduler is used vitally to sort all the stored jobs or applications using the sorting algorithm.
This play a vital role in deciding which job is to be executed first based on priority.

The main functions of the Grid Service Provider (GSP) are to

- manage the grid applications
- submit the jobs to multiple grid infrastructures
- enable the grid resources and applications in multiple portals
- save the grid job configurations and history

GSP is a set of high-level grid services. The GSP package includes the features of Job Submission Service, Application Management Service, Short News Service and Provider Management Service. After finishing the job the payment can be verified by the payment server.

The details of the figure 6.1 are given below.

1. The user requests through the user interface.

2. The user interface in turn passes the request to the Application Queue which makes use of the advance reservation and the scheduling algorithm.

3. The application queue passes the request according to the scheduling algorithm to the corporation server. Once the corporation server is accomplished with the request, it passes it to the intermediate queue.

4. The intermediate queue which queues the request accordingly and sends it to the CWSS server. Then the CWSS server sends it to the process server.

5. The process server then processes the request and sends it to the corresponding EB Server.
6. The payment details are sent to the payment server from the meta scheduler after the completion of the EB server's process.

7. Then the payment server sends the corresponding details to the user interface. So these are the internal process flow in the grid environment designed for the hybrid architecture.

### 6.2.2 Hybrid-Core Algorithm

The figure 6.2 gives the hybrid-core algorithm. This algorithm improves the performance simply by dividing the entire process into two parts. One for the resources to be guaranteed prior to their executions with various PE’s and the other is scheduling the processes for the job.

The available resources are updated and sorted as a group after completion of every job. For each and every application from the user the hybrid-core algorithm calls the advance reservation and the scheduling process to complete the entire process with the help of three servers.

In this hybrid-core algorithm the user submits jobs through the user Interface. The user Interface in turn passes the request to the Application Queue which makes use of the advance reservation and the scheduling algorithm. The Application queue passes the request according to the scheduling algorithm to the corporation server.
Input: User submitting jobs

1. Create the Application Queue (AQ)
2. Sort the job then determine Intermediate Queue (IQ)
3. Calculate the absolute queue strength
4. Repeat the following steps for each job to be dispatched:
   5. (i) Call Advance Reservation
   6. (ii) Call Scheduling
   7. Select a job from the list
   8. Request sends to Corporation server, and then Return
   9. Request sends to CWSS server, and then Return
   10. Request sends to EB server, and then Return
   11. Update the payment server
   12. Until the job is empty
   13. end.

Figure 6.2 Hybrid-Core Algorithm

Once the corporation server gets the request, it passes the request to the intermediate queue. Sorted requests are sent to the CWSS server. Then the process is verified and the CWSS server sends it to the process server. The process server then processes the request and sends it to the corresponding EB Server. The EB server processes the payment details and sends it to the payment server. Finally the payment server sends the corresponding details to the user interface.

The hybrid-core algorithm calls the advance reservation and scheduling algorithms in line 5 and line 6 respectively. The advance reservation algorithm is given in figure 6.3 and the explanations are given in the chapter 4. This guarantees
the resources prior to their executions. The scheduling algorithm is described in figure 6.4 and the explanations are described in the previous chapter.

**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 6.3 Algorithm: Advance Reservation*
If any of the resource has assigned jobs, non-dispatched jobs will be moved to Unassigned-Jobs-List. This helps in updating the whole schedule. Based on the latest resource availability information, it repeat the steps (5) to (11) in figure 6.2 for the jobs in unassigned-jobs-list until all the jobs are completed.

### Algorithm for Scheduling

```plaintext
while J ≠ Ø OR T_{current} < T_{Deadline} do
  // If jobs created is not null, or the expected time is lesser than the deadline
  foreach r ∈ R do
    // For every resources
    Calculate performance data on the basis of resource performance in the previous polling interval
  end
  foreach d ∈ D do
    // For every data host
    Update the network information
    Let R_d ← \{r_m | r_m < r_{m+1} \text{ if } 1/BW(\text{Link}(d, r_m)) < 1/BW(\text{Link}(d, r_m)) \}
    ∀ r_m ∈ R, 1 ≤ m ≤ M
  end
  // Mapping Begins
  foreach j ∈ J do
    // For every jobs created for application
    Let S_j ← \{R_j, D_j\}, R_j ← φ, D_j ← φ
    Let R_{j_{temp}} ← φ //A temporary variable
    foreach f ∈ F_j do
      // For every dataset required by j
      Let U ← \{(d_f, r)| d_f ∈ D_f \} where r is the first element of ordered set R_{d_f}
      Find (d_f, r) such that T_t(f, d_f, r) + T_e(j, r) is minimum over U
      if S_j = \{φ, φ\} then
        R_j ← \{r\}, D_j ← \{d_f\}, R_{j_{temp}} ← \{r\}
      end
      else
        R_j ← \{r\} D_j ← \{d_f\}
      end
    end
    S_j ← min{\{R_j, D_j\}, \{R_{j_{temp}}, D_j\}\}, R_{j_{temp}} ← R_j
  end
  // Mapping Ends
  Dispatch(J, T_{Deadline}) // Dispatch with time deadline
  Wait until next polling interval
  Update resource by taking into account jobs completed in the last interval
end
```

Figure 6.4 Algorithm: Scheduling
Any job given to the resources for a water connection process will be queued in a list. Once the job arrives, the size and the waiting time of the jobs are estimated. So that a lengthy job will not wait for a long time. Jobs are sorted and assigned to the resources according to the assigned priorities. If any of the e-governance jobs exceeds the deadline, the job is sent back to the queue, assigning the actual remaining time and the waiting time is assigned to zero.

This hybrid-core algorithm implemented for e-governance applications is optimal. Because the resources are allocated prior to the job arrival. Accordingly it calculates and updates the available resources. So, once a job arrives the only process left is to assign the same to an apt resource. This drastically reduces the time taken for the execution of a job.

6.3 **Simulation of grid**

Extraction of the code in building a simple experiment with GridSim is shown in this section. In this experiment users are trying to reserve compute nodes by applying the schedule process.

Before creating any GridSim entities and running the experiment, it is needed to initialize the SimJava2 simulation kernel. The initialization must be done through the GridSim.init() method, as shown in figure 6.5. The method requires three parameters such as the total number of users, the current starting time of the experiment and a flag denoting whether to record communication.
events among GridSim entities to a log or trace file. The trace file can be used for debugging purposes.

```java
// Code snippet for initializing GridSim
public static void main ( String [] args )
{
  try {
    int num_user = 5 ; // number of users created
    Calendar cal = Calendar.getInstance(); // experiment starting time
    boolean trace_flag = false ; // trace GridSim events or not
    GridSim.init ( num_user , cal , trace_flag ) ;
    . . . // other code for instantiating new grid resources and users
  }
  catch (Exception e)
  {
    . . . // other code for handling errors
  }
}
```

**Figure 6.5 Initializing GridSim**

The next step of building an experiment with GridSim is to create one or more grid resources by using the `createGridResource()` method, as shown in figure 6.6. First a list of machines can be created, where each machine has more than one PE or CPU. In GridSim, the total processing capability of a resource's CPU rating is modeled in the form of Million Instructions per Second.

In this example, a cluster with homogeneous machines and these systems stores static properties of a resource, such as operating system, system architecture and time zone are created. These properties may influence the user’s decision in submitting their jobs. Next the object `SimpleLink` and `ResourceCalendar` are created for linking this resource to a network and storing information about its
machines availability at various times. Finally a scheduler can be created that supports advance reservation. In this case the ARSimpleSpaceShared object is created.

```java
private static ARGridResource createGridResource ( String name , int totalPE , int totalMachine , int rating )
{
    MachineList mList = new MachineList ( ) ;
    for ( int i = 0 ; i < totalMachine ; i++ )
    {
        PEList peList = new PEList ( ) ;
        for ( int k = 0 ; k < totalPE ; k++ )
        {
            peList.add ( new PE(k , rating ) ) ;
        }
        mList.add ( new Machine ( i , peList ) ) ;
    }
    ResourceCharacteristics resConfig = new ResourceCharacteristics ( . . . ) ;
    Link link = new SimpleLink ( . . . ) ;
    ResourceCalendar cal = new ResourceCalendar ( . . . ) ;
    ARGridResource gridRes ;
    try
    {
        ARSimpleSpaceShared policy = new ARSimpleSpaceShared ( . . . ) ;
        gridRes = new ARGridResource ( name , link , resConfig , cal , policy ) ;
    }
    catch ( Exception e )
    {
        . . . // other code for handling errors
    }
    return gridRes ;
}
```

**Figure 6.6 Creating grid resources**

This experiment can be executed by calling the GridSim.startGridSimulation() method as shown in figure 6.7. Once the simulation
starts, the newly created entities (e.g. resources, users and routers) run in parallel in their own thread according to the runtime behavior.

```java
// Code snippet for building and running GridSim.

public static void main ( String [ ] args )
{
try {
    int num_user = 5 ;
    Calendar cal = Calendar.getInstance( ) ;
    boolean trace_flag = false ;
    GridSim . init ( num_user , cal , trace_flag ) ;
    int total_resource = 3 ;
    ArrayList resList = new ArrayList ( total_resource ) ;
    for ( int k = 0 ; k < total_resource ; k++)
    {
        ARGridResource res = new create GridResource ( . . . ) ;
        resList.add ( res ) ;
    }
    ArrayList userList = new ArrayList ( num_user ) ;
    for ( int i = 0 ; i < num_user ; i++)
    {
        UserEntity user = new UserEntity ( . . . ) ;
        userList.add ( user ) ;
    }
    connectEntity ( resList , userList , trace_flag ) ;
    GridSim.startGridSimulation ( ) ;
}
catch ( Exception e )
{
    . . . // other code for handling errors
}
}

Figure 6.7 Running GridSim
```
6.4 Implementation of Hybrid Architecture

The open source Globus Toolkit 4.0 (GT4) [142] is fundamental enabling technologies for the “Grid”, letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

![Components of Globus Toolkit](image)

**Figure 6.8 Components of Globus Toolkit**
6.4.1 Globus Toolkit 4.0

The figure 6.8 represents the components of GT4. The GT4 is composed of several software components. As shown in the figure, these components are divided into five categories: Security, Data Management, Execution Management, Information Services, and the Common Runtime.

- **Security**: Using the Security components, based on the Grid Security Infrastructure (GSI), the communications are secure.

- **Data Management**: This component will allow managing large sets of data in our virtual organization.

- **Execution Management**: This components deal with the initiation, monitoring, management, scheduling and coordination of executable programs, usually called *jobs*, in a grid.

- **Information Services**: The Information Services, more commonly referred to as the Monitoring and Discovery Services (MDS), includes a set of components to discover and monitor resources in a virtual organization. Note that GT4 also includes a non-WS version of MDS (MDS2) for legacy purposes.

The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. The figure 6.9 shows the extraction of the code to create interfaces with GT4.
In addition to being a central part of science and engineering projects that total nearly half-a-billion dollars internationally, The Globus Toolkit is a substrate on which leading IT companies are building significant commercial grid products.

Figure 6.9 Creating Interfaces in GT4

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and
portability. It is packaged as a set of components that can be used either independently or together to develop applications.

6.4.2 Experiment setup

The hardware details of the resources used for the simulation are given in this section. Details of the resources including configuration are also provided.

6.4.2.1 Hardware Details

The experiment models six resources with different characteristics, configurations and capabilities.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Resource Characteristics (Vendor, type and OS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Intel, P4 2.4 GHz, Linux RLC5</td>
</tr>
<tr>
<td>R2</td>
<td>Sun, Ultra, Solaris</td>
</tr>
<tr>
<td>R3</td>
<td>Intel Core 2 quad, P4 2.4 GHz, Windows XP SP3</td>
</tr>
<tr>
<td>R4</td>
<td>IBM, IBM RACK Server, Unix</td>
</tr>
<tr>
<td>R5</td>
<td>Intel, P4 2.4 GHz, Linux RLC5</td>
</tr>
<tr>
<td>R6</td>
<td>Intel, P4 2.4 GHz, Windows XP SP3</td>
</tr>
</tbody>
</table>

Table 6.1 Testbed resources simulated using GridSim

The six selected resources included in this experiment are mentioned in the table 6.1. The table gives all the relevant information about the resources such as R3 and R4 are the faster resources. The resources R1, R2 and R5 are medium
speed processors. The resource R6 will be used for only processing the jobs not for storing any data. For example the details of the resource R3 is given in detail in table 6.2. This experiment simulates a scenario demonstrating GridSim’s ability to handle advance reservation functionalities. In addition, this experiment aims to find out the effects of advance reservation from the users and the resources point of view.

<table>
<thead>
<tr>
<th>Configuration (HCL Infiniti PC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
</tr>
<tr>
<td>Main Board</td>
</tr>
<tr>
<td>RAM (Random Access Memory)</td>
</tr>
<tr>
<td>HDD (Hard Disk Drive)</td>
</tr>
<tr>
<td>ODD (Optical Disk Drive)</td>
</tr>
<tr>
<td>LAN Port (On board)</td>
</tr>
<tr>
<td>USB Port (On board)</td>
</tr>
<tr>
<td>PCI Slot (On board)</td>
</tr>
<tr>
<td>PCI Express Slot (On board)</td>
</tr>
<tr>
<td>I/O Device</td>
</tr>
<tr>
<td>Monitor</td>
</tr>
</tbody>
</table>

Table 6.2  Configuration of the resource used for evaluation of R3
The following simulation setups are carried out:

- Five created resources, each able to handle advance reservation and scheduling functionalities.
- Twenty five users are created, each with fifty jobs.
- Job size is uniformly distributed in Millions Instructions (MI) per unit time.
- An average job for all users per time unit is 5.
- Assuming one simulation time is equivalent to one minute, maximum arrival time for this experiment is 25 minutes.
- Allocations of jobs are uniformly distributed among the five resources.
- Scheduled jobs are sent 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.

6.5 Experimentation via simulation

The screen shots taken during the processes execution in GridSim toolkit are given in this section. The figure 6.10 denotes the creation of grid resources using GridSim. In this figure it shows that the creation of the resource details with the resource id.
The figure 6.11 gives the completion of the simulation processes. The details of the processes with the resource id are given. The details of the requests and the acknowledgement between the resources are also given.
NewPolicy.gridletSubmit(): it works ........receiving Gridlet #1
NewPolicy.gridletSubmit(): sends back Gridlet #1 to User #11

NewPolicy.gridletSubmit(): it works ........receiving Gridlet #2
NewPolicy.gridletSubmit(): sends back an acknowledgement.
NewPolicy.gridletSubmit(): sends back Gridlet #2 to User #11
Ack = true

User_0: Sending Gridlet_3 to GridResource_0 with id = 8 at time = 26.479999999999997
NewPolicy.gridletSubmit(): it works ........receiving Gridlet #3
NewPolicy.gridletSubmit(): sends back Gridlet #3 to User #11
<<<<<< pauses for 20 seconds >>>>>>>
User_0: Receiving Gridlet 0
User_0: Receiving Gridlet 1
User_0: Receiving Gridlet 2
User_0: Receiving Gridlet 3
User_0: Exiting body()
Sim_system: No more future events
Gathering simulation data.
Simulation completed.

OUTPUT for User_0 ===========
<table>
<thead>
<tr>
<th>Gridlet ID</th>
<th>STATUS</th>
<th>Resource ID</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
<td>8</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>Success</td>
<td>8</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>Success</td>
<td>8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Time below denotes the simulation time.
Time (sec) Description Gridlet #0
0.00 Creates Gridlet ID #0

Figure 6.11 Process execution in simulation

The figure 6.12 shows the details of the processes. The details of the grid resources with their status such as whether the request is processing or waiting for an event. The input and the output user information are also given.
Figure 6.12 Details of the processes

The screen shots of the experimentation in the globus toolkit are given below. Figure 6.13 and figure 6.14 represents the details of the processes running in globus toolkit. The details of the processes and the services are displayed in the screen.
Figure 6.13 Process execution in GT4

Figure 6.14 Process execution in GT4
6.7 Summary

The hybrid architecture is given for reserving resources in advance for jobs in grid environment with scheduling strategies to sort and speedup the processes when required in e-governance applications. The new algorithm called hybrid-core for the grid architecture for e-governance applications is given with the experimentation. The thesis recommends a perfect blend of scheduling algorithms with the advance reservation to produce better results based on cost and time. These studies were carried out through the simulation on GridSim, a discrete-event grid simulation tool since different scenarios need to be evaluated and repeated. The output is generated by using Globus toolkit 4.0. The screen shots taken during the experiment is also given.