Chapter 6

Prototype Implementation of QoS Grid
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

6. Prototype Implementation of QoS Grid

6.1 Experimental Setup

The grid test-bed created in the laboratory consists of multiple linux machines running Ubuntu linux 9.10 and windows system connected with network. The multiple nodes with different configuration have been chosen to represent the heterogeneity in the grid. The Globus Toolkit 4.2.1 is used as grid middleware in the prototype implementation.

Operating System: Ubuntu linux 9.10, Windows XP
Grid Middleware: Globus 4.2.1, Java COG Kit 4
Local scheduler: Open PBS, Condor, fork (linux operating system)
Agent Development Environment: JADE 3.7 (Java Agent DEvelopment Framework)

Grid Nodes used in the Prototype Implementation:

grid1.aes.edu (Job submission Master node with QoS Oriented Scheduler, also used as processing node)
grid2.aes.edu (processing node and backup node for Scheduler)
grid3.aes.edu (processing node with fork)
grid4.aes.edu (processing node with PBS)
grid5.aes.edu (processing node with Condor)

“The Globus Toolkit is open source software for building computational grids by Globus Alliance. It is open source grid middleware and fundamental set of technologies for the implementing grid. It allows sharing of all the compute resources in an organization securely across the organizational and administrative boundary [49]”. The toolkit provides the protocols, services, software libraries with API for development of grid applications, effectively utilizing the distributed grid resources and infrastructure using grid services like resource discovery, single sign-on, remote job submission and file staging [57].
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

The grid implementation demonstrates the interaction of QoS oriented resource broker that mediates access to distributed computational resources and provides resource discovery, job scheduling, execution, monitoring and gathering of output.

User Authentication using GSI (Grid Security Infrastructure)

Installing Grid Certification Authority (CA), Acquiring Globus container, host and user certificates from CA, Adding authorization, the certificate subject name of a user and account name it should map to (/etc/grid-security/grid-mapfile)

Configuration of GRAM (Grid Resource Allocation and Management)

GRAM protocol enables grid users to locate, submit, monitor and cancel remote jobs on Grid-based compute resources.

Grid Monitoring and Discovery System (MDS)

It includes suite of web services to monitor and discover resources and services on Grids. GT 4.2.1 includes WS MDS, the WSRF implementation of MDS.

Ganglia Information Provider

It is a scalable distributed monitoring system for high-performance computing systems based on a hierarchical design. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. It gathers cluster data from resources running Ganglia using the XML mapping of the GLUE schema and reports it to a GRAM4 service. Ganglia Information Provider is used to query CPU processor clock speed, current CPU load, main memory status, file system data and other attributes of host machines required for QoS scheduling in the grid [167].
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

Web based MDS
Web MDS enables grid end users to view monitoring information via a standard web browser interface, without installing any additional software on their PC [168]. It is implemented as a servlet program deployed on Apache Tomcat application server.

Grid Middleware Interface

The resource broker interfaces to Globus toolkit based grid service functionality using the Java Commodity Grids (CoG) Kit [169] that provides APIs for accessing Globus services through the Java framework. The Java CoG Kit provides convenient access to Grid middleware through the Java framework. It contains abstract APIs for accessing standard Grid services, such as job submission and file transfer in a grid middleware independent way. This CoG kit provides high-level Application Programming Interface (API) to the services provided by the Globus toolkit [170].

JADE (Java Agent Development Environment)

“JADE is an open source framework for development of mobile agents and peer to peer applications. It simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA (Foundation for Intelligent Physical Agents) specifications [171]”. The QoS oriented scheduler and related functionality is implemented as agents which are deployed on multiple heterogeneous nodes in the grid (which not even need to share the same OS and Hardware platform).

Agent-oriented approach to grid system engineering involves multiple coordinating agents:
Grid Job Agent

It extracts all job details and its QoS requirements (using GUI), interfaces with scheduler and monitoring performance. Each agent represents a client user, carries his/her job requests, interacts with resource broker’s agent and retrieves back the results.

QoS Scheduling Agent

It is responsible for providing QoS oriented resource management and scheduling. To enhance adaptability and fault tolerance, a simple replication schema is applied to scheduling agents; each resource possesses a single primary manager (for consistency sake) but knows also of secondary managers or co-managers. Primary manager creates clone of itself, so that whenever primary manager fails, secondary manager’s takes over its responsibility.
6.2 Use Case Model of Grid Prototype Implementation

6.2.1 Use case model for Grid Authentication

Figure 6.1: Use Case Model for Grid Authentication
6.2.2 Use Case Model for Grid Scheduling

Figure 6.2: Use Case Model for Grid Scheduling
6.3 Structural Model of Grid Prototype Implementation

Figure 6.3: Structural Model of Prototype Grid Implementation
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

Sequence Diagram: Grid Authentication

![Sequence Diagram](image)

Figure 6.4: Sequence diagram for grid authentication process
Activity Diagram for QoS Oriented Scheduling

Figure 6.5: Activity Diagram for QoS Oriented Scheduling
6.4 Grid Application Scenarios

6.4.1 Grid Parameter Sweep Application (PSA)

Initial grid scheduling experiment was based on simple application of Grid enabled factorial calculation which is a parameter sweep application. It involves large range of numbers and high precision calculation - distributed among grid nodes using QoS scheduling algorithm.

![Parameter Sweep Application (Grid enabled Factorial)
Speed up in execution time Job parameters 1 to 1000](image)

Figure 6.6: A parameter sweep factorial job having parameters 1 to 1000
6.4.2 High Performance Application Scenario of Histogram Calculation

For experiments and evaluation of parallel application on Grid resource broker, we have considered the parallel application scenario of histogram calculation having large range of numeric array elements of a parallel application to generate the histogram.

Histogram is a statistical application, which requires processing large number of tabulated frequencies into discrete intervals (bins) and then graphically representing the distribution of data. In such type of histogram called as frequency histogram, the tabulated frequencies are represented as filled rectangles having the height equivalent to the number of tabulated frequencies in the interval [185], E.g. Tabulating the CGPA of 1000 students of University on a ten point interval scale.

The histogram application takes as input an array with maximum number of elements. Enter the number of processes to generate histogram for that array. Input number of histogram bins (index). Application will count the frequency of elements in a range to generate histogram. The output should generate a histogram of the block along with number of elements within the range of block. This problem can be solved very efficiently by parallelizing the histogram application on Grid. I have applied the proposed Grid resource brokering algorithm for scheduling distributed histogram application and present experiment results. Depending upon maximum and minimum value of array element, main process computes the bin size as

\[ \text{Binsize} = (\text{max} – \text{min}) / \text{no of bins} \]

The problem is parallelized using a heterogeneous network of machines in grid. The master process (rank 0) accepts the size of array elements, number of histogram blocks and total number of processes. The same process fills the data (dynamic array) with the following function

\[ \text{Fill} (\text{Data}, \text{Size}, \text{Lo}, \text{Hi}) \]
Typically the histogram computation is parallelized by partitioning the size of array elements into various segments and then assigning each segment to different processor.

**Main process:** Divide the array elements into number of processes and distribute it on other processors including itself to compute bin and generate partial histogram. The main process receives the partial histogram computed by various processor and integrates the result.

**Other processes:** Read array element and compute bin using following equation

\[
\text{bin} = 1 + \left(\text{int}\left( \frac{\text{abs}(a[i] - \text{min})}{\text{binsize}} \right) \right)
\]

If bin > number of bins then set bin to number of bins. Once bin is computed update the histogram for that bin as

\[
\text{Histogram}[\text{bin}] += 1
\]

**Experiment Results and Evaluation**

The grid resource broker algorithm is evaluated by executing the above parallel application on our grid testbed. We have performed the scheduling experiment by executing the parallel application on different grouping of nodes in the grid and its effect on the execution efficiency.

Table-6.1: Combination of Grid nodes in different groups

<table>
<thead>
<tr>
<th>Group</th>
<th>Nodes in Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Node 1, Node 2</td>
</tr>
<tr>
<td>G2</td>
<td>Node 1, Node 2, Node 3</td>
</tr>
<tr>
<td>G3</td>
<td>Node 1, Node 2, Node 3, Node 4</td>
</tr>
<tr>
<td>G4</td>
<td>Node 1, Node 2, Node 3, Node 4, Node 5</td>
</tr>
</tbody>
</table>
The above result indicates that that for a given number of array elements when number of used processing nodes in Grid increases, the time initially decreases for given number of processes. But as number of processes increases, the computational time taken by nodes also increases. The improvement in execution time is depicted in the figure.

Figure 6.7: Improvement in Execution time of Histogram Application

Figure 6.8: User interface of Histogram Calculation on Grid
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

Figure 6.9: Job Result of Histogram Calculation on Grid

Figure 6.10: QoS Oriented Grid Resource Information using MDS
Figure 6.11: Screen for specifying Job QoS Requirements

Table 6.2: Histogram Execution with increase in no. of data items

<table>
<thead>
<tr>
<th>Group Size (No. of bins) = 5</th>
<th>No. of Processes = 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Data Items</td>
<td>Execution Time (in ms)</td>
</tr>
<tr>
<td></td>
<td>1 node</td>
</tr>
<tr>
<td>1000</td>
<td>0.875</td>
</tr>
<tr>
<td>10000</td>
<td>1.254</td>
</tr>
<tr>
<td>100000</td>
<td>10.912</td>
</tr>
<tr>
<td>1000000</td>
<td>32.187</td>
</tr>
<tr>
<td>10000000</td>
<td>310.26</td>
</tr>
<tr>
<td>100000000</td>
<td>3092.922</td>
</tr>
<tr>
<td>1000000000</td>
<td>27732.169</td>
</tr>
<tr>
<td>100000000000</td>
<td>59493.756</td>
</tr>
</tbody>
</table>
Table 6.3: Histogram Execution with increase in no. of grid nodes used

<table>
<thead>
<tr>
<th>Group Size (No. of bins) = 5</th>
<th>No. of Processes = 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Nodes</td>
<td>Execution Time (in ms)</td>
</tr>
<tr>
<td>No. of Data Items = 1000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>1</td>
<td>90.5</td>
</tr>
<tr>
<td>2</td>
<td>52.4</td>
</tr>
<tr>
<td>3</td>
<td>34.85</td>
</tr>
<tr>
<td>4</td>
<td>20.35</td>
</tr>
<tr>
<td>5</td>
<td>13.45</td>
</tr>
</tbody>
</table>
Figure 6.13: Histogram Execution with increase in no. of grid nodes used
6.4.3 Monte Carlo Simulation of PI using Parallel Processing on Grid

Monte Carlo (MC) methods are stochastic techniques—meaning they are based on the use of random numbers and probability statistics to investigate problems [172] Monte Carlo methods use statistical simulation using sequences of random numbers generation to perform the simulation. When performing this experiment, I realized that it takes a very large number of throws to get a decent value of pi, well over 1,000. So, we distribute the number of trials the user wants to attempt among all the available nodes in the grid. It should be apparent that of the total number of darts that hit within the square, the number of darts that hit the shaded part (circle quadrant) is proportional to the area of that part. In other words [115],

\[
\frac{\# \text{darts hitting shaded area}}{\# \text{darts hitting inside square}} = \frac{1}{4} \pi r^2 = \frac{1}{4} \pi \quad (6.1)
\]

\[
\Pi = 4 \frac{\# \text{darts hitting shaded area}}{\# \text{darts hitting inside square}} \quad (6.2)
\]

The simulation problem involves large amount of processing which can be parallelized on multiple nodes in the grid using Message passing interface (MPI). In this grid enabled Monte Carlo simulation of PI, the large number of throws using random numbers required in the simulation experiment are distributed to the different processors of the grid machines based on brokering algorithm. On each node, the intermediate results of calculating above equations are found out and send to the coordinator process. The results of experiment trials from processors of different machines in the grid are then aggregated to calculate the simulated value of PI.
The algorithm for Monte Carlo simulation of PI is as under:

Generate random numbers a and b representing the random generated point P(a, b).
Increment the Total No. of Throws by 1
Calculate the distance of the point P from the origin of the Circle Q (0,0).

Distance from origin of circle \( D = \sqrt{a^2 + b^2} \)
If \( D \leq 1.0 \)
   No. of Successful throws = Earlier No. of Successful throws + 1
Approximate value of \( \pi = 4 \times (\text{No. of Successful throws} / \text{Total No. of Throws}) \)

Figure 6.14: Algorithm for Monte Carlo simulation of PI

Table 6.4: Input Parameters for Monte Carlo Simulation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed</td>
<td>It’s a random number generator</td>
</tr>
<tr>
<td>Iteration</td>
<td>No. of trials/attempt made</td>
</tr>
<tr>
<td>Assigned Processors</td>
<td>No. of processors which we are used for Simulation</td>
</tr>
</tbody>
</table>

The following factors are important output parameters for the evaluation of Monte Carlo Simulation and its grid resource brokering on compute grid.

Table 6.5: Output parameters for Monte Carlo Simulation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI Estimated</td>
<td>PI value which is calculated by program</td>
</tr>
<tr>
<td>Deviation (Original PI – Estimated PI)</td>
<td>Difference between original PI value and program generated PI value</td>
</tr>
<tr>
<td>Time taken (in Milliseconds)</td>
<td>Time taken by the program processing</td>
</tr>
</tbody>
</table>
The above simulation results indicate that increasing the number of trials to large value helps to get more accurate value of PI.
Above experiment results indicate that when the number of processors is increased, the calculation time is decreasing comparatively. The iteration parameter is fixed as 1000 for the Monte Carlo PI approximation problem. Step by step, when more number of processors are assigned, result calculation time is decreased. Here, the least time is obtained, when the maximum number of 4 processors available is used in the experimental study.
6.4.4 Bioinformatics Grid Application – Sequence Alignment

Bioinformatics applications use computational methods and processing to manage, analyze and process large quantity of biological data sets. Researchers in bioinformatics require processing of large molecule databases for new drug and molecule discovery leading to applications which require parallel processing and high performance computing. Grid computing can provide useful infrastructure for execution of such applications. Features of bioinformatics applications which make them ideal for parallel execution on compute grids are [173]:

- Creation and management of large sized biological and molecular public databases like GenBank (Genetic Sequence Databank), Protein Data Bank (PDB) and various large and small molecule databases.
- Execution of compute intensive algorithms, E.g. to determine different relationships among members of molecular databases, molecular docking algorithms which predicts the orientation of one molecule with another to create a new molecule/complex for drug design.
- SIMD (Single instruction, Multiple Data) nature of tasks which needs to be executed millions of times with different data each time, e.g. comparing existing molecule sequence with millions of other molecules in the database

Sequence Analysis and Alignment

In bioinformatics, sequence analysis involves the use of analytical methods to study the Protein, DNA, RNA sequences to understand the features and structural properties. Various techniques are used like sequence alignment, molecule structure similarity searching against large biological databases etc.

Sequence alignment is the technique of arranging and comparing the sequences of protein, DNA, RNA to identify the regions of similarity that helps in inferring the functional, evolutionary or structural relationship between them.
Two types of alignment techniques are used: global and local. Global alignment is used for two sequences which are of same type and same length, and so the entire sequences are aligned by global alignment algorithm. Local alignment is used to find certain portion of the sequences which is matching among large different sequences. Global Sequence Alignment which is used to understand evolutionary relationships between sequences E.g. Human DNA vs. Mouse DNA has been implemented in the grid. The algorithm used for global alignment is Needleman-Wunsch algorithm which is based on dynamic programming approach [174]. The above compute intensive algorithm is parallelized and implemented in our grid test bed. The heuristic algorithm compares every pair of characters in the two sequences to be aligned and generates a best or near optimal alignment.

The working of the algorithm is as under [174]:

In bio-informatics, sequences are represented as sequence of characters.
DNA & RNA: |alphabet|=4
Protein: |alphabet|=20

**Sequence Alignment**

**Input:** two sequences over the same alphabet

**Output:** an alignment of the two sequences with results

Consider the two sequences to be aligned are as under:

Sequence 1: CTTAACT
Sequence 2: CGGATCAT

On alignment of above two sequences, one of the possible output is:

\[
\begin{array}{ccccccccccc}
C & - & - & - & T & T & A & A & C & T \\
C & G & G & A & T & C & A & - & - & T
\end{array}
\]

Position 1 2 3 4 5 6 7 8 9 10
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

The above resulting alignment of sequence 1 and 2 has below types of result:

1. Perfect match (at position 1, 5 and 10)
2. Mismatch (at position 6)
3. Insertion Gap (at position 2, 3, 4)
4. Deletion Gap (at position 8 and 9)

The Sequence Alignment Graph is as under:

![Sequence Alignment Graph]

Figure 6.18: Sequence Alignment Graph

The scoring scheme following in Global Alignment algorithm is

Matching = +2 \( w(x, y) = 2 \), if \( x = y \)
Mismatch = -1 \( w(x, y) = -1 \), if \( x \neq y \)
Gap (Insertion or deletion) = -2 \( w(-, x) = w(x, -) = -2 \)

\[
\begin{array}{cccccccc}
C & - & - & - & T & T & A & A & C & T \\
C & G & G & A & T & C & A & - & - & T \\
+1 &  -2  & -2  & +2  & -1  & +2  & -2  & -2  & +2  & =  -4 \\
\end{array}
\]
Optimal Alignment with Maximum Score

To find the optimal alignment with maximum score, a two-dimensional matrix $S$ is created. The element at $i$th row and $j$th column is indicated by $x_i, y_j$.

- Let $A = x_1, x_2...x_m$ and $B = y_1, y_2...y_n$.
- $S_{i,j}$: the score of an optimal alignment between $x_1, x_2...x_i$ and $y_1, y_2...y_j$
- With proper initializations, $S_{i,j}$ can be computed as follows [175]:

$$S_{i,j} = \max\left\{ S_{i-1,j} + w(x_i, -), S_{i,j-1} + w(-, y_j), S_{i-1,j-1} + w(x_i, y_j) \right\} \quad (6.3)$$

Initialization of the similarity matrix

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>-3</td>
<td>-6</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
<td>-18</td>
<td>-21</td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td>0</td>
<td>-3</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
<td>-18</td>
<td>-21</td>
</tr>
<tr>
<td>A</td>
<td>-6</td>
<td>-3</td>
<td>0</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
<td>-18</td>
<td>-21</td>
</tr>
<tr>
<td>T</td>
<td>-9</td>
<td>-6</td>
<td>-3</td>
<td>0</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
<td>-18</td>
</tr>
<tr>
<td>A</td>
<td>-12</td>
<td>-9</td>
<td>-6</td>
<td>-3</td>
<td>0</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
</tr>
<tr>
<td>A</td>
<td>-15</td>
<td>-12</td>
<td>-9</td>
<td>-6</td>
<td>-3</td>
<td>0</td>
<td>-9</td>
<td>-12</td>
</tr>
<tr>
<td>C</td>
<td>-18</td>
<td>-15</td>
<td>-12</td>
<td>-9</td>
<td>-6</td>
<td>-3</td>
<td>0</td>
<td>-9</td>
</tr>
<tr>
<td>T</td>
<td>-21</td>
<td>-18</td>
<td>-15</td>
<td>-12</td>
<td>-9</td>
<td>-6</td>
<td>-3</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.19: Initialization of the similarity matrix
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

Figure 6.20: Optimal Alignment with Maximum Score

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-3</td>
<td>-6</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
<td>-18</td>
<td>-21</td>
<td>-24</td>
</tr>
<tr>
<td>-3</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>-1</td>
<td>-4</td>
<td>-7</td>
<td>-10</td>
<td>-13</td>
</tr>
<tr>
<td>-6</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>-3</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>-2</td>
</tr>
<tr>
<td>-9</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>-5</td>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>9</td>
</tr>
<tr>
<td>-12</td>
<td>-1</td>
<td>-3</td>
<td>-5</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>-15</td>
<td>-4</td>
<td>-6</td>
<td>-8</td>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>-18</td>
<td>-7</td>
<td>-9</td>
<td>-11</td>
<td>0</td>
<td>-2</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>-21</td>
<td>-10</td>
<td>-12</td>
<td>-14</td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

Figure 6.21: Sequence Alignment Job Input Screen User Interface
The results of alignment of sequence 1 and 2 are displayed in the result text area and the alignment score obtained is 14. The alignment job is scheduled on all the available grid nodes matching the QoS requirements (if specified).

Figure 6.22: Sequence Alignment Job Output Result
Chapter 6: Prototype Implementation of QoS Grid Using Globus Toolkit

**Experiment Results**

Sequence Alignment Time required v/s Number of Grid Nodes Used

Sequence Size = 25

![Sequence Alignment Time v/s Grid Nodes Used](image)

Figure 6.23: Increase in number of characters in Sequence Alignment v/s Time required

Using variable number of Grid nodes and increasing the size of sequence

![Size of Sequence v/s Execution Time](image)

Figure 6.24: Size of Sequence Alignment v/s Execution Time