ALCHEMI VS GLOBUS: A PERFORMANCE COMPARISON

Md. Ahsan Arefin, Md. Shiblee Sadik, Serena Coetzee, Judith Bishop

Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, Department of Computer Science, University of Pretoria, Pretoria, South Africa

Email: arefin@csebuet.org, shiblee@afrigis.co.za, {scoetzee,jbishop}@cs.up.ac.za

ABSTRACT
Alchemi and the Globus Toolkit are open source software toolkits for implementing a Grid. Although both toolkits are designed for the same purpose, their architecture and underlying technology are completely different. Thus, a performance comparison of a Grid implementation in Alchemi with a similar Grid implementation in the Globus Toolkit will be interesting. We built a test bed to compare the performance of the two toolkits. This paper includes tables and graphs to illustrate the comparison.

1. INTRODUCTION
A server processing database requests from multiple clients is a common example of a distributed system. Using a single server machine in such a scenario often results in unwanted delays to its clients due to the processing load on the server. A supercomputer or any other multiprocessor computer can solve this problem, but these computers are not freely available and are often too expensive.

The power of a supercomputer can be simulated by applying Grid technology to the under-utilized resources of available computers on the Internet [4, 5, 7]. Various software toolkits are available to build such a Grid environment. Among them, the Globus Toolkit has been widely used in research all over the world for longer than a decade, while Alchemi is the latest addition to Grid software.

The focus in this paper is on the selection of suitable Grid software when implementing a Grid environment where processing speed and request response times are of importance. Ultimately the Grid is evaluated in terms of the applications, business value, and scientific results that it delivers, not its architecture [2, 4]. So, we used a server that processes multiple database requests in our comparison of the two toolkits with the same architecture.

2. ALCHEMI AND THE GLOBUS TOOLKIT
Alchemi and the Globus Toolkit are two open source software toolkits implementing a Grid environment [7, 6]. Alchemi runs on the Windows operating system in the .NET Framework [8] while the Globus Toolkit has its origins in Linux [3]. The complete Globus Toolkit is only available in Linux, but recent versions include some of its functionality also for Windows. The Globus Toolkit supports Java, C++, and Perl (among others) for the development of services [3]. But the Windows .NET framework is not supported for Globus. Alchemi provides an API for C# and C++ [8], and operates in the Windows .NET framework.

In our research we used Alchemi V1.0 and the Globus Toolkit V4.0.1. We used the C# API to develop services in Alchemi and Java when developing services in the Globus Toolkit.

Since the interpretation/execution speeds of programs in Java and C# differs, we set-up a control environment in which we compared the interpretation/execution speed of C# in Windows .NET with the interpretation/execution speed of Java on Linux. In the control environment, we executed a binary search in both C# and Java. We found that Java on Linux is 1.68 times slower than C# in the .NET framework in Windows. We used this factor when comparing the grid responses.

3. COMPONENTS OF THE TEST BED
An open, extensible and scalable system for querying a federation of heterogeneous distributed spatial data in Grid is feasible and aided by the emerging standards in grid computing [9, 10]. In our test bed, a grid is constructed from a number of different types of nodes (or hosts), each type playing a different role when a request from a client is processed. A Manager node and one or more Executor nodes that connect to
the Manager node are configured when constructing a desktop grid in Alchemi. This architecture is similar to the deployment of a grid entry portal and one or more executor nodes in Globus. The operation, responsibility and structure of the Manager/Entry Portal, the Executor/Worker nodes, the Client and the DataServer nodes in our test bed, as well as the data we used, are described below. Refer to Figure 1 for the architecture of the different types of nodes.

![Architecture Diagram](image)

**Fig.1. Distributed Components and their Relations**

3.1 Manager/Entry Portal Node

This node provides services associated with managing the execution of grid applications and their constituent threads. A Client node sends requests to the Manager/Entry portal node, which then distributes the jobs among the Executor/Worker nodes [7]. Threads are scheduled on a Priority and First Come First Served (FCFS) basis. In Globus this Scheduling is performed using a Gram Service [1].

3.2 Executor/Worker Node

Executor/Worker nodes accept threads and execute them. They are responsible for requesting data from the DataServer nodes, and processing the data that is received from the DataServer node [7].

3.3 DataServer Node

The DataServer node accesses the data in the database. A DataServer node receives a request from an Executor/Worker node and accesses the database as specified in the request.

3.4 Client Node

The top layer in Figure 1 represents the client in our test bed. A Client node represents any user on any PC that is connected to Internet. We implemented the client application using C# in Alchemi and Java in the Globus Toolkit. In fact, the client application is totally independent from the lower three layers in our test bed illustrated in Figure 1. The client application is neither Alchemi nor Globus specific, and can be written in any programming language [6]. The single purpose of the client application is to send requests to the Entry Portal from where they are forwarded onto the grid.

3.5 Database

The data used in our test bed consist of a single table with spatial address data. The format is shown in Figure 2. In our test bed the database contains around 2,550,000 rows.

![Database Table](image)

**Fig. 2: Fields in the Database Table**

3.6 The Comparison

In our comparison we used two scenarios:
- a database on a single DataServer node with high processing capacity; and
- a database replicated on two different DataServer nodes, each with high processing capacity.

For the comparison, we executed different queries. Here the result is shown for the following query. The Executor/Worker node requests data from the grid with the following SQL statement:

```
Select * from NAD where Province= 'Gauteng'
```
The comparison results for all other queries nearly follow the same response curves. On receipt of the data the Executor/Worker node iterates through return rows and calculates the number of rows. This processing is done to achieve parallel processing outside of the database.

4. RESULTS

The results of the comparison are shown in Table 1 and 2, and in Figure 3 and 4. Mathematically or theoretically the performance can be described as follows. Let

\[ n \] be the number of nodes (all equally powerful)
\[ x \] be the number of jobs assigned
\[ t \] be the time required to finish a job
\[ T \] be the total time
\[ t_s \] be the scheduling and networking delay time.

As with the use of fibre optics, the networking overheads in our test bed are negligible and for this small Grid environment the scheduling overhead can also be neglected. Now, if only one CPU is available then the total time is

\[ T = x \times t \]

But if \( n \) nodes are available, then we have

\[ \text{if } (n > x) \quad T = t + t_s \]
\[ \text{if } (t_s << t) \quad \Rightarrow T \approx t \]
\[ \text{if } (n < x) \quad T = \left(\frac{x}{n}\right) \times t + t_s \]
\[ \text{if } (t_s << t) \quad \Rightarrow T \approx \left(\frac{x}{n}\right) \times t \]

Thus, by increasing the number of Executor/Worker nodes, the processing time of a request decreases in a linear fashion, proportionally to the number of Executor/Worker nodes in the Grid. This is evident in the results shown in Table 1 and 2, and the graphs in Figure 3 and 4.

Table 1 shows the results for a single DataServer node where 10 jobs were submitted from the client application. For easy representation let

\[ N \] be the number of Executors
\[ M \] be the sum of all CPUs in Ghz
\[ T_a \] time required in Alchemi in sec
\[ \gamma \] multiplying factor (see section 2)
\[ T_{ar} \] relative time for Alchemi in sec
\[ T_{gr} \] relative time for Globus in sec

The graph in Figure 3 shows the actual comparison results for the table represented, and this requires some explanation.

Table 1: Scenario 1: Single DataServer Node

<table>
<thead>
<tr>
<th>( N )</th>
<th>( M )</th>
<th>( T_a )</th>
<th>( \gamma )</th>
<th>( T_{ar} )</th>
<th>( T_{gr} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.70</td>
<td>275</td>
<td>1.68</td>
<td>462</td>
<td>569.0167</td>
</tr>
<tr>
<td>2</td>
<td>3.41</td>
<td>225</td>
<td>1.68</td>
<td>378</td>
<td>263.2931</td>
</tr>
<tr>
<td>3</td>
<td>4.905</td>
<td>202</td>
<td>1.68</td>
<td>339.36</td>
<td>214.8697</td>
</tr>
<tr>
<td>4</td>
<td>6.408</td>
<td>198</td>
<td>1.68</td>
<td>332.64</td>
<td>207.0153</td>
</tr>
<tr>
<td>5</td>
<td>8.102</td>
<td>195</td>
<td>1.68</td>
<td>327.6</td>
<td>203.5577</td>
</tr>
</tbody>
</table>

However we expect that with more DataServer nodes in the Grid there will be a linear improvement in the performance of the Executor/Worker nodes. This will be tested in the second scenario.

A slightly different situation occurs when we use two DataServer nodes and submitting 10 jobs submitted to the Grid. The results from this scenario are shown in Table 2 and Graph 4. Here there is a performance improvement when the data is processed in parallel on the Executor/Worker nodes. Thus, for high volumes of data processing requests, the architecture with more than one DataServer node may provide best results.

Fig. 3: Scenario 1: Single DataServer – Number of Executor/Worker Nodes and Execution Time (\( T_{ar} \) and \( T_{gr} \))
Table 2: Scenario 2: Two DataServer Nodes

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>T_0</th>
<th>γ</th>
<th>T_{ar}</th>
<th>T_{gr}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.648</td>
<td>275</td>
<td>1.68</td>
<td>462</td>
<td>569.02</td>
</tr>
<tr>
<td>2</td>
<td>3.41</td>
<td>165</td>
<td>1.68</td>
<td>277.2</td>
<td>240.65</td>
</tr>
<tr>
<td>3</td>
<td>4.905</td>
<td>142</td>
<td>1.68</td>
<td>238.56</td>
<td>210.078</td>
</tr>
<tr>
<td>4</td>
<td>6.408</td>
<td>138</td>
<td>1.68</td>
<td>231.84</td>
<td>200.64</td>
</tr>
<tr>
<td>5</td>
<td>8.102</td>
<td>125</td>
<td>1.68</td>
<td>210</td>
<td>185.203</td>
</tr>
</tbody>
</table>

Fig. 4: Scenario 2: Two DataServer Nodes – Number of Executor/Worker Nodes and Execution Time (T_{ar} and T_{gr})

5. CONCLUSION

Our test bed is an example of Grid technology that shows how it can make our life much easier. Our main goal was to compare the performance of the widely used Globus Toolkit with the newly introduced Alchemi. From the experimental results, we can conclude that the performance of Globus is better than Alchemi when processing database queries in a grid environment.

ACKNOWLEDGEMENTS

This research is supported in part by the South African Department of Trade and Industry (www.dti.co.za) and AfriGIS (www.afrigis.co.za). The test bed was built on hardware provided by SDSL (www.sdsl-it.com).

REFERENCES


SOFTWARE

Alchemi http://www.alchemi.net
Globus Toolkit http://www.globus.org
Global Grid Forum http://www.ggf.org
Grid Café, http://www.gridcafe.nl