Laying the Foundation for the Investigation of the Quality of Service for Resource Management in Grid Computing

Marc A. Kapke and Dr. Paul J. Wagner (faculty mentor)
Computer Science Department
University of Wisconsin – Eau Claire
Eau Claire, Wisconsin 54701
{kapkema, wagnerpj}@uwec.edu

Abstract

By pooling the resources of virtually any functional machine, grids have quickly become a powerful, low-cost, attractive solution in a number of application domains. As a result of the surge in interest in grid computing, a variety of implementations have emerged. The availability of grid systems in commercial industry is providing profitable solutions to several very real problems including high-availability systems, failover scenarios, mass storage, data backup and duplication. The Globus grid framework has emerged as a leading solution providing industry, developers, and researchers a tool to develop systems to meet their individual needs and contribute to the evolving grid leader.

Our research is twofold. First, we worked to implement a grid service across a two system grid using the Globus toolkit framework. Second, we have begun to investigate the current status of resource allocation within grid systems.

1 Introduction

1.1 Overview

The majority of grid projects have been driven by the *quantitative* capabilities that a grid provides. Our interests lie in investigating the *quality* of service provided by a grid. First, we researched the current effectiveness of the Globus grid framework in distributing resources to assure quality of service in conjunction with the quantity of available resources. By benchmarking the grid against a real-world problem we identified room for innovation to deliver a higher quality of service than grids that are currently focused primarily on high quantitative service. Second, based on the benchmarking results, we are working to design and implement quality of service improvements within the Globus grid framework.

1.2 Grid Background

Conceptually, a grid is nothing more than a collection of computing resources that can perform tasks. The grid provides end-users with a single point of access to powerful distributed resources. There are some grid solutions available, some commercial and some open source. While each model varies slightly the overall concept remains the same. The low-cost and scalability of a grid provides an ideal environment where "no one lacks needed performance, yet nothing is wasted." [5] Grids are an ideal candidate for use in many different areas, including collaborative problem solving, data-mining, and video rendering. Projects such as SETI@Home have harnessed the quantitative power of the grid in their quest to discover extraterrestrial life. By allowing users across the globe to connect to the grid they have created an enormous 'supercomputer' at virtually no cost. In the SETI project, users' resources are utilized to help analyze enormous amounts of recorded radio telescope data [4].

Interoperability is the fundamental concern of grid computing. "Interoperability ensures that sharing relationships can be initiated among arbitrary parties, and accommodate new participants dynamically across different platforms, languages and programming environments" [1]. The use of standard protocols helps to assure interoperability during dynamic grid formation.

The Globus Alliance group [2] focuses on the research and development of the fundamental technologies behind the grid. They are located across the globe working together to contribute to the open-source Globus Toolkit. The toolkit provides research and development teams with software services and libraries for resource monitoring, discovery, overall management, security and file management.

Quality of service is a key aspect in creating the future of grid computing, but to date has not been heavily focused on. Providing grids with the ability to react to service demands by assigning the best combination of available resources to any given service based on standards, best practices, business costs and tradeoffs, and job priority will be the next

logical step in advancing grids from a primitive and dedicated tool to intelligent and flexible/reactive tool.

Other approaches have been taken to implementing and utilizing grids. The Oracle database management system has now integrated some grid functionality into its flagship database product, Oracle 10g [3].

2 Research

2.1 Globus Grid Service Implementation

The Globus toolkit provides programmers with a factory/instance approach to web services. The factory is responsible for maintaining multiple instances of a service, (see Figure 1, from [6]). The web client applications talk to the factory only during creation and destruction of a service instance. Any client calls to service operations go through an instance instead of directly to the factory. Note that clients are not bound necessarily to one instance of the service.

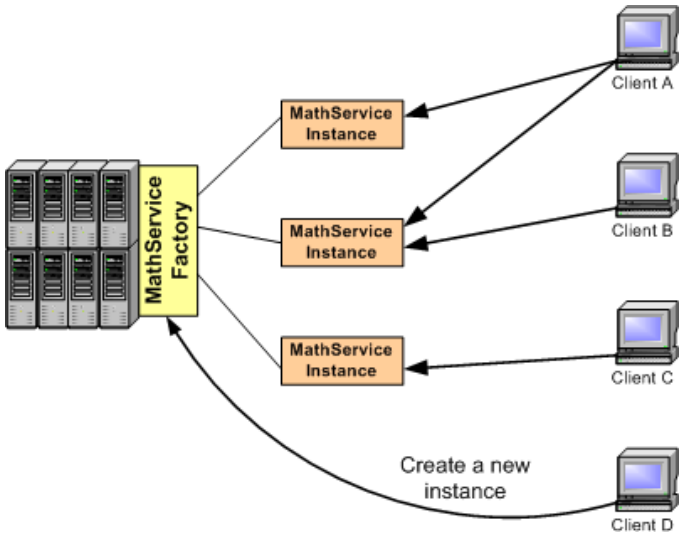

Figure 1 - Conceptual Architecture of a Grid Service

Much like a web service, a grid service consists of two essential pieces. The first piece of any grid service is its exposed interface. The exposed interface is the set of operations that web clients can call directly. The Grid service interfaces utilize an extension of the Web Service Description Language (WSDL) known as the Grid Web Service Description Language.

There are four main parts to the GWSDL interface. The first is to define the necessary namespaces for the grid service. Defining the correct namespaces allows access to all the necessary components in the GWSDL. For the factorial service we implemented it looks like this:

```
<definitions name="FactorialService"
targetNamespace="http://www.globus.org/namespaces/2004/02/FactorialService"
xmlns:tns="http://www.globus.org/namespaces/2004/02/FactorialService"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
xmlns:sd="http://www.gridforum.org/namespaces/2003/03/serviceData"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Figure 2 - Namespace definition for FactorialService GWSDL

In order to correctly define the interface to the FactorialService we had to include three main components: Types, Messages and Operations. The operations for a service are the exposed methods defined in the services custom PortType (Port type is the name given to expose the interface). All services extend the generic GridService portType and include service specific operations.

As a simple example, imagine that FactorialService has one method called Factorial(int inputValue). In order to correctly describe this function in the interface the GWSDL must include a portType definition and within the portType definition there must be a list of public operations.

```
<gwsdl:portType name="FactorialServicePortType" extends="ogsi:GridService">
        <operation name="getValue">
                <input message="tns:getValueInputMessage"/>
                <output message="tns:getValueOutputMessage"/>
                <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>
        <operation name="factorial">
                <input message="tns:factorialInputMessage"/>
                <output message="tns:factorialOutputMessage"/>
                <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>
</gwsdl:portType>
```

Figure 3 - PortType definition for a simple Factorial Service

Notice that the defined FactorialServicePortType extends from the more generic port type in the Open Grid Services Infrastructure (OGSI) namespace defined in Figure 2. Also, note there are two operations defined in the interface. The first is getValue, this method returns the factorial value computed by the service. The other is the input method factorial() which takes in a value and computes the factorial result. Note that for each method there are two required message types: input and output. There is also a required fault definition. The FactorialService GWSDL definition utilizes the FaultMessage definition provided in the OGSI namespace.

Note that in Figure 3 all message definitions are defined within the target namespace (tns). The next step in creating a GWSDL is to define the behavior of all input and output messages defined within the target namespace.

```
<message name="getValueInputMessage">
        <part name="parameters" element="tns:getValue"/>
</message>

<message name="getValueOutputMessage">
        <part name="parameters" element="tns:getValueResponse"/>
</message>

<message name="factorialInputMessage">
        <part name="parameters" element="tns:factorial"/>
</message>

<message name="factorialOutputMessage">
        <part name="parameters" element="tns:factorialResponse"/>
</message>
```

Figure 4 - Factorial Service GWSDL Message definitions

Figure 4 defines all necessary messages for the simple factorial service. Each message contains one 'part' which is defined by its name and what type of element that part is. In our example, all the parts are defined by complex types that also are defined in our target namespace, which means each element must be defined within the GWSDL too.

The element definitions are the final step to creating a GWSDL. By defining each of the parts as complex types you can specify the parameters for an operation in the element definitions.

```
<xsd:element name="getValue">
        <xsd:complexType/>
        </xsd:element>
        <xsd:element name="getValueResponse">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="value" type="xsd:int"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="factorial">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="inputVal" type="xsd:int"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="factorialResponse">
                <xsd:complexType/>
        </xsd:element>
```

Figure 5 - GWSDL Element Def inition for Factorial Service

Notice that for each element there are simple type definitions that correspond to the parameters for each operation.

After defining the GWSDL for the factorial service we had to create the corresponding Java implementation to meet the specs defined by the GWSDL interface. The implementation for the example factorial service had two public methods, one that takes in a value called factorial, and one that provides clients access to the current computed factorial value called getValue.

Service implementation classes must extend the OGSI GridServiceImpl class (this is referenced in the GWSDL definition as well). In addition, the service implementation must implement the exposed port type interface (the GWSDL interface), in my example the FactorialServicePortType.

```
public class FactorialServiceImpl extends GridServiceImpl implements FactorialServicePortType
{
	public FactorialServiceImpl()
	{
		super("Factorial");
	}
}
```

Figure 6 - Java Stub for Implementation of Factorial Service

Figure 6 represents the bare essentials for a service class definition. At this point, figure 6 suggests there are no exposed operations in this service. Since the GWSDL defines two operations, the Java implementation must include those as public methods.

```
protected int fact;
	public void factorial(int input) throws RemoteException{
		// TODO Auto-generated method stub
		fact = 0;
		fact = calcFactorial(input);
	}
	public int getValue() throws RemoteException{
		// return Factorial
		return fact;
	}
	// private recursive factorial method
	private int calcFactorial(int inputValue){
		int result = 0;
		if(inputValue < 2){
			result = 1;
		}
		else{
			result = inputValue * calcFactorial(inputValue - 1);
		}
		return result;
	}
```

Figure 7 - Factorial Service Operation Implementation

Figure 7 demonstrates how to implement the factorial functionality of the operations defined in the interface. All methods exposed by the GWSDL must throw RemoteExceptions (defined by FaultMessage in GWSDL). This factorial service will

persist the value of fact calculated by the factorial operation for access by clients through the getValue method. This simple example may not be the best to demonstrate the persistence features of a grid service because the instant a client calls factorial() again, the current persisted value is overwritten.

However, consider a system that tracks system transactions from millions of clients. Imagine as part of the system there is a service whose only purpose is to increment by one every time a transaction is invoked. Looking back at figure 1, with a grid service we can invoke multiple instances of the transaction counting service, all accessing the same counter in the factory. This is beneficial because with a standard web service all clients would have to go through the same instance which is a large bottleneck. By using grid services we can access that counter in a synchronized manner ensuring the integrity of the service data and eliminate the bottleneck.

2.2 Grid Service Client Implementation

After deploying the factorial service onto the grid service container, we need to provide a client application to access the operations in the service. This is fairly trivial using the Globus toolkit.

As an example, consider a simple command line client that takes in a set of arguments. The first argument is the URL of the service you wish to access, for example where the factorial service is deployed. The other argument in the factorial service is the input value to compute a factorial for. Extracting these values from the command line is trivial. The next step is to get a reference to the factorial service.

```
// Get command-line arguments
URL factorialURL = new java.net.URL(args[0]);
int inputValue = Integer.parseInt(args[1]);

//Get a reference to the FactorialService instance
FactorialServiceGridLocator
      factorialServiceLocator = new FactorialServiceGridLocator();
FactorialServicePortType portType =
      FactorialServiceLocator.getFactorialServicePort(factorialURL);
FactorialServiceImpl factorial = new FactorialServiceImpl();
factorial.setProperty(factorial.PORT_TYPE, portType);

// Call remote method 'factorial'
System.out.println("Calculating factorial of " + a);
factorial.factorial(inputValue);
// Get current value through remote method 'getValue'
int result = factorial.getValue();
System.out.println("Factorial is: " + result);
```

Figure 8 - Factorial Client Implementation

The middle section of Figure 8 demonstrates how simple it is to gain reference to the service you wish to utilize by taking advantage of the service locator inherited by extending the GridServicePortType. The only information you need is the location of the service instance.

The third section demonstrates the simplicity of calling the exposed methods in the service that are defined by the GWSDL. In my example service we are able to calculate the factorial of an integer and get its value.

2.3 Roadblocks

The major roadblock to the developing the groundwork for our research was setting up the grid environment. The biggest delay in deploying the environment was getting a version of the Globus toolkit to compile and run on Redhat Enterprise 3 because at the time of our research there was not a compatible binary version available. A lot of time was spent on installing and configuring the Toolkit properly, given the lack of a pre-existing binary version. Unfortunately, this kept us from making as much progress on the actual service issues as we had hoped.

3 Conclusion

Our research has provided insight into grid frameworks and the areas that can be improved. Our current work has laid the foundation for continued research into the grid framework and resource management. We are now at a point where we can further investigate the quality of service issues that were the original focus of our research.

4 Future Work

In the future we hope develop a benchmark service to expose the current resource allocation inefficiencies. There is a need for further analysis of the data provided by more complex benchmarking services to help expose these efficiency issues.

Our goal is to modify the current resource manager implementation within the Globus Toolkit to provide more efficient resource management. As we modify the algorithms we plan to continue to use our benchmarking service to track changes in efficiency. Our benchmarking data will be used to aid in our research and to help identify more areas for improvement within the Globus Toolkit.

As our research progresses we plan to contribute our improvements back to the open source Globus project. The data gathered from a benchmark system could be used to implement innovations in the current resource management system used in the Globus Toolkit. This implementation could then be applied back to the open source Globus project.

References

[1] “Anatomy of the Grid”; http://www.globus.org/research/papers/anatomy.pdf

[2] Globus Alliance; http://www.globus.com

[3] “Oracle 10g: Putting Grids to Work”;
http://www.oracle.com/technology/tech/grid/collateral/idc_oracle10g.pdf

[4] SETI@home (Search for Extra-Terrestrial Intelligence);
http://setiathome.ssl.berkeley.edu/

[5] “Sun Powers the Grid”;
http://www.sun.com/servers/entry/v60x/docs/whitepaper.grid.pdf

[6] “The Globus Toolkit 3 Programmer's Tutorial”
http://gdp.globus.org/gt3-tutorial/multiplehtml/index.html