Resource Monitoring with Globus Toolkit 4

Vijay Sahota[1], Maozhen Li[1] and Wenming Guo[2]

[1]*School of Engineering and Design*
Brunel University, Uxbridge, UB8 3PH
Email: {Vijay.Sahota, Maozhen.Li}@brunel.ac.uk

[2]*School of Software Engineering*
Beijing University of Posts and Telecommunications
Beijing, 100876, P.R.China

Abstract

The past few years have seen the Grid rapidly evolving towards a service-oriented computing infrastructure. With the OGSA facilitating this evolution, it is expected that WSRF will be acting as the main an enabling technology to drive the Grid further. Resource monitoring plays a critical role in managing a large-scale Grid system. This paper presents GREMO, a lightweight resource monitor developed with Globus Toolkit 4 (GT4) for monitoring CPU and memory of computing nodes in a Windows and Linux environments.

1. Introduction

The Grid [1] couples a global array of distributed resources that require constant monitoring if any integration and coordination is to take place. By processing this raw monitored data into useful information ensures optimal use of resources, pooling them for large capacity workloads, but still be able to work over a heterogeneous and geographically dispersed environment. Ease of use and accessibility is the major factor for rapid uptake and acceptance of Grid computing, but as usual the commercial aspect in providing services will have the greatest impact, but before the commercial sector can take any interest the Grid must provide a means of guaranteed service. Since monitoring is a key to organising any operations in a computing environment building a history of resource usage, one can perform some intelligent predictions on the state of the network in the near future, hence enabling the Grid to provide a guaranteed service from predicting which services will be available.

To reach as many users as possible with global coverage, the internet provides a universal foundation for communication whilst using existing technology. Its intrinsic property of interoperability is still an issue yet to be resolved in Grid computing. So far the general direction of using Web services [2] has been the main approach, to allow the Grid to operate over the internet whilst enabling utilisation through a standard Web browser, benefiting clients behind firewalls. The past few years have seen the Grid evolving rapidly towards a service-oriented computing infrastructure. The Open Grid Services Architecture (OGSA) [3] has facilitated this evolution. It is expected that Web Services Resource Framework (WSRF) [4] will be acting as an enabling technology to drive this further.

In this paper we present GREMO, a lightweight resource monitor using the Globus Toolkit 4 (GT4) [5], an implementation of WSRF standard, that included the WSN (Web Services notification) specifications [6] to support notifications. Currently, GREMO only monitors CPU and Memory usage, however its design is kept generic so that it can be applied to monitor any Grid resource. Many have tried and succeeded well in producing a monitoring system that works well on a large scale network, optimising programs that use minimal system resources whilst working towards a real time performance, such as Ganglia [7] and Network Weather Service (NWS) [8], but in most cases this entails a complex software set-up, restriction to a certain kind of network/operating system or a lack of the functionality to be as easy to use and accessible as a Web page. GREMO is implemented as a lightweight monitoring system requiring only a standard web server running. Using standard Web Service technologies, it can monitor resources in both Windows and Linux environments.

Proceedings of the Second International
Conference on Semantics, Knowledge, and Grid (SKG'06)
0-7695-2673-X/06 $20.00 © 2006 IEEE

The rest of the paper is organised as follows: Section 2 briefly reviews WSRF and WSN. Section 3 introduces the design of GREMO, and describes the main components of GREMO. Section 4 presents some experimental results to show the performance of GREMO. Section 5 concludes this paper.

2. WSRF and WSN

WSRF is a set of specifications that specify how to make Web services stateful amongst other aspects. The problem of where to store state involved the introduction for the concept of 'resources' (WS-Resource); a persistent memory for services which may reside in memory, hard disk or even a database. Each resource uses a unique address termed 'endpoint reference' to isolate resources from services enabling other services to use them directly with out having to go through the parent service, given this the introduction of other useful functions were also created.

- WS-Resource Lifetime – manages resources by setting a life time
- WS-Resource Properties (RPs) - many elements to a resource, similar to an object.
- WS-Service Group- enables grouping certain services to aid searching for them.
- WS-Base Faults- returning error exception that may be produced by WS-Resource.
- WS-Addressing- actual address given to services and resources rather than URL, enables one to use resources or Web services independently.

Finally, but key to creating a truly independent running service, the WSN system allows services to independently notify an authority via a SOAP [10] message when changes in a resources occur. Replacing the need for an authority to systematically poll for monitoring data, resulting in the inherent saving in time and bandwidth and with no need for special network conditions. Quite simply having created and invoking the resources like standard Web services, clients can easily be created to modify these RPs in relation to the monitored resource given its qualified name (qName) which is a concatenation of the resources namespace and RP name as a qName type.

An authority can then use their own (client) End Point Reference (ERP) to become a subscriber, and given the qName from which to receive notifications registration can be set, where a listener client will act on received notifications. Once a notification is received, EPR and RP's qName (from the sender) along with its new value can be extracted form the message and then be processed. The addition of this new functionality also means that the WSDL [9] documents also need to show its descriptions of a RP, straying away from the standard format, List 1 shows the additional code need for the WSRF standard.

```
<portType name="RegPortType"
   wsdlpp:extends="wsrpw:GetResourceProperty
   wsntw:NotificationProducer"
 wsrp:ResourceProperties
         ="tns:RegResourceProperties">

   <operation name="cname">
      <input message="tns:CnameInputMessage"/>
      <output message="tns:CnameOutputMessage"/>
   </operation>

</portType>
```

List 1: WS-Resource property definition in WSDL.

Note that the `wsrp: Resource Properties` attribute of the `portType` element specifies what the service's resource properties are. The resource properties must be declared as a type where the monitoring state information is kept. Firstly the 'wsdlpp:extends' attribute allows the use of predefined port types, in this example we have used both the 'get resource' & 'send notification' with bindings automatically created by GT4, so there is no need to specify in the WSDL code.

3. GREMO Architecture

Figure 1 shows GREMO architecture the following sections describe each GREMO components.

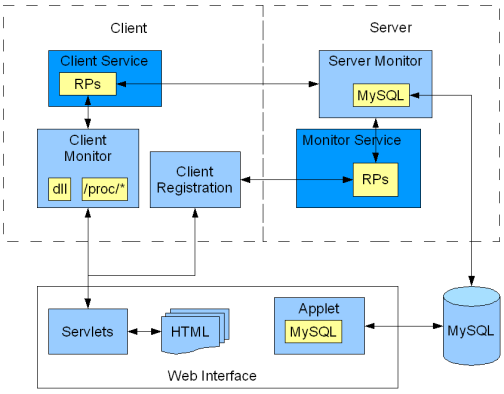

Figure 1: GREMO architecture.

3.1 Client Service

This service runs locally, its main function is to provide RPs that represents the local resources being monitored along with methods to modify them. The notification system is used to alert the subscriber of any changes in these RPs. In this case the service here only has to send out notifications (requiring a web container). Each client with its own service means they have

their own set of RPs. This approach means that there is not a single service (server) having to be constantly updated for each user, giving rise to unnecessary instance and processing problems on the receiving side, rather having the client(s) send the notifications. The server simply processes notifications to reduce strain and improve flow by not having to call and wait for a result/ response, not to mention the delay incurred when a client gets disconnected abruptly.

3.2 Server Service

Similar to the client service, the Server Service provides RPs that represent registration information of clients, with notifications sent when a new user registers. Every time the service is invoked a new set of RPs are created (service instances). Each client has its own set of RPs differing from the Client Service, here one main service which processes all the registration information. Since registration is less periodic than the actual monitoring, this seem the most economic way to create this registry service.

3.3 Client Side Monitoring

Client registration must take place first, given the EPR of the Server Service. The client then invokes the Server Service modifying its RPs allowing the server monitor to use this information to subscribe to the monitoring RPs managed by client. In a similar fashion, the client monitor modifies its Client Service's RPs in accordance to local monitored resources at given set intervals. Both Windows and Linux environments are accommodated to ensure cross platform functionality. Since the information being monitored is CPU and MEM, this information cannot be directly accessed through a Java Virtual Machine (JVM), hence code native to the OS is need. In the case for the Windows part of retrieving monitoring data, pre-complied 'C' dll files were used along with Java Native Interface (JNI) to access them, where as in the case for Linux such values have to be calculated using the /proc/ virtual file system.

3.4 Server Side Monitoring

Here is where the bulk of the monitoring information is processed. The code is a Java application that uses the Server Service RPs as registration information. Subscription to these RPs allow the monitor to subscribe to new users that have registered (monitoring data) as well as adding the registering data to a buffer, constituting of several vectors representing the monitored data including IP addresses, usage values of CPU and memory.

Using the IP address attached on the notification message as a primary key, the monitor modifies its buffers accordingly, whilst adding the values to the mySQL database. In a similarly fashion de-registration follows the same pattern. Fundamental to this tool's functionality is keeping a record of all users & their resources, using their IP addresses as an index for the buffers that are implemented using multiple vectors.

3.5 Storing Monitoring Data

A mySQL table is used as a persistent storage for the monitoring data, as it would be inefficient to keep over 200 values (integer in memory) for each user that can potentially run into the thousands! From keeping the most recent 100 values (each for CPU and memory) a relative history can be produced along with other useful information, but this is only temporary since when a user logs-off their entry is removed from the table.

Used as a buffer for the Server Monitor, information in this table uses IP address as a primary key index. Essential in acting as a temporary buffer is the ability for data to continuously loop around the set 100 fields given for each, monitored type. This means keeping a counter for each monitored resource, checking if the condition has reached 100 (and consequently around wrap back to 1), before executing a mySQL update.

3.5 Web Interface

HTML Web pages are used for both registering and monitoring clients. In this case an HTML form is used as an interface to a Java Servlet which in turn uses the Client registration class, and Client monitoring class to start monitoring, with a DHTML page to update the client on the resources they were monitoring. Using similar code as in the Server side monitor an applet version uses mySQL connector is created allowing a remote administrator to view the current usage of the monitoring service.

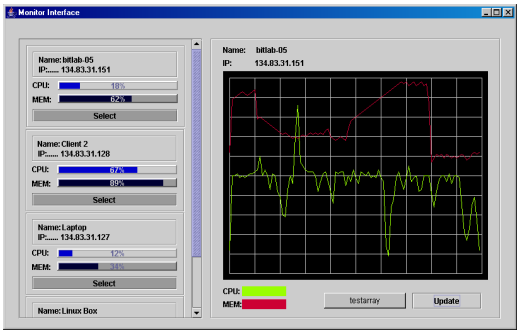

Figure 2: A snapshot of GREMO.

4. GREMO Performance

GREMO is implemented with GT4 on both Windows and Linux platforms. Figure 2 shows a snapshot of GREMO. Ideally GREMO could handle many hundreds of subscribed users, but in the real case there always is usually a limiting factor. GREMO has to processing a large number of SOAP notification messages. Having done a number of experimental tests, using two Pentium IIII workstations running Windows XP, both with 512Mb of RAM running Globus 4.0.0 container, Apache Tomcat 5.0.28 and mySQL 4.1.15 (server monitor only). Tests were carried out with a client sending multiple notifications in burst of 10-500 with the resulting average delays, lags & processing times recorded shown in figure 3.

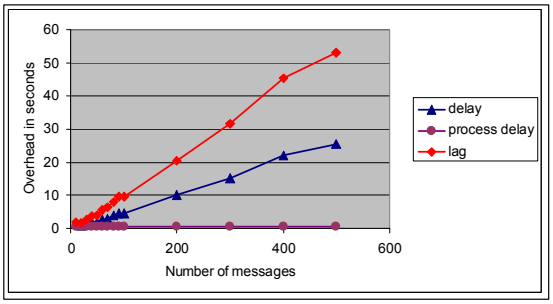

Figure 3: GREMO performance.

Even though testing occurred on a local network, considerable lags and delays can be seen which seem to increase in a linear fashion and start to become quite considerable after burst of 100 messages. Ideally a burst of 40 messages producing a delay and lag of 1.7 and 3.7 seconds respectively seems acceptable. This would be the upper operation limit in this service, making it not feasible to have more than 40 users registered. Note that these delays are for 40 instantaneous whereas 40 users may send 40 messages over a period of time and reduce such delays.

5. Conclusions and Future Work

In this paper we have presented GREMO and have discussed its implementation using the GT4-WSRF notification system. By modifying resource properties of a Web service from which notifications are produced, monitoring data can be logged in a grid environment. This

service offers granularity in that subscription is required and is dynamic, letting the GREMO perform independently without relying on other services to provide this information, as well as keeping track of registered users locally.

Having defined a basic structure the possible uses are widespread; As far as performance goes, we have been using the GT4 notification, which uses Apache Axis. Since at any one time a maximum of two integers are sent, the overhead data wise seems excessive, however, studies have shown that the actual conversion to and from ASCII data is very time consuming leading to performance decreases when the amount of data is increased [11]. Keeping this to a minimal level is beneficiary not only in processing time but also when network traffic is heavy. Time stamping of data to overcome any loss in accuracy history building still has to be implemented.

Taking into account the results shown current implementation of GREMO would be limited to around 40 users. The results also show a steady increase in lag and delay whilst processing delay remain constant suggesting that the GT4 container was processing these notifications at one time, storing the rest in an internal buffer. Work to make this operate in a multiple instance fashion (multi-threaded) will need to be carried out if this is to be a feasible solution. In addition, the building of a large history of resource usage would be the next logical step enabling external services to access this information to aid in performance prediction and job scheduling, producing guaranteed execution times of jobs submitted.

References

[1] I. Foster and C. Kesselman, The Grid, Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, USA, 1998.

[2] Web services, http://www.w3.org/2002/ws/

[3] Open Grid Services Architecture (OGSA), http://www.globus.org/ogsa/

[4] Web Services Resource Framework (WSRF), http://www.globus.org/wsrf/

[5] Globus toolkit 4 (GT4), http://www.globus.org/toolkit/

[6] Web service notification, http://www-128.ibm.com/developerworks/library/specification/ws-notification/

[7] Ganglia, http://ganglia.sourceforge.net/

[8] Network Weather Service, http://nws.cs.ucsb.edu/

[9] Web Service Description Language (WSDL), http://nws.cs.ucsb.edu/

[10] SOAP, http://www.w3.org/TR/soap/

[11] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, M. J. Lewis: Toward Characterizing the Performance of SOAP Toolkits. GRID 2004: 365-372