Performing Grid Computation with Enhanced Web Service and Service Invocation Technologies

Gang Xue, Graeme E. Pound, and Simon J. Cox

Southampton Regional e-Science Centre
School of Engineering Sciences
University of Southampton
Highfield, Southampton, SO17 1BJ, UK
{gx,gep,sjc}@soton.ac.uk

Abstract. Exploitation of Web service technologies is being attempted in various areas of Grid computing. In our effort to perform Grid computation tasks based on the Web service enabled job submission system, a series of new technologies have been adopted and developed in order to achieve better functionalities, performance, and seamless integration with existing Grid computing environment. This paper presents work with detailed descriptions of new Web service and service invocation technologies, as well as demonstrations of how they are deployed in a job submission service. Experiences with these technologies are also provided, accompanied by results of tests on some of the enhanced service functionalities for evaluation and reference.

1 Introduction

Progress in the research and development of Grid computing has given rise to numerous innovative applications of computer technologies in various science and engineering domains [1]. A significant and elementary Grid functionality often exploited in these applications is the organisation and delivery of spare compute power from resource providers in heterogeneous computer environments. It is often used to solve computationally intensive problems, in a way similar to how electrical power is delivered and consumed. Such functionality is usually implemented as job submission services, through which computation tasks are dispatched to resource sites, performed, monitored, and managed. Well-known examples of this technology can be found in the GRAM system of Globus and the UNICORE system.

Early attempts of job submission services were often restricted to specific selections of programming tools and platforms, or particular usage scenarios. While the desired functionality was still achievable, there could only be limited successes as the basic requirement of interoperability for Grid computing was hardly met. New solutions were called for, which was answered by the emergence of open-standard XML Web service technologies in the Grid scenario.

The maturing and standardisation of the elementary Web service technologies – XML, SOAP, XSD and WSDL – have drawn great interest from the Grid community for their capability of standardising Grid technologies, which is essential to overcoming heterogeneity in the Grid environment. Extensive and intensive investigations were made on the applications of Web services for Grid computing, which in general showed that such applications are viable [2,3]. The same conclusion can also be drawn from the announcement and acceptance of the Open Grid Service Architecture (OGSA) [4]. In earlier work [5], we demonstrated that XML Web service technologies could be applied to the construction of job submission service with high interoperability. Through further attempts on putting Web service based job submission into practical use, we identified a number of problems, which need to be solved with enhancement to the technologies before successful Grid computation can be performed.

One easily identifiable issue with a Web service based job submission service is the degraded performance. Compared to binary-based messages, the network overhead and the cost of message processing with the use of XML are significantly higher [3,6]. While performance penalties from submission requests and resource negotiations are arguably tolerable, especially for jobs with long execution times, huge costs brought by the transmissions of large job files and result data files in SOAP and base64 XML have caused great concerns. We believe that in order to reduce the unnecessary performance penalties, an additional standard message format is needed to facilitate the transmission of attachments along with SOAP messages. In our recent work, we have attempted to apply the latest progress in the development of such technologies to our job submission service, which is described in detail in the next section.

Like all Internet based computer applications, security management is one of the most common concerns for Grid computing and the Grid job submission service. However, no standard security mechanism has been defined by basic Web service technologies. In addition, in order to facilitate the use of the job submission service, the service needs to be compatible with GSI [7], the common security infrastructure in Grid computing. Our work demonstrates the adoption of the candidate Web service security standard for the job submission service, and shows how GSI or other security mechanisms can be integrated.

Experience with Grid computation through our job submission service shows that Web service technologies, especially the service invocation technologies, need to be improved in order to achieve the transparency and interoperability desired by Grid computing. Currently, Web services are normally consumed in a language API style: in most of today's Web service tools, the common practice is to get the WSDL file of target Web services and generate a service proxy based on it, which will then be called in the client program just like normal language APIs. While the development work is facilitated, the disadvantage is obvious: the service and the client are still tightly coupled by the interface definition, and it is impossible for client applications to consume other services with different interfaces, or adapt to changes on the service side without undergoing major modifications. In our client tool for the job submission service, we take advantage of the simple and standard SOAP protocol, which allows layered processing with independent intermediaries, to develop a new service invocation mechanism, which is implemented as a chain of SOAP filters. With this tool, client applications are provided with consistent and transparent access to Grid computation resources.

In this paper, work on performing Grid computation with Web services is described with two parts: firstly we consider the job submission service and secondly we discuss the implementation of the client tools. The descriptions were followed by evaluation and practical experience with the system. In the final part, we draw our conclusions and describe our future work briefly.

2 Enhancements to the Grid Job Submission Service

Our previous implementation of a job submission service with Web service technologies has provided users with basic functionalities for carrying out Grid computation tasks [5]. Users can submit computation tasks with specified resource requirements, monitor the job execution, perform basic job management operations, and retrieve job results through the exchanges of standard XML/SOAP messages. In order to integrate our job submission service with common Grid computing environment such as Globus, so that standard access to general Grid systems and resources can be provided, we have applied several recent new Web service technologies to enhance the original system.

2.1 Exploiting DIME for Data Transmission in Web Service Interactions

Direct Internet Message Encapsulation (DIME) [8] is a MIME-like new specification designed mainly for the transmission of SOAP messages together with additional attachments, such as binary files, XML fragments, and perhaps other SOAP messages, using standard transport protocols like HTTP and TCP. It defines a standard message structure in which data of various types that do not fit expediently or efficiently into the XML format can be contained and transmitted along with the SOAP messages. Compared to data transfer with SOAP and base64 XML, using DIME will bring significantly higher efficiency and flexibility. The DIME specification has been submitted to the Internet Engineering Task Force (IETF) [9].

Just like MIME, a DIME message is comprised of a number of DIME records, which are similar self-describing data sections with headers of binary information used for message parsing. The structure of a DIME record is shown in Fig. 1, and detailed descriptions of the record fields can be found in [10]. Compared with MIME, DIME defines a simpler message format, which does not allow the inclusion of extra metadata with custom message headers. It only contains information about the length and encoding of the message header fields and payload. While being less flexible, this ensures faster and more efficient processing of the messages. And when used together with SOAP, additional information could be delivered within the SOAP message.

When applying DIME for data transfers in the job submission service, the SOAP message signalling the data transfer operation is contained in the first DIME record. All attached data files are contained in the subsequent records and are identified by UUIDs (Universal Unique Identifier) in the ID fields of the corresponding DIME records. The attachments are cross-referenced in the SOAP message by the UUIDs, with the file names and sizes also specified. This enables a check outside the DIME protocol to make sure of the data integrity. In addition, in order to indicate the use of DIME, extensions to the WSDL file of the job submission service have been added with reference to the corresponding standard [11]. A sample DIME message used in the data transfer operation and the piece of WSDL with DIME extensions can be found in Figs. 2 and 3. In our service implementation, the layout attribute of the DIME message is set to *http://schemas.xmlsoap.org/ws/2002/04/dime/closed-layout*, which specifies that all parts of the DIME message should be referenced in the SOAP message in proper order.

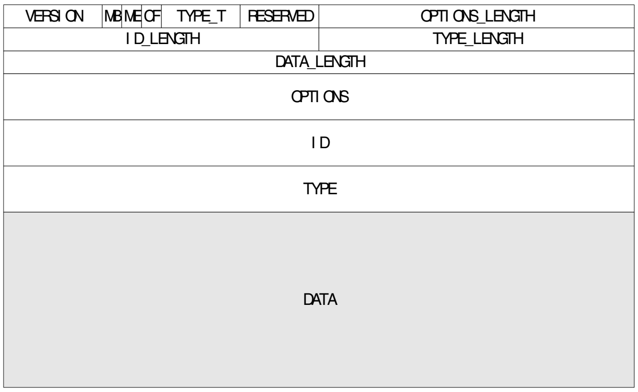

Fig. 1. The DIME record structure

```
00001100000100000000000000000000
00000000000000000000000000101000
00000000000000000000001001011011
http://schemas.xmlsoap.org/soap/envelope
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ref="http://schemas.xmlsopa.org/ws/2002/04/reference"
>
  <soap:Body>
    <DIMEJobFileTransfer
      xmlns="http://draco.sesnet.soton.ac.uk/ComputationService2/">
      <JobID>8fa464fc-a300-424b-9827-45b3bb223127</JobID>
      <FileNames>
        <file ref:location="uuid:fd25bb9e-7fd2-4a0f-8cb4-
                            d6db005e443a">job.exe</file>
        <file ref:location="uuid:3c9d6fa9-50ab-4da3-b859-
                            db4934f3243f">library.dll</file>
      </FileNames>
      <FileSizes>
        <size>1024</size>
        <size>1048576</size>
      </FileSizes>
    </DIMEJobFileTransfer>
  </soap:Body>
</soap:Envelope>
00001000000100000000000000000000
00000000001010010000000000010111
00000000000000000000010000000000
uuid:fd25bb9e-7fd2-4a0f-8cb4-d6db005e443a
application/macbinhex40
<<1024 bytes of binary data for job.exe>>
00001010000100000000000000000000
00000000001010010000000000010111
00000000000100000000000000000000
uuid:3c9d6fa9-50ab-4da3-b859-db4934f3243f
application/macbinhex40
<<1 MB of binary data for library.dll>>
```

Fig. 2. Sample DIME message for job file transfer

```
<binding name="JobSubmissionSoapDIME" type="s0:JobSubmissionSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
          style="document"/>
  <operation name="DIMEJobFileTransfer">
    <soap:operation
      soapAction="http://draco/ComputationService2/DIMEJobFileTransfer"
      style="document" />
    <input>
      <dime:message
          layout="http:/schemas.xmlsoap.org/ws/2002/04/dime/closed-layout"
          wsdl:required="true"/>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

Fig. 3. WSDL extensions for DIME in the job submission service

In order to achieve a better understanding of the DIME performance, a number of tests were carried out, comparing data transfers with HTTP, HTTP+DIME, and SOAP with base64 XML. The results are shown in Fig. 4. The advantage of DIME over normal SOAP data delivery is clearly demonstrated. It is mainly because the data can be placed in DIME messages directly without additional encoding, which is unavoidable for SOAP because of the SOAP message encoding style.

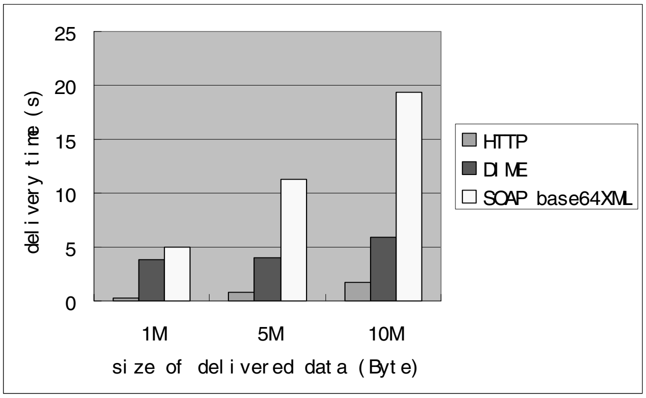

Fig. 4. Test of performance in data transmission

2.2 Using WS-Security with the Grid Security Infrastructure

Since no formal security framework was defined by the basic Web service technologies such as SOAP and WSDL, security mechanisms of the job submission service had to be built based on the underlying transport protocol – HTTP. While basic message authentication, integrity and privacy can be assured, such solution cannot provide satisfying features for Grid computing. One reason is that in some sophisticated operations, message exchanges may involve a route more complicated than the end-to-end HTTP connection, and may even involve different transport protocols. Another important reason is it does not integrate well with GSI. The Grid Security Infrastructure (GSI) [12] is a security mechanism based on public key cryptography, and is widely supported by the Grid community. In order to adopt GSI in the job submission service, a standard mechanism is needed for the delivery of GSI certificates and proxy certificates, which is the central part of GSI authentication. As a result, an implementation of the recently proposed Web services specification, WS-Security [13], has been provided in our service.

WS-Security defines a message level security mechanism, which is independent of the transport methods. Instead of defining a whole new solution, WS-Security focuses only on specifying how security information should be embedded in a SOAP message. It is therefore an open protocol, which allows existing security solutions, including Kerberos, X.509, and GSI to be leveraged.

When delivering GSI user credentials in WS-Security, the user certificates or proxy certificates are contained in a *BinarySecurityToken* element, and is placed in the *wsse:Security* SOAP header element. Since the GSI user certificates are encoded in the X.509 format, they can be directly treated as standard WS-Security binary security tokens[1]. As for GSI proxy certificates, an extension to WS-Security is defined for the job submission service, which is illustrated in Fig. 5.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ref="http://schemas.xmlsopa.org/ws/2002/04/reference"
>
  <soap:header>
  <wsse:Security
      xmlns:wsse=http://schemas.xmlsoap.org/ws/2002/07/secext
      soap:mustUnderstand="1">
    <gsi:BinarySecurityToken gsi:ValueType="GlobusProxy"
       Id="dab19bd1-a680-4a98-aa81-562e0cb48e70"
       xmlns:gsi="http://www.globus.org/gsi">
        MIIB3TCCAUYCAwnAiDANBgkqhkiG9w0BAQQFAD...
       ..........
     </gsi:BinarySecurityToken>
    </wsse:Security>
  </soap:header>
  <soap:Body>
    <JobSubmissionRequest
      xmlns="http://draco.sesnet.soton.ac.uk/ComputationService2/">
      ..........
    </JobSubmissionRequest>
  </soap:Body>
</soap:Envelope>
```

Fig. 5. Using GSI with WS-security

In addition to authentication, WS-Security also defines formal methods for encryptions of important messages parts. The service and users can therefore use information from the user accounts (the password) to encrypt/decrypt the security tokens, so that user privacy can be protected. The GSI user certificates can also be applied in asymmetric encryptions of important job information, such as the job handler, with the public key in the certificate for encryption and the private key held by the service client for decryption.

[1] The WS-Security specification only defines three value types for binary security token: *wsse:509v3*, *wsse:Kerberosv5TGT*, *wsse:Kerberosv5ST*.

2.3 Integration with General Grid Computing Environment

The purpose of the job submission service is to provide standard, transparent access to computation resources in various Grid computing environments. One of the currently most popular Grid environments is the Globus system. With the help of its middleware collection, which provides core Grid functionalities including security (GSI), resource allocation (GRAM), data transfer (GridFTP), and resource information service (MDS), the Globus Toolkit [14] enables the construction of computational grids through the aggregation of resources that are presented as Grid services. As compatibility with GSI has been achieved, it is feasible to integrate the job submission service with the Globus environment, and therefore make the resources managed by Globus accessible to client applications in a programming language and platform independent fashion. This integration also solves some of the firewall problems associated with Globus caused by proprietary network port settings in Globus system components such as GASS and GridFTP.

The integration with Globus was accomplished with the help of the Commodity Grid (CoG) kits [15], which provide core Globus functionalities as sets of client APIs in 'commodity technologies' including Java, Python, CORBA and Perl. In our work the Java CoG kit [16] is applied, as it is the most suitable one to work with our service.

3 The Client Tool for Service Oriented Grid Computing

As explained in the introduction, the language API-styled Web service invocation technologies are not suitable for the Grid environment, which is vast in scale and anarchic in nature. The true value of applying Web services for Grid computing is that it provides a way of interaction between independent components that share a common understanding of operational semantics, but are loosely-coupled at the interface level. Web service client tools used in Grid computing must therefore be compliant with this feature. As a result, our client tool for the job submission service has been implemented in two parts - the client utility and the message processor.

The client utility exposes job submission functionalities to application programs that need to perform computation tasks on the Grid. Different form normal service proxies, it only represents the minimum semantics of the job submission operations and procedures, and bears no information about the target service. The utility is therefore completely independent of the implementation of the computation resources, and can remain unchanged in spite of the highly changeable Grid environment.

When the client utility is called, the operation instructions are passed on to the message processor, which is responsible for the underlying interactions with targeted services, and feeding the results back to the utility. The message processor does not have a fixed composition. It is implemented as two dynamic chains of input and output message filters. Each filter is responsible for the process of a specific message part, or even the entire message. Important filters include the SOAP message handler, the WS-Security handler, and DIME handler, etc. The filter chains are formed during the runtime based on the information loaded from a configuration file, which is detached from the client applications and can be modified to add or remove message filters so as to adapt to any potential changes.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <soap>
     <outputfilters>
        <filter name="jobsubmissionservice.soapprocessor"
                   assembly="C:\windows\system32\SOAPFilters.dll" />
        <filter name="WSSecurity.GlobusCertificateOutputFilter"
                   assembly="C:\windows\system32\SOAPFilters.dll" />
     </outputfilters>
   </soap>
   <dime>
      <outputfilters>
        <filter name="dime.dimeoutputfilter"
                   assembly="C:\windows\system32\DIMEFilters.dll" />
      </outputfilters>
   </dime>
</configuration>
```

Fig. 6. Configuration file for the client tool message processor

Figure 6 shows a sample configuration file, which provides the full names of the filters and the locations of the libraries where the filter classes are contained. In the current implementation, the information is used by object reflection technology to dynamically generate filter instances and compose the filter chains.

A complete view of the client tool for the job submission service can be found in Fig. 7. In addition, it is necessary to point out that the filters for the message processor are not restricted to Web services messages. Custom filters can also be developed to enable access to other computation services using the same client tool.

4 Exemplar of Grid Computing with Enhanced Web Services

In order to examine how our enhancements to the job submission service and the integration with the Globus system work, the client tool, the job submission service and a Globus system were put together to create a sample Grid computing scenario, which is illustrated by the following figure.

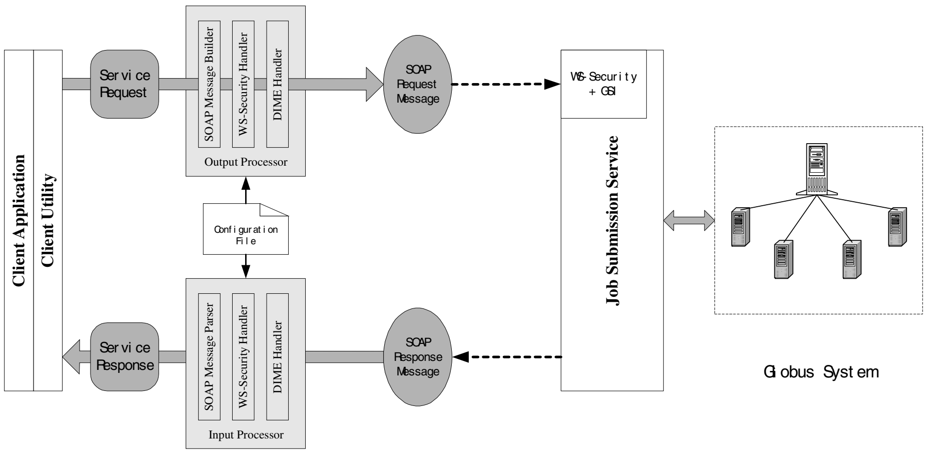

Fig. 7. The Grid computing scenario

A relatively simple computation task [2] was created and carried out under different conditions to evaluate the effects on job submission operations with various service technologies. The test results were shown in Fig. 8.

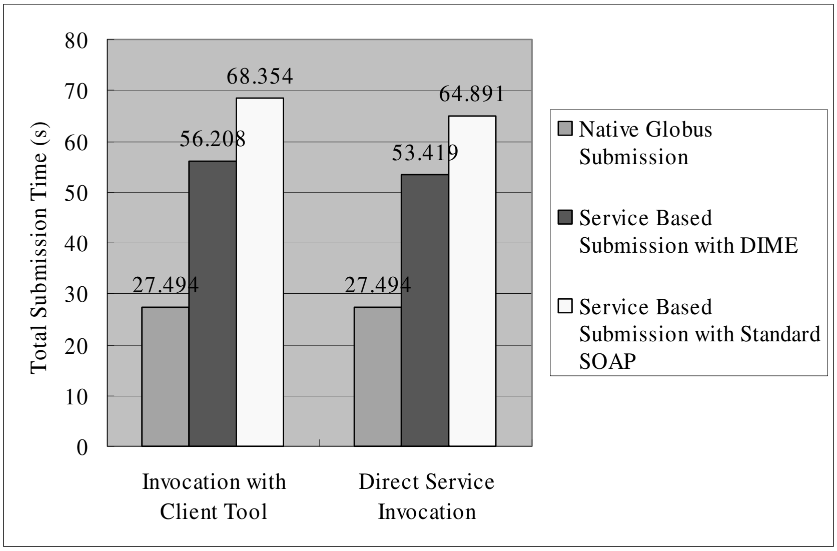

Fig. 8. Test results from sample job submissions [3]

Although the results might vary due to differences in network conditions, computer system statuses and service implementations, they in general show the cost of Web service based job submission, and the difference in performance of the new technologies. It also shows that the performance penalty of the filter-based service client tool is less than anticipated.

5 Conclusion and Future Work

Recent developments in Web service technologies have provided better solutions to issues unsolved by the basic standards. In this paper, we discussed and demonstrated how these new technologies can be applied to improve the job submission service for Grid computation. In the next stage of our work, we will try to extend the applications to other Web service enabled components and operations on the Grid, while keeping the services up-to-date with the latest developments in these technologies.

[2] The job has a typical composition of an executable, a runtime library and an input file with the total size of 8365961 bytes.

[3] The test is conducted on a 100M LAN, with the job submission service running on a server with a 900MHz CPU and 768M memory. The job submission procedure includes submission request, job file transfer and job-start notification.

References

[1] The GEODISE project: http://www.geodise.org

[2] Satoshi Shirasuna, Hidemoto Nakada, Satoshi Matsuoka, Satoshi Sekiguchi: Evaluating Web Services Based Implementations of GridRPC. Proceedings of HPDC-11, Edinburgh, Scotland, 2002.

[3] Kenneth Chiu, Madhusudhan Govindaraju, Randall Bramley: Investigating the Limeits of SOAP Performance for Scientific Computing. Proceedings of HPDC-11, Edinburgh, Scotland, 2002.

[4] I. Foster, C. Kesselman, J. Nick, S. Tuecke: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[5] S.J. Cox, M.J. Fairman, G. Xue, J.L.Wason, and A.J. Keane. The Grid: Computational and Data Resource Sharing in Engineering Optimisation and Design Search. IEEE Proceedings of the 2001 ICPP Workshops, Valencia, Spain, September 2001.

[6] Dan Davis, Manish Parashar: Latency Performance of SOAP Implementations. IEEE Proceedings of CCGrid, Berlin, Germany, May 2002.

[7] Grid Security Infrasturcture. http://www.globus.org/security/

[8] Direct Internet Message Encapsulation. http://www.ietf.org/internet-drafts/draft-nielsen-dime-02.txt

[9] Internet Engineering Task Force (IETF). http://www.ietf.org

[10] Jeannine Hall Gailey: Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation. MSDN Magazine, 12/2002. http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx

[11] Mike Deem: WSDL Extension for SOAP in DIME.

[12] http://www.gotdotnet.com/team/xml_wsspecs/dime/WSDL-Extension-for-DIME.htm

[13] Grid Security Infrastructure. http://www.globus.org/security/

[14] WS-Security. http://www-106.ibm.com/developerworks/library/ws-secure/

[15] The Globus Toolkit. http://www.globus.org/toolkit/

[16] Commodity Grid Kits. http://www-unix.globus.org/cog/

[17] Java CoG Kit version 0.9.13 . http://www-unix.globus.org/cog/java/index.php