Extending Globus Toolkit Java WS Core to Support Reliable Grid Messaging Services

PING-JER YEH[†], WINSTON LO[‡], YUNG-YU CHEN[*], SHYAN-MING YUAN[†]

Department of Computer Science
National Chiao Tung University
1001, Ta Hsueh Road, Hsinchu 300
TAIWAN

Dept. of Computer Science and Information Engineering, Tunghai University
181, Taichung Harbor Road, Sect. 3
Taichung 40704
TAIWAN

Abstract: The newest version of Globus Toolkit (GT4) adopts services-oriented architecture to provide grid environment based on Web Services. However, Globus Toolkit does not guarantee to reliably send and receive messages during messages passing between Web Services. Furthermore, the messages communication mechanism which Globus Toolkit Java Web Services Core provides is based on the changes of resource properties. In other words, it regards messages as resources properties but that is unreasonable for the perspectives of the programmers. In this research, we integrate GT4 Java WS core and our persistent version of JMS middleware (PFJM) to design reasonable programming styles and provide convenient and useful tools for Web Services development users by wrapping PFJM into PFJM Web Services (PFJM WS). Finally, we give a throughput test of messages communication respectively for GT4 Java WS core and PFJM WS. In the report, we can see that PFJM WS has a higher performance than GT4 Java WS core.

Key-Words: grid computing, Web service, Java Message Service (JMS), PFJM, Globus Toolkit

1 Introduction

1.1 Movitation

In recent years, grid technology [1, 2] is considered a good opportunity to integrate enterprise resources such as computing powers, storages, etc. Among them, the Globus Toolkit (GT) [3] is an open source software toolkit used for building grid systems and applications. It is developed by the Globus Alliance and many others all over the world. Java WS Core [4] is one of GT common runtime components and it provides APIs and tools for developing Grid services and offers a run-time environment capable of hosting them. The Java WS Core in GT4 implements the Web Services Resource Framework (WSRF) [5, 6, 7] and the Web Service Notification (WSN) [8] family of standards.

However, the communication mechanism between Web services does not guarantee reliable messaging. In other words, the notification consumer can not reliably receive the messages sent by notification producer since the unreliable network. Furthermore, communication between applications in enterprise environment is expected to be reliable because the messages lost may case a very serious consequence. It stands to reason that we extend Globus Toolkit WS Core to support reliable messaging.

Looking from another view, the architecture of Java Message Service (JMS) [9] can remedy the unreliability problem of Web Service Notification used by GT Java WS Core. It is more important that JMS provides the guarantee of reliable messaging. So in this research we integrate Persistent Fast Java Messaging (PFJM) [10, 11, 12, 13, 14, 15], a JMS-compliant product developed by our laboratory, into GT Java Web Core to provide useful web services for reliable messaging.

1.2 Research Objectives

In this research, we discuss the necessity of reliable messaging and the defective GT4 Java WS core. There are two objectives in this research including reliable messaging and reasonable programming styles.

1.2.1 Reliable messaging

Since the GT4 Java WS core lacks reliable messaging which is important for grid applications, we integrate a JMS compliant product, PFJM, into GT4 Java WS core to provide reliable messaging mechanism. The

features are described in the following:

1. Persistent messaging: Persistent messages are guaranteed to survive through JMS provider failure. If a message is set as persistent, before it is sent to the network, it must be stored in a persistent storage.
2. Durable subscription: Durable subscribers are guaranteed to receive persistent message published during their registration and de-registration, even they are not always active.

1.2.2 Reasonable programming styles

Since messages in GT4 Java WS core are always marked as a resource property, it is unreasonable from the programmer's point of view. For programmers, they expect to use Topic as the message destination to send and receive messages. Through our system, PFJM WS, clients can use JMS-like programming style to send and receive messages reliably.

2 System Architecture and Design

Globus Java WS Core is an implementation of WSRF, WSN, and other relevant Web Services family of standards. It provides an environment and tools to help develop plain Web Services and stateful Web Services. So the solution we use is to utilize these components Java WS core provides to wrap PFJM into Web Services. Whenever a client wants to communicate with others using reliable messaging, it can simply exploit the Web Services we provide to easily achieve its goal.

In Section 2.1, we will show the system architecture and introduce the basic operation and relationship of each component in the architecture. Then we will introduce the individual service portType we support in Section 2.2 and the mechanism of communication in Section 2.3.

2.1 High-Level Architecture

Figure 1 depicts the simplified system architecture. Whenever a client wants to be a message sender or a message receiver, it firstly must locate the persistent JMSFactoryService. A persistent service is a service which resides in the Web Services container when the container starts. After locating the JMSFactoryService, the client can use that to create a transient JMSPublisherService or JMSSubscriberService depending on what the client wants to be. Compared to a persistent service, a transient service is a service which can be created and destroyed dynamically. Then the client can use JMSPublisherService or JMSSubscriberService to create a PFJM instance, a publisher or a subscriber,

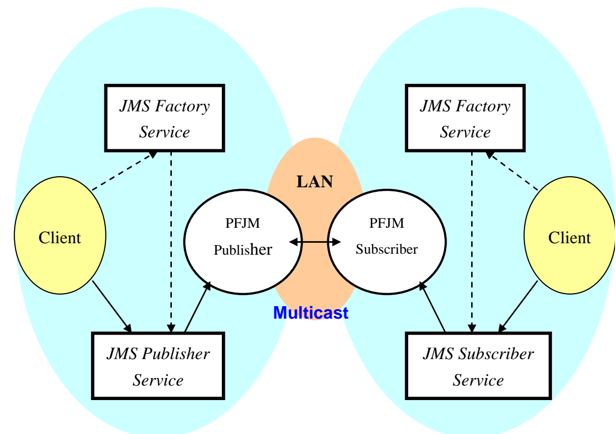

Fig. 1: Simplified System Architecture

and finally use the PFJM instance to do publish or subscribe operation via reliable messaging.

2.2 Service PortTypes

2.2.1 JMSFactory PortType

First of all, a Web Service can be addressed by a so-called EndpointReference. By passing the EndpointReference to a ServiceAddressingLocator, a client can get the service's portType implementation defined in the WSDL file. As well as JMSFactory portType which we provide using ***Factory*** design pattern, the client can use JMSFactoryAddressingLocator to locate the JMSFactory service.

The position of the JMSFactory is to create a JMSPublisher service or a JMSSubscriber service. Now let us take a look at how the JMSFactory works. For the purpose of managing created services, we provide two auxiliary managers, JMSPublisherManager and JMSSubscriberManager respectively in charge of JMSPublisher and JMSSubscriber services. When a client asks the JMSFactory to create an instance service, the JMSFactory passes the job to the manager. The service manager takes care of actually creating a new JMSPublisher or JMSSubscriber service and receives an object of type ResourceKey returned from a service resource home which implements ResourceHome interface provided by GT WS core. The ResourceKey is the identifier which we need to create the endpoint reference returning to the client. Figure 2 depicts the relationship of the service manager and the service resource home.

2.2.2 JMSPublisher PortType

JMSPublisherService created from JMSFactoryService is a transient service responsible

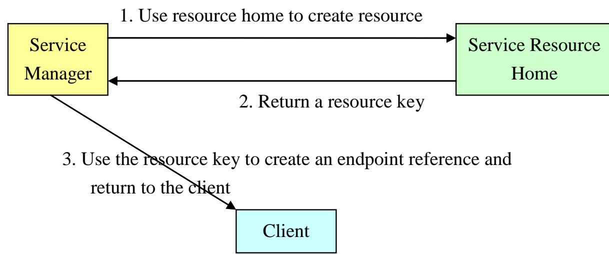

Fig. 2: Relation between Service Manager and Resource Home

to publish messages to a specific Topic. Whenever clients want to communicate to each other with reliable messages, the message sender can create a JMSPublisherService by passing specified topic name to JMSFactoryService and then utilize the created JMSPublisherService to publish messages. More precisely speaking, the JMSPublisherService is a PFJM instance which actually handles message sending.

The JMSPublisherService exposes only one operation, publish, to the public. And the publish operation has one parameter which is a String object indicating the sending messages.

2.2.3 JMSSubscriber PortType

As described in Section 2.2.2, JMSSubscriber is also created from JMSFactoryService and responsible to subscribe to the specific Topic. When a message receiver has created a JMSSubscriberService from JMSFactory Service by passing topic name, it then could use the JMSSubscriberService to do subscribe operation. In addition, the message receiver must implement a **NotifyCallback** interface which defines one function called **deliver** and pass itself to the subscribe operation of JMSSubscriberService. The deliver function is the callback function which the message receiver wants to be called back asynchronously when messages arrive.

The same as JMSPublisherService, JMSSubscriberService is a PFJM instance which actually handles message receiving. In addition to expose subscribe operation to the public, the JMSSubscriberService must also implement an interface, MessageListener, which JMS spec defines for asynchronously receiving messages.

2.3 Communication Mechanism

2.3.1 Publish Mechanism

Whenever a client wants to publish messages, in the beginning it must locate the JMSPublisherService by passing endpoint reference got from JMSFactoryService to JMSPublisherAddressingLocator. After locating the JMSPublisherService, it can call the publish function exposed by JMSPublisherService to send messages to a topic. Then JMSPublisherService activates the real publish operation provided by PFJM instance.

2.3.2 Subscribe Mechanism

Whenever a client wants to subscribe to a topic, it firstly locates the JMSSubscriber by passing endpoint reference got from JMSFactoryService to JMSSubscriberAddressingLocator. After locating the JMSSubscriberService, it passes itself implementing NotifyCallback interface as a parameter to the subscribe function exposed by JMSSubscriberService. Then the JMSSubscriberService activates the real durable subscribe operation provided by PFJM instance.

2.3.3 Delivery Mechanism

As long as messages are sent to Topic, the PFJM core will invoke the **onMessage** method which JMSSubscriberService implements. Then JMSSubscriberService will invoke the callback function **deliver** implemented by the client. Finally the client will receive the messages.

2.3.4 Recovery Mechanism

Now let's take a look at how the recovery mechanism works when the receiver crashes or the network fails and later the receiver revives again. In JMS lingo, when a client wants to receive reliable messages, it must register a durable subscription with a unique identity also known as subscription name that is retained by the JMS provider. Subsequent subscriber objects with the same identity resume the subscription in the state in which it was left by the previous subscriber. If a durable subscription has no active subscriber, the JMS provider retains the subscription's messages until they are received by the subscription or until they expire.

In PFJM WS, the receiver also passes a subscription name in addition to a topic name to the JMSFactoryService to register itself. After getting the EPR of JMSSubscriberService returning from JMSFactoryService, the receiver can use the EPR to locate JMSSubscriberService and then subscribe to the specific topic with the subscription name. Then the JMS provider will be in charge of sending messages reliably to the receiver.

If the network fails, the JMS provider will store the messages published to the topic. Until the receiver revives with the same topic and subscription name, the subscription will be reactivated, and the JMS provider will deliver the messages that are published while the subscriber is inactive.

3 Programming Styles

In this section, we will discuss GT4 Java WS core and the integrated system, PFJM WS, which we provide to support reliable messaging. We will first give a simple but typical grid service application. Then we will give scenarios respectively for PFJM WS in Section 3.2 and for GT4 Java WS core in Section 3.3. In Section 3.4 and Section 3.5, we will discuss the detail implementation about using messaging mechanism respectively for PFJM WS and for GT4 Java WS core. Finally, we will give a discussion about the comparison of programming styles between PFJM WS and GT4 Java WS core.

3.1 Simple Grid Service Application

This section uses a simple grid computing example called divide-and-conquer [16] as demonstration. As shown in Fig. 3, when the arithmetic grid service responsible for the four fundamental operations of arithmetic receives a task, it then **divides** the task into two subtasks, a multiplication subtask and a division subtask. Afterward, the arithmetic grid service will locate two other grid services respectively responsible for multiplication and division and **dispatch** the divided subtasks to those. After the multiplication and division grid service finish their operation, they will pass the results to the arithmetic grid service. Finally the arithmetic grid service will **merge** the results and return to the client.

3.2 Conceptual Scenario in PFJM WS

Here we will use the example in Section 3.1 to describe how to implement the scenario from PFJM WS.

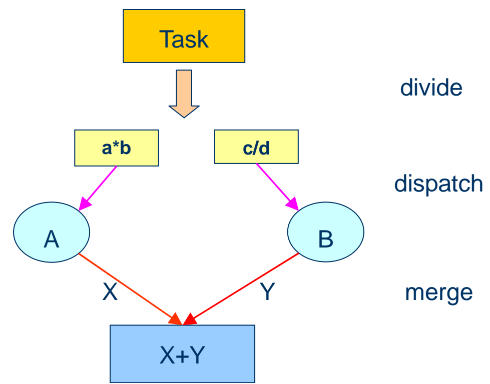

Fig. 3: A Simple Grid Service Application

As Fig. 4 shows, after a client submits a task to the Arithmetic Service, the service internally divides the task into two subtasks and locates two other grid services responsible for multiplication and division services. Then the Arithmetic Service being a publisher dispatches the two subtasks to the multiplication service and division service being subscribers.

Once the multiplication and division services finish their work, they will become publishers and return the results to the Arithmetic Service being a subscriber. And the Arithmetic Service will merge the results and finally return to the client.

3.3 Conceptual Scenario in GT4 Java WS Core

In GT4 Java WS Core, as shown in Fig. 5, after a client submits a task to the Arithmetic Service, the service internally divides the task into two subtasks and declares two resources, X and Y, standing for the results of the two subtasks. The Arithmetic Service also locates two other grid services responsible for multiplication and division services and dispatches the two subtasks to them.

Once the multiplication and division services finish their work, they will utilize the operation provided by the Arithmetic Service to update the properties of the resources. While the Arithmetic Service receives the notification about changing of resources properties from Java WS core, it finally merges the results and returns to the client.

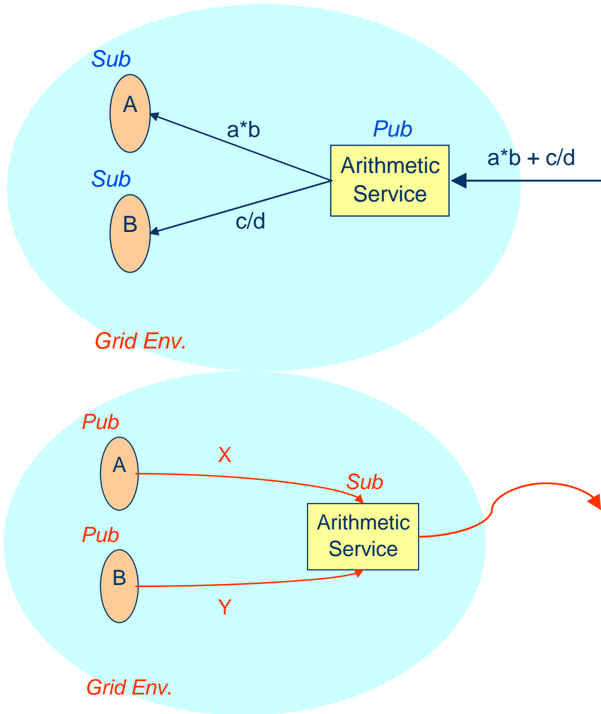

Fig. 4: Conceptual Scenario in PFJM WS

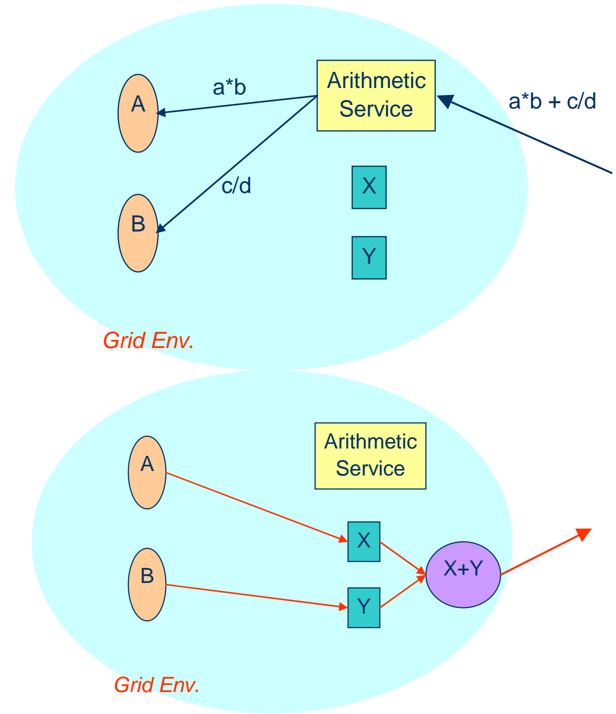

Fig. 5: Conceptual Scenario in GT4

3.4 Discussion

From the two sets of scenarios mentioned above, we know that if a client wants to use messaging mechanism based on GT4 Java WS core, it can achieve this goal by means of modifying the resource properties and receiving the notification. This is not intuitive from the point of view of a programmer since messages are received passively. Besides, the API of GT4 Java WS core is complex for a programmer and a lot of processes must be done by the client himself. On the contrary, if a client uses PFJM WS to do message communication, it is not only reasonable for the point of the view of a programmer, but the programmer also can use the API provided by PFJM WS more easily than GT4 Java WS Core.

4 Experimental Results

We have conducted some experiments to investigate the performance difference between original GT4 Java WS Core and the integrated system, PFJM WS. During the experiment, we use the FX-05EA 5 Ports 10/100 Mb switching hub to form a local area network with one notebook and three PCs connecting to a 100 Mbps Fast Ethernet. Each PC runs Windows XP with JDK 1.4.2 or 1.5.0.

The experiments are divided into two categories. The first is one-to-one communication using notebook as a message sender and PC1 as a message receiver. The second is one-to-many communication, using notebook as a message sender and PC1-PC3 as message receivers.

We use the default setting for PFJM and measure throughput of different data sizes for GT4 Java WS core and PFJM WS. For each data sizes, we perform one hundred times message transmission and calculate the average throughput in bytes per second.

The results reveal that the throughput of PFJM WS is better than that of GT4 Java WS core in both cases. Consequently, if a user wants to use reliable messaging in GT4 Java WS environment, our PFJM WS providing a convenient manner and having nice efficiency is a good choice.

5 Conclusion

In this paper we have presented our approach to wrapping the internal communication framework in GT4 Java WS core with a persistent JMS middleware developed in our laboratory (i.e., PFJM) so as to provide more reliable messaging and reasonable programming styles than those in GT4 Java WS Core. We have also demonstrated that using PFJM WS to send and receive messages is more efficient.

Acknowledgments
This work was partially supported by National Science Council grant NSC95-2752-E-009-PAE: advanced technologies and applications for next generation information networks, grant NSC94-2213-E-009-026: a research on next generation massive multiplayer virtual environment platform, and grant NSC94-2520-S-009-004.

References:

[1] I. Foster, C. Kesselman, J. Nick, and 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.

[2] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, and J. Treadwell, *The Open Grid Services Architecture, Version 1.0*, J. Von Reich. Informational Document, Global Grid Forum (GGF), January 29, 2005.

[3] Globus Toolkit, http://www.globus.org/toolkit/

[4] GT 4.0 Java WS Core, http://www.globus.org/toolkit/docs/4.0/common/javawscore/

[5] Web Services Resource Framework, http://www.globus.org/wsrf/

[6] K. Czajkowski, DF Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S.Tuecke, and W. Vambenepe, *The WS-Resource Framework*, Version 1.0, March 2004. http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[7] M. Humphrey, G. Wasson, K. Jackson, J. Boverhof, M. Rodriguez, Joe Bester, J. Gawor, S. Lang, I. Foster, S. Meder, S. Pickles, and M. McKeown, "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations," *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, Research Triangle Park, NC, July 2005, pp. 3-13.

[8] Steve Graham and Bryan Murray, *Web Services Base Notification 1.2 (WS-BaseNotification)*, OASIS Working Draft 03, June 2004. http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf

[9] Sun Microsystems, *Java Message Service*, Version 1.1, April 2002.

[10] Chuan-Pao Hung, Hsin-Ta Chiao, Yue-Shan Chang, Tsun-Yu Hsiao, Tzu-Han Kao, and Shyan-Ming Yuan , "FJM: A Fast Java Message Delivery Mechanism based on IP-Multicast," *Proceedings of the 3rd International Conference on Communications in Computing (CIC 2002)*, Las Vegas, June 2002.

[11] Tsun-Yu Hsiao, Nei-Chiun Perng, Winston Lo, Yue-Shan Chang, and Shyan-Ming Yuan, "A New Development Environment for an Event-based Distributed System," *Computer Standards & Interfaces*, vol. 25, no. 4, August 2003, pp. 345-355.

[12] Tsun-Yu Hsiao, Ming-chun Cheng, Hsin-Ta Chiao, and Shyan-Ming Yuan, "FJM: A High Performance Java Message Library," *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER '03)*, Hong Kong, December 2003, pp. 460-463.

[13] Yu-Fang Huang, Tsun-Yu Hsiao, and Shyan-Ming Yuan. "A Java Message Service with Persistent Message," *Proceedings of 2003 Symposium on Digital Life and Internet Technologies*, Tainan, Taiwan, September 2003.

[14] Ruey-Shyang Wu and Shyan-Ming Yuan, "An Adaptive Architecture for Secure Message Oriented Middleware," *WSEAS Transactions on Information Science and Applications*, vol. 3, no. 7, July 2006, pp. 1239-1246.

[15] Ruey-Shyang Wu, Kuo-Jung Su, and Shyan-Ming Yuan, "FJM2: A Decentralized JMS System," to be presented in 2nd International Conference on Trends in Enterprise Application Architecture (TEAA2006), Berlin, Germany, November 29-December 1, 2006.

[16] I-Chen Wu and H. T. Kung, "Communication complexity for parallel divide-and-conquer," *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, September 1991, p.151-162.