Distributed Workflow Management System based on Publish-Subscribe Notification for Web Services

Hisashi Shimosaka[1], Tomoyuki Hiroyasu[2], and Mitsunori Miki[2]

[1] Graduate School of Engineering, Doshisha University,
1-3 Tatara Miyakodani Kyotanabe, Kyoto, Japan,
hisashi@mikilab.doshisha.ac.jp

[2] Department of Engineering, Doshisha University
tomo@is.doshisha.ac.jp,mmiki@mail.doshisha.ac.jp

Abstract. In recent years, Grid technologies have been standardized based on Web service specifications. Of these specifications, the WS-Resources Framework and WS-Notification have attracted a great deal of attention. This paper focuses on scientific applications integration. We propose and implement a new distributed workflow management system called the "Application Igniting System." This system is based on the publish-subscribe notification defined by the WS-Notification specification and realizes a flexible and loosely coupled workflow control by providing some utility services, which handle message exchange. By applying to a typical bioinformatics workflow, we concluded that the overhead time related to message exchange is very short.

1 Introduction

With the development of Grid technologies in recent years, many scientific applications are being implemented on the wide area network. In addition, various Grid workflow management systems have been developed to integrate these applications[1]. In the field of bioinformatics, as in other multidisciplinary fields, these workflow management systems are applied to solve collaboration-hungry problems[2]. On the other hand, recent Grid technologies have been standardized based on Web service specifications[3]. Of these specifications, the WS-Resources Framework (WSRF)[4] and WS-Notification (WSN)[5] have attracted a great deal of attention because these specifications enable the construction of Grid-enabled Web services. The WSRF specification enables the definition of a stateful resource, and the WSN specification also realizes message exchanges driven by a state transition. By applying these specifications, more Grid-oriented workflow management can be realized. Based on this background, we propose a new distributed workflow management system, which is called "the Application Igniting System." This system adopts a decentralized architecture based on message exchange defined by the WSN specification. The architecture enables the system functions to easily be enhanced by adding a simple service that handles message

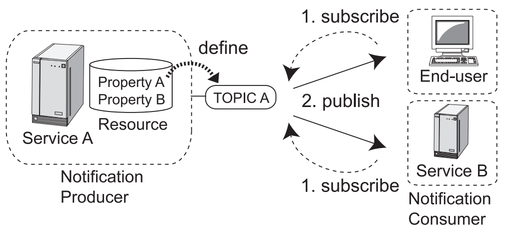

Fig. 1. Overview of the WS-Notification framework

exchange. Then, a more flexible and loosely-coupled workflow control can be realized. In the simulation example, we apply the Application Igniting System to a typical bioinformatics workflow and evaluate its basic performance.

2 WS-Resource Framework and WS-Notification

WSRF is a specification to introduce a stateful resource into a Web service. The stateful resource consists of a set of resource properties. The WSRF specification enables the Web service to express a state transition during service execution. For example, in an application service, it can express the state transition of application execution by introducing the resource property corresponding to the application execution status, such as "Pending," "Active," "Finished," or "Failed."

WSN is a sophisticated specification of the topic-based publish-subscribe notification for Web services. Fig. 1 shows an overview of the simplest architecture in the WSN specification. As shown in Fig. 1, the NotificationProducer that publishes notification messages defines a set of notification topics associated with each resource property participating in the value change notification pattern in advance. The NotificationConsumers that receive the notification messages subscribe to the NotificationProducer with the item of interest. Then, whenever the resource property is changed, the NotificationProducer publishes the notification message that includes the details of the state information. Hence, each NotificationConsumer can execute the appropriate method according to the notification message context.

3 Application Igniting System

In this paper, we propose a new distributed workflow management system based on the publish-subscribe notification realized by the WSN specification. The proposed system is implemented using the Globus Toolkit (version 4.0.1)[6] and is called the Application Igniting System.

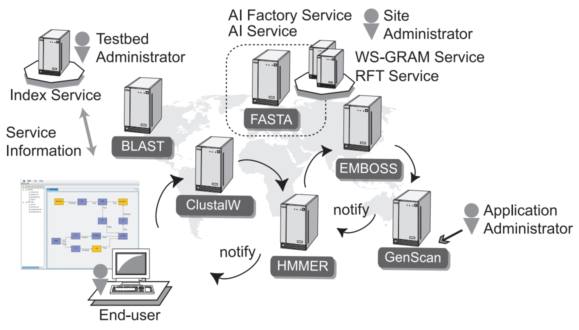

Fig. 2. Overview of the Application Igniting System

3.1 Overview of the Application Igniting System

Fig. 2 shows an overview of the Application Igniting System. In this system, the Grid testbed administrator first prepares the Index Service provided by the Globus Toolkit. Then, each site administrator introduces the Application Igniting Factory Service (AI Factory Service) and the Application Igniting Service (AI Service). We provide a package to introduce both services. Each site administrator also prepares the WS-GRAM (Grid Resource Allocation and Management) Service and the RFT (Reliable File Transfer) Service provided by the Globus Toolkit. The AI Service uses the WS-GRAM Service for job submission and uses the RFT Service for file transfer. The application administrator registers their own application information with one of the AI Factory Services using the Resource Specification Language (RSL), which is a language for requesting job submission to the WS-GRAM Service. All of the registered application information is gathered in the Index Service and is provided to end-users. The end-user first selects some applications and designs a workflow to solve a problem. The designed workflow is realized by notification messages among the AI Services.

3.2 Resource Creation and Port Types

The end-user obtains the application information from the Index Service and selects some applications of interest by requesting resource creations from the AI Factory Services, which have information regarding each selected application. Each AI Factory Service makes a resource-specific working directory, creates the resource based on the registered application information, and returns the pairing information of the AI Service introduced by the same site administrator and the created resource to the end-user. Each of the created resources includes the resource property corresponding to the application execution status. In addition, the AI Service defines the Application Invocation Topic (AIT) associated with this resource property. On the other hand, each of the utility services described later defines the Conditional Branch Topic (CBT) associated with the different resource property corresponding to the service execution status.

The typical port types, which are used for workflow design and execution, that are provided by each AI Service with a resource are as follows.

- setSubscription: Subscribe to a topic CBT specified in the input argument. The input argument is the paring information of a utility service that defines the topic and a resource that includes the resource property associated with the topic.
- createTransferResource: Create a resource for receiving an input file from an end-user or for sending an output file to an end-user or another AI Service. This is realized by requesting resource creation to the RFT Service prepared by the same site administrator. The input arguments are the source and destination file information. The return value is the pairing information of the RFT Service and the created resource. The caller accomplishes the file transfer using the returned information.
- addTransfer: Add source file information specified in the input argument to the list. The specified source file is an output file generated by another AI Service. The listed output files are acquired as input files before invoking the application.

3.3 Application Invocation Procedure

Each AI Service with a resource invokes the application using the stored application information in the resource. The application invocation is driven by the receipt of a notification message. Fig. 3 shows the application invocation procedure in the AI Service with a resource. The end-user dictates the subscription from the AI Service to the NotificationProducer using the setSubscription port type in advance. In addition, the source file information is specified using the addTransfer port type. These specified source files are the output files generated by other AI Services with a resource. Based on these end-user requests, the AI Service begins the application invocation by receiving the notification message with the value “True” from the NotificationProducer. The application invocation consists of the following procedure. The AI Service first acquires each source file specified by the end-user as the input file. This is realized using the createTransferResource port type provided by another AI Service with a resource, which has the source file. After acquiring all source files, the AI Service requests job submission to the WS-GRAM Service using the RSL information stored in the resource. When job submission is completed successfully, the value of the resource property corresponding to the application execution status is changed to “Finished.” When job submission is not completed successfully, the value is changed to “Failed.” These updates of the resource property cause another notification message driven by the state transition.

3.4 Workflow Design and Execution

The Application Igniting System provides some utility services, which handle message exchange and support workflow design and execution.

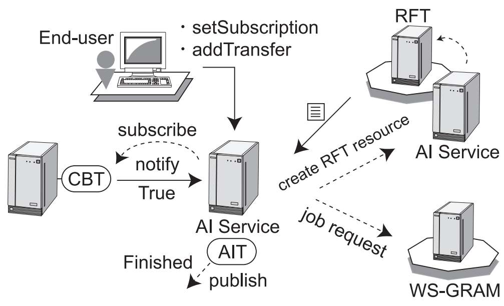

Fig. 3. Application invocation procedure

Sequential Workflow and Conditional Branch The IF Service is the most important of the utility services to support the workflow design and execution. The IF Service has two port types: one that can subscribe to the topic AIT specified in the input argument, while the other can subscribe to the specified topic CBT, similarly to one of the port types provided by the AI Service. The input argument of these port types is the pairing information of a service that defines the topic and a resource that includes the resource property associated with the topic. Moreover, the condition value can be included in the input argument. The IF Service also handles the resource that includes the resource property corresponding to the service execution status and defines the topic CBT associated with the resource property. Then, if the received notification message includes the same value as the condition value, the resource property corresponding to the service execution status is changed to “True.” If the notification message does not have the same value, the value is changed to “False.” These updates of the resource property cause another notification message driven by the state transition.

An example of sequential workflow using the IF Service is shown in Fig. 4. The end-user creates three resources in advance and dictates that the IF Service with a resource subscribes to the AI Service with the resource that includes the information of application A. The condition value is set to “Finished.” In addition, the end-user dictates that the AI Service with the resource that includes the information of application B subscribes to the IF Service. Then, when the execution of application A is completed successfully, the notification message with the value “Finished” is delivered from the AI Service with the resource that includes the information of application A to the IF Service. As this notification message has the same value as the condition value of the IF Service, another notification message with the value “True” is delivered continuously from the IF Service to the AI Service with the resource that includes the information of application B. Then, the AI Service with the resource that includes the information of application B begins the application invocation. As shown above, the IF Service can intercept the notification message and translate its value. Thus, the IF Service assumes an important role in the Application Igniting System.

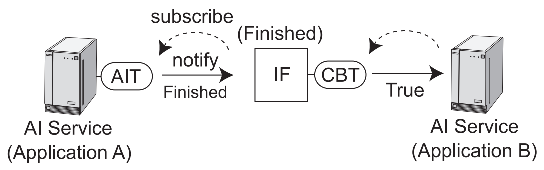

Fig. 4. Sequential workflow

Parallel and Synchronous Workflow The notification message driven by a state transition is delivered to all of the NotificationConsumers, which subscribe to the same topic, defined by the same service and associated with the same resource property. Therefore, parallel workflow execution can be realized easily by delivering the notification message with the value "True" driven by a state transition to more than one AI service with a resource simultaneously. In this case, a function that can synchronize the parallel workflow execution is required. Then, the Application Igniting System also provides the AND Service. The AND Service has the port type to subscribe to the topic CBT specified in the input argument and waits the notification message with the value "True." In addition, the AND Service has the characteristic function, which can subscribe to two NotificationProducers. The AND Service also handles the resource including the resource property corresponding to the service execution status and defines the topic CBT associated with the resource property. Then, upon receiving notification messages from both NotificationProducers, the value of the resource property corresponding to the service execution status is changed to "True." This update of the resource property causes another notification message driven by the state transition. An example of workflow using the AND Service is shown in Fig. 5. In this example, the end-user creates six resources in advance and dictates the subscriptions of the inverse directions of the arrows in Fig. 5. When the IF* Service with resource A updates the value of the resource property to "True," both applications A and B are invoked by the AI Services with a resource in parallel. In addition, the AND service synchronizes the normal terminations of execution of both application by waiting to receive the notification messages with the value "True" from two IF Services with a resource. After synchronizing, the AND Service updates the value of the resource property to "True."

Loop Workflow To support the loop workflow, the Application Igniting System provides the LOOP Service and the OR Service. The details of these services are omitted in this paper due to space limitations.

Input and Output File Transfer When the end-user prepares the input files of an application, the end-user first requests the resource creation and obtains the pairing information of the RFT Service and the resource to send the input file using the createTransferResource port type provided by the AI Service, which

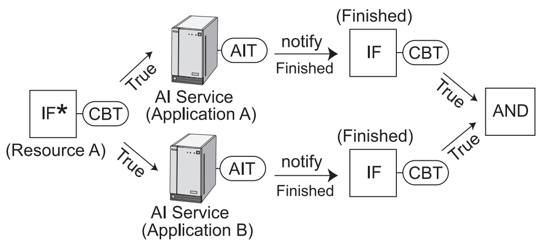

Fig. 5. Parallel and synchronous workflow

has the resource that includes the information of the application. Then, file transfer is accomplished using the pairing information. The procedure is also the same when the end-user receives an output file of an application. On the other hand, when an output file of application A is used as an input file of application B, the end-user specifies the mapping information between the input and output files to the AI Service with the resource that includes the information of application B using the addTransfer port type.

File Format Translation To support file format translation, the Application Igniting System provides the Editor Service, which has the same port types and functions as the AI Service except the application management mechanism. The Editor Service invokes the application prepared by the end-user using the RSL file, also prepared by the end-user. The prepared application is transferred using the createTransferResource port type of the Editor Service as in the case of input file transfer. In addition, to register the prepared RSL file, the Editor Service provides the setEditor port type. The prepared RSL file is specified in the input argument of the port type. Fig. 6 shows an example of the file format translation procedure. In this example, the output file of application A is translated and is used as the input file of application B. The end-user creates the application for file format translation using the application information gathered on the Index Service in advance. In addition, the end-user prepares the RSL file for invoking the application on the Editor Service. These are transferred to the Editor Service using the createTransferResource and setEditor port types. Then, the end-user specified the mapping information for file transfers such that the output file of application A is used as the input file of the Editor Service and the output file of the Editor Service is used as the input file of application B. These operations are realized using the addTransfer port types of the Editor Service and the AI Service, which has the resource that includes the information of application B. Finally, the end-user dictates the subscriptions to invoke the transferred application on the Editor Service before invoking the application B. Thus, file format translation is realized.

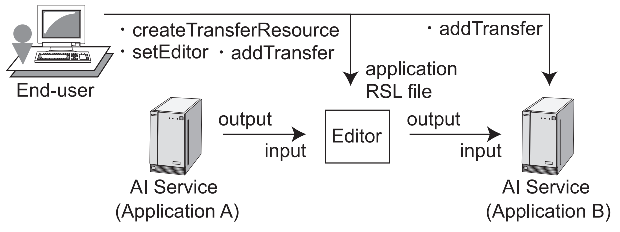

Fig. 6. File format translation

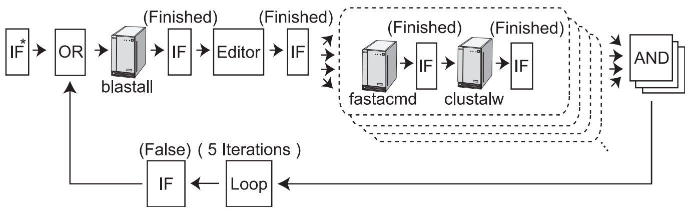

Fig. 7. Notification message flow

4 Bioinformatics Workflow

Here, we applied the Application Igniting System to a typical bioinformatics workflow using the BLAST[7] and ClustalW[8] packages and evaluate its basic performance.

4.1 Experimental Environment and Workflow

In this simulation example, a Grid environment consisting of two PC clusters was used. The blastall and fastacmd commands were deployed on either cluster and the clustalw command was deployed on the other. Each command was invoked via the job scheduler and executed on any slave node in each cluster. In addition, the end-user designed the notification message flow shown in Fig. 7 and dictated the file transfers and file format translation shown in Fig. 8. In this workflow, to evaluate the basic performance in parallel workflow execution, some pairs of fastacmd and clustalw commands were prepared and these were executed in parallel. We evaluated the basic performance with 1, 2, 4, and 8 pairs.

4.2 Workflow Execution Results

Table 1 shows the breakdown of the elapsed time when the number of pairs was 1. As shown in TABLE 1, in pre- and post-processing, the file transfers and resource creation took a long time. In file transfers, the average transfer time of a file or an application was 4.4 s. The long resource creation time was due to the

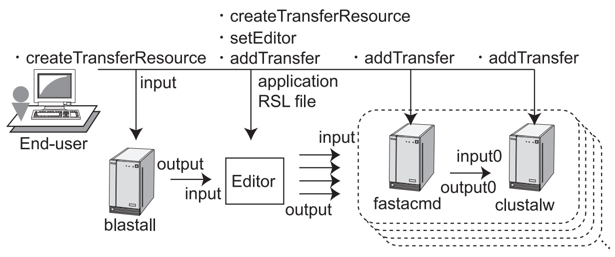

Fig. 8. Input and output file transfer and file format translation

Table 1. Breakdown of the elapsed time

Item	Detail of item	Time (s)
Pre processing	12 resource creations	28.5
	12 subscription dictates	4.9
	5 input file transfers (each 0.3KB)	25.7
	1 application transfer (0.1KB)	6.0
	1 RSL file registration	0.1
	3 file exchage dictates	0.4
Workflow execution	Command execution (blastall)	1047.7
	Output file acquisition (Editor,341KB)	37.5
	Application execution (Editor, translation)	19.1
	Output file acquisition (fastacmd,3KB)	20.9
	Command execution (fastacmd)	63.2
	Output file acquisition (clustalw,107KB)	22.4
	Command execution (clustalw)	678.4
	Message exchanges and otherwise	12.8
Post processing	5 output file transfers (each 146KB)	17.2
	12 resource destructions	4.9

security mechanism of the Globus Toolkit. On the other hand, as shown in the breakdown of the workflow execution time, the total overhead time was 112.6 s, which was short compared with the application execution time. However, the average acquisition time of an output file was 5.4 s. The other overhead time was 6.4 s on average. As the average time of file format translation was 3.8 s, the average overhead time related to message exchange was only 2.6 s, which is very short compared with the total overhead time.

The increments in the elapsed time with 2, 4, and 8 pairs compared with the case with a single pair are shown in Fig. 9. As shown in Fig. 9, as the numbers increase, the increments in workflow execution make up a substantial portion of the total increments. However, as the execution time of a pair of fastacmd and clustalw commands was 784.9 s (TABLE 1), these increments are very short.

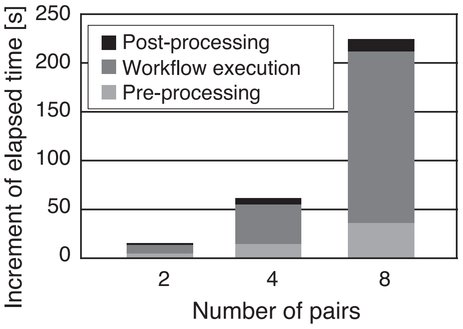

Fig. 9. Increment of elapsed time

5 Conclusions

In recent years, Grid technologies have been standardized based on Web service technologies. In this paper, we proposed and implemented a new distributed workflow management system called the "Application Igniting System." This system is based on the message exchange defined by the WSN specification and realizes a flexible and loosely coupled workflow control by providing utility services that handle message exchange. By applying to a typical bioinformatics workflow, we concluded that the overhead time related to message exchange is very short.

References

1. Yu,J. et.al., A Taxonomy of Scientific Workflow Systems for Grid computing, *Special Issue on Scientific Workflows*, SIGMOD Record, ACM Press, 2005.
2. Oinn,T. et.al., Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics*, 20(17):3045-3054, Oxford University Press, London, UK, 2004.
3. Foster,I. et.al., The Open Grid Services Architecture, Version 1.0, Global Grid Forum OGSA-WG, GFD-I.030, 2005.
4. Czajkowski,K. et.al., The WS-Resource Framework, Version 1.0, 2004, [Online]. Available: `http://www.oasis-open.org/committees/download.php/6796/ws-wsrf.pdf`
5. Graham,S. et.al., Publish-Subscribe Notification for Web services, Version 1.0, 2004, [Online]. Available: `http://www.oasis-open.org/committees/download.php/6661/WSNpubsub-1-0.pdf`
6. Foster,I. et.al., Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, Vol.11, No.2, pp.115-128, 1997.
7. Altschul S.F. et.al., Basic local alignment search tool, *Journal of Molecular Biology*, 1990.
8. Thompson,J.D. et.al., CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Reseach*, 1994.