A Novel Architecture for Realizing Grid Workflow using Tuple Spaces

Jia Yu and Rajkumar Buyya
GRIDS Laboratory and NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{jiayu, raj}@cs.mu.oz.au

Abstract

Grid workflow can be defined as the composition of grid application services which execute on heterogeneous and distributed resources in a well-defined order to accomplish a specific goal. Uncertainties within grid environments pose new challenges for grid workflow management systems such as lacking central control and undedicated resource sharing. In this paper, we provide a workflow enactment engine together with an XML-based workflow language (xWFL). The workflow engine supports a just in-time scheduling system, thus allowing the resource allocation decision to be made at the time of task execution and hence adapt to changing grid environments. We also show that an event-driven scheduling architecture using tuple spaces provides a highly flexible approach for executing large scale complex grid workflows.

1. Introduction

Grid computing [1] facilitates the sharing and aggregation of heterogeneous and distributed resources, such as computing resources, data sources, instruments and application services. With the advent of grid technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed grid resources. Means of composition and executing distributed applications to form complex workflow are needed. Some efforts in the research of business workflow and Web services orchestration can be reused in grid workflow systems. However, there are new challenges needing to be addressed.

In a typical Grid environment, organizational structures known as virtual organizations (VOs) [4] are dynamic federations of individuals and organizations aimed at solving a problem of common interests via collaboration. Resources are shared between users in the VOs, and added and removed all the time. Therefore, computational and networking capabilities can vary significantly over time.

Another difficulty for workflow management in Grids is that there is no central ownership and control. The control over grid resources is dispersed across multiple administrative domains. The grid resources are not under the exclusive and complete control of the owner of the workflow systems. The local policy is subject to change without informing the users of workflow systems.

The execution of a grid workflow faces many uncertain factors such as unavailability, incomplete information and local policy changes. A full-ahead-plan [24] is not always suitable, since it requires the specification of the exact location of resources assigned to every task, as well as the physical files of the data in advance before the workflow execution. The workflow execution failure could be caused by the status change of the specified resources during the workflow execution. Therefore, the grid workflow enactment engine must be able to adapt to the changing conditions and be able to quickly re-allocate resources.

Moreover, the processing of grid workflow could be quite complex. The tasks of grid workflows are expected to be executed on heterogeneous resources which are geographically distributed. Many different resources may be involved in one workflow execution. For example, in a scientific experiment, one needs to acquire data from an instrument, and run on resources owned by other organizations in sequence or parallel to analyze datasets. Therefore, the processing of resource discovery and selection could be quite complicated. In addition, a large number of tasks may be required to execute in parallel and the location of intermediate data may be known only at run-time.

In this paper, we propose a workflow enactment engine (WFEE) with a just in-time scheduling system using tuple spaces. It allows the decision of resources allocation to be made dynamically at the time of the execution of tasks in the workflow. Compared with the approach of partial workflow submission [5], we believe a decentralized event-driven scheduling architecture provides a more flexible and loosely-coupled control. It also allows tasks to be scheduled as data becomes available rather than waiting for the completion of parent tasks.

The remainder of this paper is organized as follows: In Section 2, we present related work. The overview of WFEE is briefly introduced in Section 3. The workflow description language for the engine is shown in Section

4. The mapping of applications approach for resource discovery in VOs is discussed in Section 5. Detailed design of the scheduling system is described in Section 6. Section 7 shows the implementation of WFEE. Experiments and result discussion are presented in Section 8. We conclude in Section 9 with discussion of future work.

2. Related Work

Our workflow engine is an independent workflow execution system and takes advantage of various middleware services such as security, grid resource access, file movement and replica management services provided by the Globus middleware [6], and parametric application execution provided by the Gridbus Broker [7] and VO directory service provided by the Grid Market Directory (GMD) [8].

Many efforts toward grid workflow management have been made. DAGMan [9] was developed to schedule jobs to Condor system in an order represented by a DAG and to process them. With the integration of Chimera [10], Pegasus [5] map and execute complex workflow based on full-ahead-planning. In Pegasus, a workflow can be generated from metadata description of the desired data product using AI-based planning technologies. The Taverna project [11] has developed a tool for the composition and enactment of bioinformatics workflow for the life science community. The tool provides a graphical user interface for the composition of workflows. Other workflow projects in the grid context include ScyFlow [12], GridFlow[2], Unicore[13], ICENI[14], GridAnt[15] and Triana [29].

Compared with the work listed above, we provide a decentralized, just in-time, meta-scheduling system which enables workflow resources to be discovered and reallocated at run-time. Another feature of our WFEE is to schedule tasks immediately after the availability of required input data rather than waiting for the complete execution of all tasks in the parents' hierarchy.

There are a number of projects[16][17] focused on developing a grid workflow language. In addition, many efforts on the composition of Web services [18] can also be complementary to the development of grid workflow management systems. The main aim of our workflow language is to support parameterization [19], which is important to scientific applications.

3. Overview of Workflow Enactment Engine

The architecture of our workflow enactment engine (WFEE) and its relationship with other components in the Grid infrastructure are shown in Figure 1. Workflow applications such as scientific application portals, submit task definitions along with their dependencies in the form of the workflow language as well as QoS requirements to WFEE. WFEE schedules tasks through grid middleware on the grid resources.

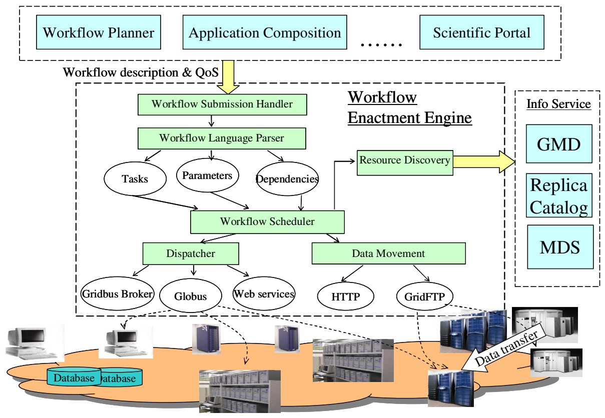

Figure1. The Architecture of WFEE.

The key components of WFEE are: workflow submission, workflow language parser, resource discovery, dispatcher, data movement and workflow scheduler.

- Workflow submission accepts workflow enactment requests from planner level applications.
- Workflow language parser converts workflow description from XML format into Java objects, *Task*, *Parameter* and *DataConstraint* (workflow dependency) which can be accessed by workflow scheduler.
- Resource discovery is intended to query grid information services such as Globus MDS [6], GMD and replica catalogs, to locate suitable resources for the tasks.
- There are several types of middleware used for Grids such as Globus 2 GRAM [6] , Web services [20] and OGSI[21]. Grid resources may be connected by different middleware. WFEE had been designed to support different middleware by creating dispatchers for each middleware to support its interaction with resources.
- Data movement system enables data transfer between grid nodes by using HTTP and GridFTP [22] protocols.
- Workflow scheduler is the central component in WFEE. It interacts with *resource discovery* to find suitable grid resources at run time; it locates a task on resources by using the dispatcher component; it controls input data transference between task execution nodes through *data movement*.

4. Workflow Language and Structure

In order to allow users to describe tasks and their dependencies, we defined a simple and flexible XML-based workflow language (xWFL). The workflow language provides the means to build new applications by linking standalone applications.

Our workflow description expressed in XML format, and its structure is shown in Figure 2. It consists of three parts, namely parameter definitions, task definitions and data link definitions.

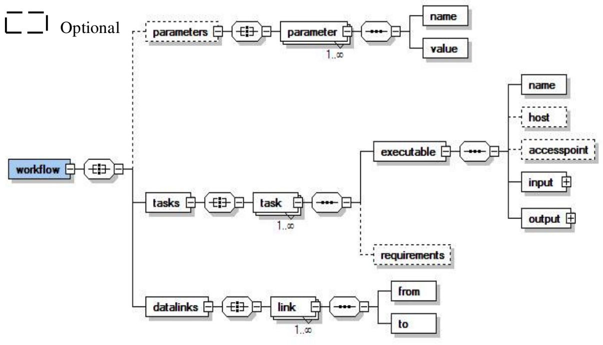

Figure 2. The Structure of Workflow Language.

4.1 Parameters

Supporting parameterization in the workflow language is very important for scientific applications. It enables scientists to do experiments on different parameters easily by changing the value of parameters without being concerned about the detailed workflow description. Multiple parameters types such as single, range, select, random, file and multi-files need to be supported. An example for a single parameter type and a range parameter type is given below:

```
<parameters>
        <para type= "single">
                <name>X</name>
                <value type=integer>10</value>
        </para>
        <para type= "range">
                <name>Y</name>
                <min>1</min>
                <max>20</max>
                <step>2</step>
        </para>
</parameters>
```

4.2 Tasks

Basically, <tasks> is a set of tasks that are to be executed. Our workflow language supports both abstract workflow and concrete workflow. As shown in the Figure 2, host and accesspoint are optional. That is the user or higher workflow planner, can either specify the location of a particular resource providing required application services or leave it to the engine to identify their providers dynamically at run-time. In the below example, task A executes dock.exe program on host *services.gridbus.com* in the directory */services* and executable *dock* has two input I/O ports: port0 (a file) and port1 (a parameter value). The example shows task A only has one output named port2.

```
<tasks>
  <task name= "A">
     <executable>
        <name>dock</name>
        <host>services.gridbus.com</host>
        <accesspoint type= "GlobusGram">/dock.exe</accesspoint>
        <input>
           <port0 type= "file" url= "http://www.gridbus.org/
                               datacenter/dock.in">dock.in</port0>
           <port1 type= "msg">$time_base_value</port1>
        </input>
        <output>
           <port2 type= "file">dock.out</port2>
        </output>
     </executable>
        …..
  </task>
</tasks>
```

4.3 Data Links

Data link is used to specify the data flow of the tasks. Below example is the data flow description of Figure 3. The inputs of task B and task C rely on the output of A. The task A's output needs to be transferred to the node on which task B and task C are executed. Input could be a file, parameter value or data stream.

```
<workflow>
   <tasks>
      <task name= "A">
          …..
      </task>
      <task name= "B">
          …...
      </task>
    <task name= "C">
     ……
    </task>
    <task name= "D">
     ……
    </task>
  </tasks>
  <links>
    <link>
     <from>A:port2</from>
     <to>B:port0</to>
    </link>
    <link>
     <from>A:port2</from>
     <to>C:port0</to>
    </link>
    <link>
     <from>B:port1</from>
     <to>D:port0</to>
    </link>
    <link>
```

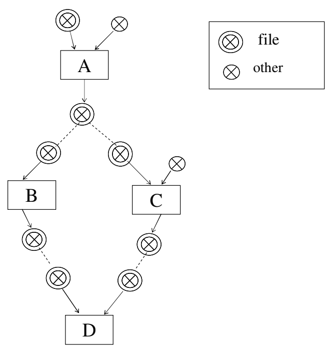

Figure 3. Flow Diagram of Task A, B, C and D.

```
    <from>C:port2</from>
    <to>D:port1</to>
  </link>
 </links>
</workflow>
```

5. Mapping of Applications

In our previous work [7], we have set up a directory service called GMD which allows resource providers to register themselves as service providers to a VO and publish their services along with the price. In order to support more broad services discovery, we had extended GMD by providing Middleware Type and Grid Application Model (GAM) publication, and allowed providers to publish additional properties of the services such as assess location.

In a grid environment, many services, which have same functionality and user interaction, may be provided by different organizations. Also, a service may be replicated and deployed in many locations. From the user's point of view, it is better to use a service that offers a higher performance at a lower price. Therefore, a method is needed to allow users to find replicated services easily. We have created Grid Application Model (GAM) which is derived from tModel in UDDI [22]. A GAM is a set of specifications and APIs for a grid application. The GAM can be published by the service providers within GMD, and the users can search GMD for services conforming to a particular GAM.

Unlike UDDI, GMD also provides more detailed Grid service attributes. The major attributes of a Grid service are service name, service type, hardware price, software price, node host name, access point, middleware type and grid application model.

In the case of our workflow system, if the locations of the resources for tasks are not indicated in the workflow description, the scheduler uses the executable name as a GAM name to query GMD. GMD will return a list of resources which are able to execute the task.

6. Workflow Scheduling System

Given the complexity of the grid workflow execution, we have designed a decentralized scheduling system which supports just in-time planning and allows the decisions of the resource allocation to be made and changed at run-time.

6.1 Architecture

We believe that decentralized scheduling architecture is more efficient over the centralized scheduling for complex workflow processing, which handles all tasks by one scheduler. In our system, every task has its own scheduler called task manager (TM) which implements a scheduling algorithm and handles the processing of the task, including resource selection, resource negotiation, task dispatcher and failure processing. The lifetimes of TMs, as well as the whole workflow execution, are controlled by a workflow coordinator (WCO).

As shown in Figure 4, dedicated TMs are created by WCO for each task. Each TM has its own monitor which is responsible for monitoring the health of the task execution on the remote node. Every TM maintains a resource group which is a set of resources that provided services required for the execution of an assigned task. TMs and WCO communicate through a event service server (ESS).

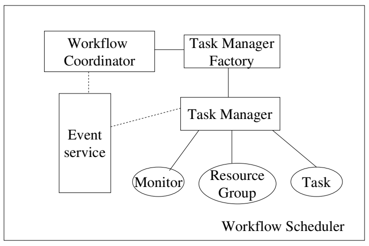

Figure 4. Scheduling Architecture.

6.2 Communication Approach

A communication approach is needed for schedulers in the decentralized scheduling system. On one hand, every task manager is an independent scheduler and they can be run in parallel. On the other hand, the behavior of each task manager may depend on the processing status of other task managers according to the tasks dependencies. For example, a task manager should not execute the task on a remote node if its input generated by its parent tasks is not available for any reason.

In addition, in a workflow, a task may have more than one input that comes from different tasks. Furthermore, the output of these tasks may also be required by other task managers as well. So the communication model between the task managers is not just one-to-one or one-to-many, but it could be many-to-many depending on task dependencies of the workflow.

In our system, we use event-driven mechanism with subscription-notification model supported by the tuple spaces model to control and manage scheduling activities.

6.2.1 Event-driven Mechanism

In the system, the behaviors of task managers and workflow coordinator are driven by events. A task manager is not required to handle communication with others and only generates events according to task's processing status. At the same time, the task managers take actions only according to occurred events without concern for details of other task managers.

The event notification is based on subscription-notification model. WCO and TMs just subscribe the events of interest after activation, and then they can do whatever they want. When a subscribed event occurs, they will be informed.

The benefit of the event-driven mechanism is it provides a loosely-coupled control; hence the design and development of the system are very flexible.

6.2.2 Event Exchange and Tuple Spaces

Where to put event messages and how to get event messages needs to be addressed. We have utilized tuple spaces to exchange events.

The tuple space model originated at Yale with the Linda system [25][26][27]. A tuple is simply a vector of typed values (Fields). A tuple space is a collection of tuples which can be shared by multi-parties by using operations such as read, write and delete. In our work, we have leveraged IBM's recent implementation of tuple spaces called TSpaces [28] to be ESS.

We defined three type event tuples: task status event, output event and job status event. Task status events sent by the TMs to inform the status of task managers. Output events are sent by TMs to announce that its output is ready along with location of its storage. Job status events are used to indicate the execution status of jobs on remote grid nodes. A job is a unit of work a TM sends to a grid node, one task may create more than one job.

As illustrated in Figure 5, TMs inform each other and communicate with WCO through ESS. For example, TMs put their task execution status (e.g. executing, done, failure) into ESS, and then ESS notifies WCO. Every executed TM subscribes its input events generated by TMs of its parent's tasks. Once a TM sends an output event to ESS, ESS notifies corresponding children TMs immediately.

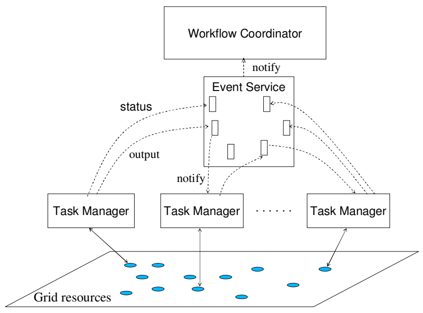

Figure 5. Event-driven Mechanism.

The other benefit from using tuple spaces is: additional components are easy to plug in. For example, a monitoring portal, can subscribe status event from tuple spaces without modifying scheduling system.

6.3 State Transition

The state transition of WCO is illustrated in Figure 6. WCO registers to ESS and start TMs of first level tasks, and then monitor activated TMs. Upon receiving execution status from a TM, WCO starts the TMs of its children tasks. If WCO receives a status *done* event, it checks whether other TMs are still running. If so, WCO goes back to *monitoring*, otherwise it exits. If WCO receives a failed event from a TM, it goes to *failure processing*, and then ends.

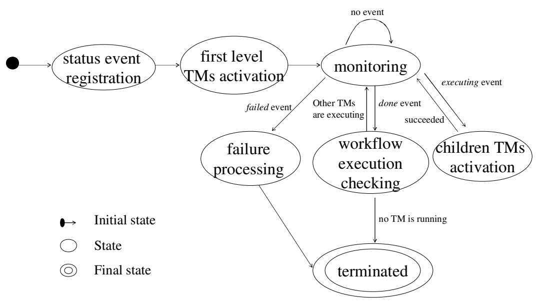

Figure 6. State Transition of WCO.

The state transition of TMs is illustrated in Figure 7. TM registers its output events generated by its parent tasks and waits for the output events, when an event comes; TM goes to *event processing* state which records the information of input data. If any other input data is still not available, TM goes back to *wait* state; otherwise, it transfers to the *resource matching* state. In *resource matching*, it takes a resource from resource group which is created by querying GMD. If a suitable resource is available, TM submits job to the resource and then monitors the status of job execution on the remote resource. If the execution failed, TM goes to *resource matching* again and takes another resource from resource group and then submits job to it. If there is no available resource, TM ends.

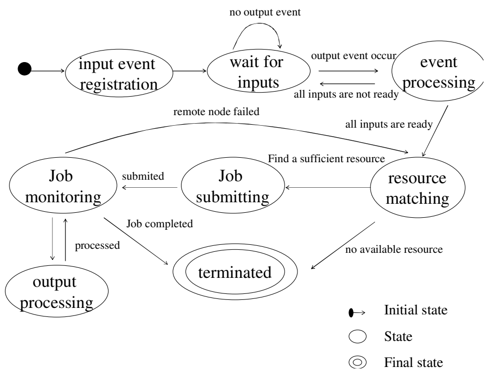

Figure 7. State Transition of TM.

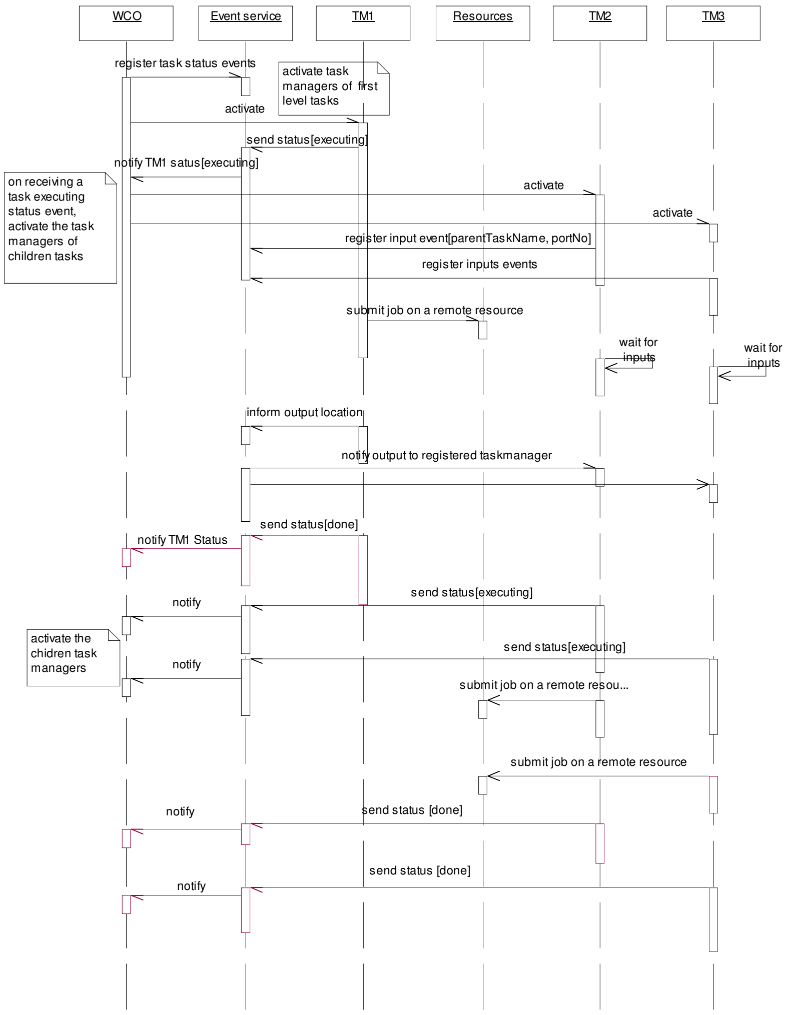
WCO
Event service
TM1
Resources
TM2
TM3
register task status events
activate task managers of first level tasks
activate
send status[executing]
notify TM1 satus[executing]
on receiving a task executing status event, activate the task managers of children tasks
activate
activate
register input event[parentTaskName, portNo]
register inputs events
submit job on a remote resource
wait for inputs
wait for inputs
inform output location
notify output to registered taskmanager
send status[done]
notify TM1 Status
send status[executing]
notify
activate the chidren task managers
send status[executing]
notify
submit job on a remote resou...
submit job on a remote resource
notify
send status [done]
send status [done]
notify

Figure 8. Interaction sequence diagram.

6.4 Interaction

The interactions between the WCO, TMs, ESS and remote resources are illustrated in Figure 8. Firstly, WCO needs to register to ESS and subscribe the events of task execution status. And then WCO activates task managers of first level tasks, in the example, only one TM1. After TM1 finishes the preprocessing of the task execution, it sends a message to ESS saying "*I am executing the task*". ESS informs the WCO and WCO activates TMs of children tasks of TM1, namely TM2 and TM3 in this example. The inputs of the task of TM2 and TM3 rely on the output of the task of TM1, so TM2 and TM3 register to ESS and listen to its output events. Once TM1 identifies a suitable resource, it submits task to that resource. As soon as TM1 knows the output of the task, it informs TM2 and TM3 through ESS, saying *"my output of port No. x is ready and its location is xxxx"*. If all input data of the task of TM2 and TM3 are ready, TM2 and TM3 reports execution status to ESS, and then proceeds to the execution of their tasks. After WCO receives the notification of the execution of TM2 and TM3, WCO will activate their children task managers, so they can begin to listen to their inputs and prepare task execution. This process will be continued until the end of workflow execution.

7. Objected Oriented Implementation

The basic technologies used in our WFEE implementation are Java and IBM TSpaces [26][28] apart from grid technologies mentioned earlier.

7.1 Design Diagram

The class design diagram of WFEE is shown in Figure 9. XMLParsingToModel parses XML formatted workflow description into java objects which are instances of class of Task, Port, DataConstraint. These objects are collected by WorkflowModel. WorkflowModelToDiGraph converts WorkflowModel into directed graph represented by class DiGraph which

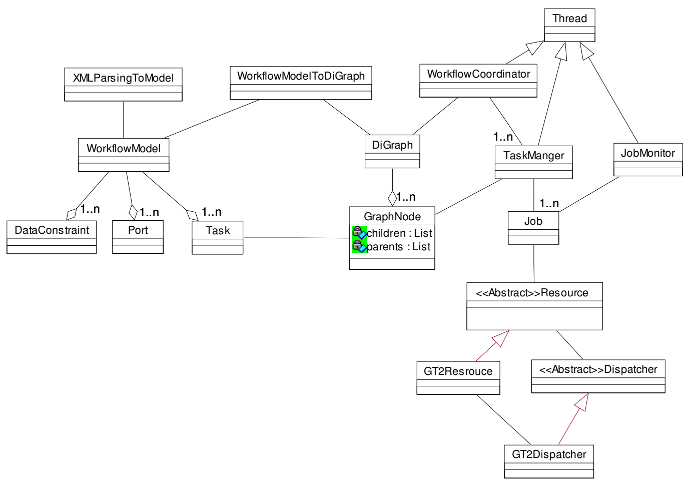

Figure 9. Class Diagram of WFEE.

Event name	Field1	Field2	Field3	Tuple template for registration
task status event	task No.	"status"	value	new Tuple(new Field(String), status, new Field(String))
output event	task No.	port No.	value	new Tuple(taskNo, portNo, new Field(String))
job status event	job No.	task No.	value	new Tuple(new Field(String), taskNo, new Field(String))

Table 1. The Format of Events.

encompasses many GraphNode objects. An instance of GraphNode recodes its parent nodes and children nodes. WorkflowCoordinator creates and controls the instantiation of TaskManager according to the graph node dependencies. Job class presents a unit work assigned to a Grid resource. Every job has a monitor implemented by JobMonitor to monitor job execution status on the remote node. In order to extend WFEE to support multiple Grid middleware, we abstract class *Resource* and *Dispatcher* which provides interfaces to interact with Grid resources.

7.2 Event Messages

There are three types of event tuples whose format is shown in Table 1. Task status event is sent by TMs and WCO use it to control TMs activation. The first field is the ID of the task, second field is string "status" to indicate the type of tuple for the registration purpose, and third field gives the value of status.

TMs need output events sent by their parent TMs to know whether their inputs are available. Task ID is given in the first field, second field is port No. and third field is the location of output.

Job status events are sent by the job monitor. Every job status event gives job ID and its task ID with status value. TMs make decisions according to the job events. For example, when a job is failed, the TM can reschedule this job to another resource in the resource group.

The tuple templates are used for subscribing to the corresponding event. For example, registration to the tuples whose second field is a string "status" can receive all task status events.

8. Experiments

We tested WFEE by creating a synthetic applications/workload that attempted to simulate a real workflow. We used Australian Belle Grid test-bed resources located in Sydney, Canberra, and Melbourne (see Table 2). The synthetic programs we created for experiments are *xcalc*, *ycalc*, *addcalc* and *merge*. Basically, *xcalc*, *ycalc* and *addcalc* programs execute mathematical functions and have varying computation complexity based on the values of two inputs. The inputs of these three programs have different formats; both inputs of the *xcalc* are parameter type; *ycalc* has one file type input and one parameter input, while *addcalc* has two file type inputs. They all save the result in a file. The *merge* program merges three input files into one file.

We have created a multi-stage workflow shown in Figure 10. The result output *Fa* of task A is needed by tasks B, C and D. The tasks E, G and F depend on the output from B, C and D. Finally the results of E, G and F are merged together by the task H. The program that tasks used and their input and output information are shown in Table 3 with the execution time when run independent of each other on belle.cs.mu.oz.au node.

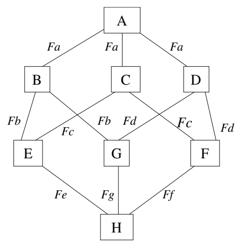

Figure 10. Workflow Diagram of the Experiment.

Node	Type	CPU	Location
belle.cs.mu.oz.au	IBM eServer	4	Melbourne
belle.anu.edu.au	IBM eServer	4	Canberra
belle.physics.usyd.edu.au	IBM eServer	4	Sydney
fleagle.ph.unimelb.edu.au	PC	1	Melbourne

Table 2. Test-bed Machine List in Australia.

Task	Program	Input 1		Input 2		Output		Time
		type	value	type	value	type	value	
A	xcalc	parameter	3	parameter	4	file	0121869	3m59.849s
B	ycalc	file	0.121869	parameter	4	file	0.071878	3m59.997s
C	ycalc	file	0.121869	parameter	5	file	0.089274	4m59.997s
D	ycalc	file	0.121869	parameter	6	file	0.106644	5m59.997s
E	addcalc	file	0.071878	file	0.089274	file	0.002813	4.996s
F	addcalc	file	0.089274	file	0.106644	file	0.003116	5.996s
G	addcalc	file	0.071878	file	0.106644	file	0.003419	5.996s
H	merge	merge three input files into one file						0.005s
Total								**19m13s**

Table 3. Task Details and the Execution Time on belle.cs.mu.oz.au.

We initiated the workflow execution on a resource located in Melbourne and workflow tasks are distributed to resources located in Sydney, Canberra and Melbourne. We registered the details of the deployed applications such as access point, middleware type within GMD. Figure 11 presents the definition of task C, D and F with the data flow between them defined in the datalink tag. Task C is specified to execute on belle.anu.edu.au. As there is no location defined for task D, the WFEE locates it by querying GMD at run time.

The start time of submitting tasks on the discovered remote nodes and end time of the task execution for each task are shown in Table 4. Its process diagram is illustrated in Figure 12. After completion of task A, the children tasks B, C and D were run in parallel. The task E started earlier than G and F, since they had to wait for completion of D.

The start time we measured is the time TM started to submit job to a remote resource. Theoretically, after task A finishes, its children tasks should be submitted immediately, however, there are some time intervals between the parent tasks end time and children task start time. That gap can be regarded as due to WFEE running overhead, including Tuple-space communication delay, event notification process, GMD query delay and event processing. However, compared to the running time of tasks, the gap is insignificant.

```
<task name="C">
      <executable>
        <name>ycalc</name>
          <host>belle.anu.edu.au</host>
          <accesspoint type="GT2Gram">/data/ycalc.sh</accesspoint>
          <input>
                   <port0 type="file">para</port0>
                   <port1 type="msg">5</port1>
          </input>
          <output>
                   <port2 type="file">output</port2>
          </output>
      </executable>
</task>
<task name="D">
      <executable>
        <name>ycalc</name>
          <input>
                   <port0 type="file">para</port0>
                   <port1 type="msg">6</port1>
          </input>
          <output>
                   <port2 type="file">output</port2>
          </output>
      </executable>
</task>
<task name="F">
     <executable>
          <name>addcalc</name>
          <input>
                   <port0 type="file">para1</port0>
                   <port1 type="file">para2</port1>
          </input>
          <output>
                   <port2 type="file">output</port2>
          </output>
     </executable>
</task>
```

```
<datalink>
   <link>
     <from>C:port2</from>
     <to>F:port0</to>
   </link>
   <link>
     <from>D:port2</from>
     <to>F:port1</to>
   </link>
</datalink>
```

Figure 11. XML workflow description for C, D and F.

Task	Node	Start time (min)	End time (min)	Time (min)
A	belle.cs.mu.oz.au	0	4.137	4.137
B	belle.cs.mu.oz.au	4.169	8.822	4.652
C	belle.anu.edu.au	4.174	9.66	5.486
D	belle.physics.usyd.edu.au	4.281	10.684	6.403
E	belle.anu.edu.au	9.669	10.097	0.427
F	belle.physics.usyd.edu.au	10.708	11.145	0.436
G	belle.cs.mu.oz.au	10.688	11.152	0.463
H	belle.cs.mu.oz.au	11.172	11.394	0.222
WFEE Execution time		0	11.394	11.394

Table 4. Concurrent Workflow Execution on Grid Resources.
.

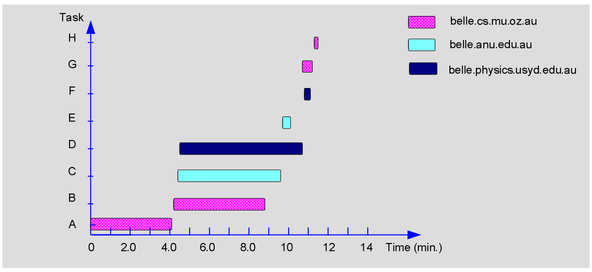

Figure 12. Execution Progress.

We also simulated a dynamic environment for WFEE. After we started WFEE again, we moved the application for task D and F from belle.physics.usyd.edu.au to fleagle.ph.unimelb.edu.au and updated the entries in GMD. We accomplished updates before D and F execution. At the time of D and F execution, the TMs of D and F discovered the new location and submitted to fleagle.

From the results of serial and parallel execution of workflow, it can be observed that in some cases the execution of a task on a remote node takes more time. This is mainly due to overhead involved in initiating remote execution and transmission of inputs and outputs between user and remote nodes. However, we notice that the total time of running all tasks on one machine in serial (19min 13sec) is larger than concurrent execution on distributed grid resources (11 min 23sec).

The experiment proves that WFEE can discover resources at run time, execute distributed installed applications in parallel and sequence, and transfer input and output data for applications automatically. Thus, new application services can be derived by composing standalone distributed resources using WFEE. Additionally, high throughput can be achieved by taking advantage of distributed computational resources.

9. Conclusion and Future Work

In this paper, we presented our current work for the Grid workflow enactment engine. The new contribution of the work is that it provides decentralized just-in-time scheduling which can adapt to heterogeneous and dynamic grid environments. Using tuple spaces model, event-driven mechanism and subscription-notification approach makes the workflow execution loosely-coupled and flexible. We also achieved services discovery at run-time by using Grid Market Directory. WFEE links geographically distributed standalone applications and takes advantage of distributed computational resources to achieve high throughput. Also the intermediated data can be transferred between Grid nodes automatically.

We have integrated our WFEE into a grid middleware – Globus 2.4 for remote job execution. Basic XML based data flow language is also provided for end users.

Our future work will focus on workflow execution optimization. We will be extending resource allocation algorithms to support optimal and QoS (Quality of Service) requirements based scheduling, using computational economy. In addition, we are extending WFEE to support loop and advanced workflow control patterns whose requirements are derived from applications in biology and natural language processing domains.

Acknowledgements

We would like to thank Srikumar Venugopal, Chee Shin Yeo, Anthony Sulistio and Rob Gray for reviewing the paper. We also want to thank Baden Hughes and Kim Branson for providing workflow requirements of their applications in natural language engineering and molecular modeling.

The work presented in this paper is partially supported through the Australian Research Council Discovery Project grant and Storage Technology Corporation sponsorship of Grid fellowship.

References

[1]. I. Foster and C. Kesselman (editors), *The Grid Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2]. JunWei Cao, et al., *GridFlow: Workflow Management for Grid Computing*, 3rd International Symposium on Cluster Computing and the Grid, May 12 - 15, 2003 Tokyo, Japan.

[3]. Giacomo Piccinelli, *Workflow for a Grid of Services*, http://www.cs.ucl.ac.uk/staff/G.Piccinelli /GridWorkflow.pdf

[4]. I. Foster, C. Kesselman, S. Tuecke, *The Anatomy*

of the Grid: Enabling Scalable Virtual Organizations, International J. Supercomputer Application, 15(3), 2001

[5]. Ewa Deelman, et al., *Pegasus : Mapping Scientific Workflows onto the Grid,* Across Grids Conference 2004, Nicosia, Cyprus

[6]. Globus Project. http://www.globus.org

[7]. Srikumar Venugopal, Rajkumar Buyya and Lyle Winton, *A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids*, Technical Report, The University of Melbourne, February 2004. http://www.gridbus.org/papers/gridbusbroker.pdf

[8]. Jia Yu, Srikumar Venugopal, and Rajkumar Buyya, *A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services*, Technical report, The University of Melbourne, 2003 http://www.gridbus.org/~raj/papers/gmd.pdf

[9]. Condor Team. The directed acyclic graph manager, http://www.cs.wisc.edu/condor/dagman, 2004

[10]. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, Chimera: *A Virtual Data System for Representing, Querying, and Automating Data Derivation*, presented at Scientific and Statistical Database Management, 2002

[11]. MyGrid Project. http://www.mygrid.org.uk/

[12]. McCann, Karen M., Maurice Yarrow, Adrian DeVivo, and Piyush Mehrotra, *ScyFlow:An Environment for the Visual Specification and Execution of Scientific Workflows*, In Proceedings of Workflow in Grid Systems Workshop in GGF10, at Berlin, Germany,March, 2004.

[13]. Romberg, M. *The UNICORE Architecture-Seamless Access to Distributed Resources*, Proceedings of 8th IEEE International Symposium on High Performance Computing, Redondo Beach, CA, USA, August 1999, pp. 287-293

[14]. ICENI Project. http://www.lesc.ic.ac.uk/iceni/

[15]. Kaizar Amin, et. al., *GridAnt: A Client-Controllable Grid Workflow System*, 37th Hawaii International Conference on System Sciences, 2004

[16]. SWFL-Service Workflow Language. http:// http://www.cs.cf.ac.uk/User/Yan.Huang/GridWF/SWFL.htm

[17]. Sriram Krishnan, Patrick Wagstrom and Gregor von Laszewski, *GSFL: A Workflow Framework for Grid Services*, http://www-unix.globus.org/cog /papers/ gsfl-paper.pdf

[18]. BPEL4WS Specification. http://www-106.ibm.com/developerworks/library/ws-bpel/

[19]. D. Abramson, J. Giddy, and L. Kotler, *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?*, International Parallel and Distributed Processing Symposium (IPDPS 2000), May 1-5, 2000, Cancun, Mexico, IEEE CS Press, USA, 2000.

[20]. W3C Web Services. http://www.w3.org/TR/ws-arch/

[21]. OGSI - Open Grid Services Infrastructure. http://www.ggf.org/documents/GWD-R/GFD-R.015.pdf

[22]. UDDI Specifications. http://www.uddi.org

[23]. GridFTP http://www.globus.org/datagrid/gridftp.html

[24]. Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman, Workflow Management in GriPhyN, *The Grid Resource Management*, Kluwer 2003

[25]. Carriero, N., and Gelernter, D., *Linda in Context*, Communications of the ACM, Vol. 32, No. 4, pp.444-458, April 1989

[26]. D. Gelernter, *Generative Communication in Linda*, TOPLAS 7, No. 1, 80-112 (1985)

[27]. D. Gelernter and A. J. Bernstein, *Distributed Communication via Global Buffer*, Proceedings of the ACM Principles of Distributed Computing Conference (1982), pp. 10-18

[28]. P. Wyckoff, *T Spaces*, IBM Systems Journal, Vol. 37, Nov 3, 1998, http://researchweb.watson.ibm.com/journal/sj/373/wyckoff.html

[29]. Ian Taylor, Matt Shields, Ian Wang and Roger Philp, *Distributed P2P Computing within Triana: A Galaxy Visualization Test Case*, IPDPS 2003 Conference, April 2003