A Novel Approach for Realising Superscalar Programming Model on Global Grids

Xingchen Chu[1], Rajkumar Buyya[1] and Rosa M. Badia[2]

[1]GRIDS Laboratory
Dept. of Comp. Science and Software Eng.,
The University of Melbourne, Australia
{xchu, raj}@csse.unimelb.edu.au

[2]Barcelona Supercomputing Center
Technical University of Catalonia
Barcelona, Spain
rosa.m.badia@bsc.es

Abstract

This paper presents a novel approach for the design and implementation of Grid Superscalar (GS) model on top of GWFE (Gridbus Workflow Engine). This new workflow-based version of the GS framework enables the easy development of applications (without the need of explicit expression of parallelism/distribution by the programmer) and scheduling them on Global Grids using the GSB (Gridbus Service Broker) transparently. By means of a simple programming model, GWFE-S provides a pure Java library and keeps the Grid as transparent as possible to the programmer. Moreover, the deployment of the applications is highly optimized by using the GSB which supports various types of Grid middleware. The runtime of superscalar has been designed to follow the Gridbus Workflow and is therefore formed by a set of dependent workflow tasks which will be scheduled and executed to different Grids. The feasibility of the work is demonstrated by conducting performance evaluation on a global Grid having resources located in Australia and USA.

1. Introduction

Grids now emerges as the next generation of distributed computing platforms for solving scientific and engineering problems that are computational and data intensive. There are a lot of efforts that have been made to develop Grid middleware and applications that uses Grids. However, Grids still like technologies that are not very easy to use, and only very experienced developers can write Grid applications. The difficulty associated with developing applications to be run on the Grid is a major barrier to adoption of this technology by non-expert users. The challenge in this case is to provide programming environments for Grid-unaware applications, defined as applications where the Grid is transparent to them but that are able to exploit its resources.

Grid Superscalar (GS) [1] is an innovative technology that provides an easy-to-use programming environment for non-expert users to develop Grid applications in a normal sequential manner. It reduces the requirement of being aware of Grids and explicitly expressing application parallelism. The application code that is written using this model can be internally translated into a workflow and will be scheduled by the GS runtime system.

Gridbus Workflow Engine (GWFE) [2] provides users a workflow engine that can run workflow applications in Grids. The tasks inside the workflow will be automatically scheduled via the Gridbus Service Broker (GSB) [5]. The GSB provides several QoS-aware scheduling algorithms and supports various types of Grid middleware including Globus [3], PBS [4], Condor [6], SGE [7], Aneka [8], and plain SSH. Utilizing the GWFE with GSB provides a powerful approach to run workflow applications on Global Grids.

This paper presents a novel approach for realising the superscalar programming model via Gridbus middleware, which provides a way to develop and deploy superscalar applications on Global Grids. It is organized as follows. Section 2 discusses some related work. Section 3 proposes a GWFE-based Superscalar (GWFE-S) system architecture. Section 4 presents the programming model for the applications that use GWFE-S. Section 5 gives some implementation details about the GWFE-S and describes the internal processes and communications that take place inside the runtime when executing an application. Section 6 discusses the results of some tests on applications that use GWFE-S on global Grids. Section 6 concludes the paper.

2. Related works

There are number of efforts that have promised to provide programming environments and tools to simplify the development of Grid applications. Some projects such as GrADS [9], introduce a special language along with a compiler in order to grid enable the applications in such a way that the applications can be compiled and run on their specific infrastructure. Other efforts like GriddLeS [10], aim to provide a more general environment that facilitates the composition of arbitrary grid applications from legacy software. It supports the construction of complete applications without source modification to the existing legacy program. The GS model unlike those approaches tries to make programming grid applications as the same as programming normal sequential applications. It means, unlike constructing the applications by linking different working legacy programs, developers still need to write the application code, however, in this case unlike learning and using new programming languages, developers can work with the existing programming language such as C++/Java, and they can write grid applications just like write normal sequential applications.

Apart from GWFE-S, there are other efforts exist for linking GS model with other Grid systems. GS-PGPORTAL [11] describes the possible integration solution of P-GRADE [13] and GS system to create a high level, graphical grid programming, deployment and execution environment that combines the workflow-oriented thin client concept of the P-GRADE Portal with the automatic deployment and application parallelization capabilities of GS. The difference between that and GWFE-S is that GS-PGPORTAL was trying to build the workflow using the P-GRADE portal and utilizing the GS runtime to run those tasks. GWFE-S builds the workflow in the opposite manner, which is dynamically generated by the superscalar applications.

The most recent work related to the integration of superscalar model is COMPs [14]. COMPs provides a superscalar model implementation based on Grid Component Model (GCM) [15]. As a result, the runtime of COMPs has gained in reusability, deployability, flexibility and separation of concerns which are from the component-based programming practice. This work also benefits the ProActive [16] by means that Java developers of ProActive now can utilize superscalar model based on its framework and runtime environment.

Our approach differs from the above approaches by means of combining the benefits of most of the features provided by those solutions. GWFE-S provides native support to compose the superscalar model as a workflow via the GWFE, the developers do not need to worry about how to construct the workflow manually as the GWFE-S will automatically detect all the dependencies and construct the workflow at runtime. It also provides support for a dynamic scheduling infrastructure to run tasks on various types of Grid middleware via GSB. Furthermore, it is pure Java based solution which will help developing superscalar applications use Java.

3. GWFE-S Architecture

The architecture of the GWFE-S, as shown in Figure 1, is primary based on the runtime environment provided by GWFE and therefore it reuses the entire system as the base infrastructure:

- Native workflow support: GWFE provides a XML-based workflow description language which can be internally translated into direct acyclic graph (DAG) and automatically schedules tasks and resolves data dependencies between tasks.
- Just in-time scheduling: it enables the resource allocation decision to be made at the time of task execution and hence adapt to changing Grid environments.
- Various Grid middleware support: it also supports schedule tasks on Global Grids via the GSB which allows multiple Grid middleware environment for executing the tasks such as Globus 2.4 and Globus 4.0, PBS, Condor, SGE, Aneka, or plain SSH.
- Easy Deployment: the deployment of the applications over various Grid middleware is fairly easy via the XML-based service and credential description language.

Most of the components in the system are reused from the existing infrastructure provided by the GWFE and GSB. The next section describes the two important components that bring superscalar model into the picture.

3.1. Context manager

The context manager is responsible for maintaining the metadata information for the superscalar applications. It consists of a IDL (Interface definition language) parser which is used to scan the IDL file and resolve any metadata about the application such as application class name, method signature and its parameter information. This information will be further processed

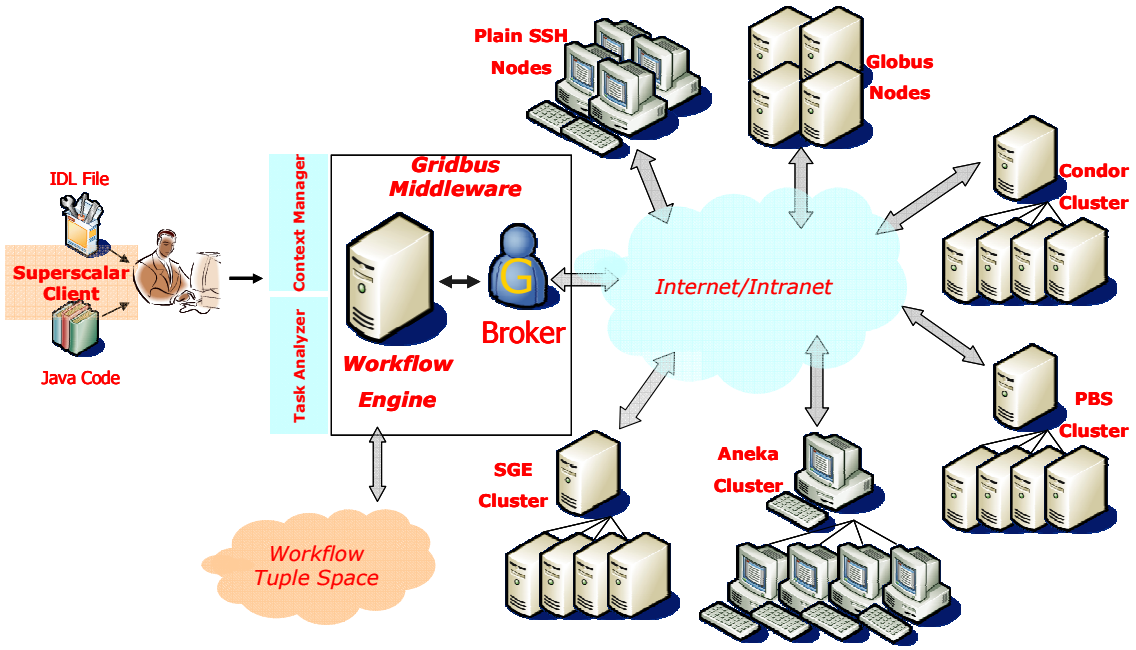

Figure 1. GWFE-based Superscalar System Overview

by the task analyzer to construct the tasks and their dependencies.

3.2. Task analyzer

It intercepts the application based on the method invocations to dynamically generate the workflow task and its dependencies at runtime. Just like the concepts in GS, a workflow task is just a method invocation which matches certain method metadata obtained by the Context manager from the IDL file. A XML file described the entire workflow will be generated and submitted to the GWFE runtime once the application triggers a certain method call (see details in section 4).

3.3. Task schedule and execution

The GWFE is responsible for resolving the task graph and all the dependencies between different tasks. There is a build-in workflow scheduler that is used to schedule tasks whose dependencies have been resolved for deploying on various remote Grid resources. The runtime environment communicates with a selected remote Grid resource for an assigned task execution. The infrastructure for scheduling and execution workflow on Global Grids described here has been leveraged without making any changes to the base software infrastructure.

4. GWFE-S Programming Model

GWFE-S aims to provide the same promise as the GS which is an easy-to-use programming model to enable applications for Grid without knowing about the Grid. It is so-called 'Grid-unaware applications' which is programmed in a sequential fashion. Nevertheless, by means of GWFE-S, it identifies the tasks that compose the application, detects task dependencies, dynamically generates the workflow on the fly, decides when to distribute the task to the Grid and manages their remote execution.

The following subsections explain, through a very simple, how to create a GWFE-S application.

```
for ( int i = 0; i < loops; i++ )
{
  Mean.genrandom("random.txt");
  Mean.mean("random.txt", RESULT);
}

//post processing the result
printResult();
```

Figure 2. Sequential Code of Mean

4.1. Original Sequential Code

Consider a Java application that generates random numbers and calculate the mean value of those random numbers. From now on, we call it *Mean*. Figure 2

shows the original sequential code of Mean application. All parameters of the methods are files. The program first generates random numbers into a random.txt file and then reads that file and appends the results to the result file.

4.2. Define the tasks

The first step consists in defining which tasks it must take into account, that is, which methods called from the application code will be actually executed on the Grid. This is done by providing a IDL (interface definition language) file which declares these methods. It is the exactly the same approach that has been used in the GS. As the current implementation, we are trying to make the least impacts on the original GS programming environment, the adoption of using IDL is one of these concerns which enables reusing the existing GS programs. The IDL provides the metadata required by the Task analyzer module to intercept the corresponding method invocations on the target class to dynamically generate workflow tasks. Figure 3 corresponds to IDL definition of the tasks of Mean.

```
interface Mean {
    void genRandom( out File rnumber_file );
    void mean( in File rnumber_file, inout File results_file );
};
```

Figure 3. IDL of the Mean application

4.3. Preparing the application

The second step involves making the application invoke the runtime of GWFE-S. This runtime must be started in order to receive the tasks to submit to the Grid and know about the files that the application accesses. Obviously, the original code of a sequential Java application cannot itself interact with GWFE-S. For that reason, we developed a interceptor-based aspect that intercepts the application at execution time by inserting some necessary logic in it; the intercepting method invocation at runtime were featured by JBoss-AOP (Aspect Oriented Programming) framework.

```
GSMaster.On();

for ( int i = 0; i < loops; i++ )
{
  Mean.genrandom("random.txt");
  Mean.mean("random.txt", RESULT);
}

GSMaster.Off(1);
//post processing the result
printResult();
```

Figure 4. Code of Mean that triggers GWFE-S

In order to enable the GWFE-S, the programmer can use up to 2 API methods (GSMaster.On and GSMaster.Off) in the application code. In particular, the API offers methods to start and stop the runtime. Figure 4 shows the modified code of Mean, resulting from the inclusion of API calls; note that the invocations of genrandom and mean remain the same, since it is the duty of the AOP interceptor to translate them into the creation of workflow tasks. Also note that although the GSMaster style may look similar to the GS, but it is a totally different implementation that works particularly with the GWFE-S runtime.

5. Design and Implementation

In order to fully understand the design and implementation of the GWFE-S runtime environment, the next subsections describe the base technologies, the different phases that are required to configure and deploy the runtime environment, as well as its underlining operations when executing an application.

5.1. Base Technologies

To implement GWFE-S, we took Java as the programming language, and JBoss AOP 2.0, GWFE 2.0 and GSB 3.0 as the base technologies.

JBoss AOP [17] is a 100% pure Java aspect oriented programming framework which allows the developers to insert behavior between the caller of a method and the actual method being called. It provides an abstraction called interceptor which can be configured to bind to certain method invocation via a XML configuration file. The task analyzer component is built primary based on the interceptor concept. A *SuperscalarInterceptor* class deriving from the *org.jboss.aop.advice.Interceptor* interface has been implemented, which generates a workflow task for a specific method invocation, adds any dependencies of that task to the workflow, and postpones the actually invocation of that method as the logic of the method will only be executed remotely.

GWFE is a Java based workflow engine that facilitates users to execute their workflow applications on Grids. It provides a XML-based workflow language for the users to define tasks and dependencies. It uses the tuple space (IBM TSpaces implementation) [18] approach to enable an event-driven scheduling architecture for simplifying workflow execution. All the tasks that have been dynamically generated by the

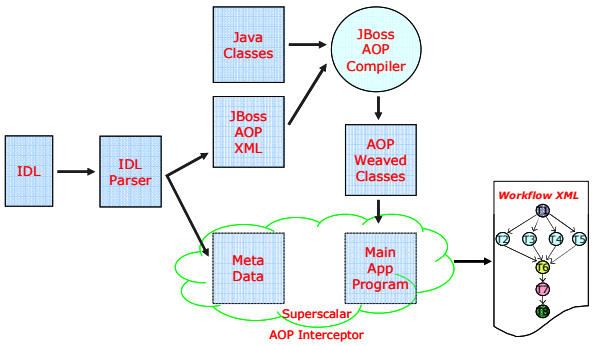

Figure 5. Phase I: Workflow Creation

task analyzer are objects representing certain XML element, those tasks and dependencies will be saved into the workflow XML descriptor and submit to the workflow engine once the application triggers the GSMaster.Off invocation. The contribution of the GWFE is mainly to build the DAG, resolve the dependencies between each task and submit the ready tasks to the broker.

GSB is an user-level Grid middleware that mediates access to distributed Grid resources. It supports various types of Grid middleware including Globus, Alchemi, PBS, Condor, SGE and also plain SSH. The major contribution of the GSB in our approach is to manage the Grid resources and the execution of workflow tasks on Grids.

5.2. Configurations

Before launching any superscalar applications with GWFE-S, there are 3 points to address regarding to the configurations. First, user needs to specify an XML file which is a list of Grid resources that can be used for GWFE-S to distribute the tasks. Secondly, the user also needs to specify an XML file describing the credentials that can be used for the GSB to execute the tasks. The XML schemas for both resources and credentials configuration file are created by GSB. Lastly, a WEProperties file must be provided to the GWFE to configure the tuple space.

5.3. Phase I : Workflow Creation

Besides the external configurations, the execution of the superscalar applications within GWFE-S is composed of two phases. The first phase as shown in Figure 5 is recognized as workflow creation. The main purpose of this phase as indicated by the name is to create the XML workflow descriptor that will be submitted to the GWFE. Phase one consists of two sub-phases: *static AOP weaving* and *dynamic task analyzing*.

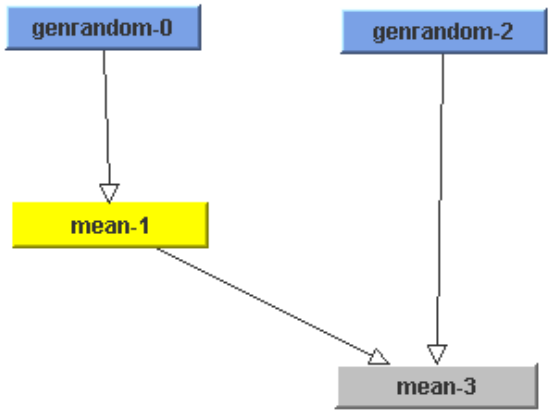

Figure 6. Workflow for Mean Application

5.3.1. Static AOP weaving. As the JBoss AOP requires a *jboss-aop.xml* file to identify which method invocations should be intercepted by the AOP interceptor in order to insert instructions into the classes, the *IDLParser* is responsible for analyzing the IDL file to store metadata including application name, method details, and it is able to utilize the metadata to generate the AOP configuration file. According to the AOP XML, and the implementation classes for the application, the instructions of the required logic can be weaved into the classes via a JBoss AOP compiler tool at compilation time. The modified AOP weaved classes that contain specific instructions to the AOP interceptor will be used by the GWFE-S runtime.

5.3.2. Dynamic Task Analyzing. Once the weaved classes have been successfully generated, the AOP interceptor will be triggered when a specific method invocation occurs within the superscalar application. Then it is the *SuperscalarInterceptor*'s responsibility to translate the normal method invocation to a workflow task with all its dependencies by looking at the metadata provided by the *IDLParser*. Once the

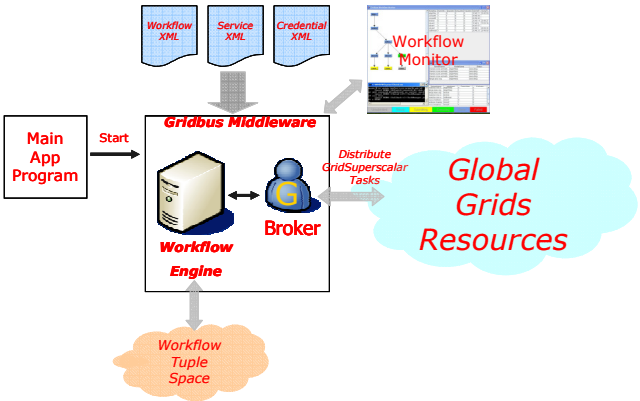

Figure 7. Phase II: Workflow submission

application calls the *GSMaster.Off* method, all the generated tasks and dependencies will be saved to the workflow XML file.

The workflow generated for the Mean the one depicted in Figure 6. Tasks with no dependencies pass will be scheduled immediately. According to the graph of Mean, the first suitable tasks are genrandom-0 and genrandom-3: they can be run in parallel on the Grid. Upon the completion of the first two tasks, the tasks with dependencies such as mean-1 and mean-3 will be executed on the Grid.

5.4. Phase II: Workflow Submission

Once phase I has been successfully passed, it comes to the second phase as shown in Figure 7 which is the submission of the workflow to the GWFE along with the services and credentials configurations. Phase two is started inside the GSMaster.Off method, which initializes the workflow monitor, initializes the GSB with the three XML files: workflow, services and credentials files and synchronizes the execution of the workflow. The GSB is responsible for managing the actual execution to the Grids, which is continuously accepting the tasks scheduled by the GWFE and dispatching them to the remote resources. Once all the tasks belonging to the application have been succeeded, relevant output files will be synchronized to the local workstation and results can be displayed by the application program. The GSB, GWFE as well as the monitor will be shutdown once the application finished.

5.5. Task Executor

What we have discussed so far are components on the master node, it is important to also mention how the local method invocation is executed remotely. The GSWorker class uses the Java reflection API to run the a method on a specific class with all required parameters. The information of the method, class and parameters will be automatically given to the GSWorker program at phase I when the workflow descriptor is generated. The GSB will copy the

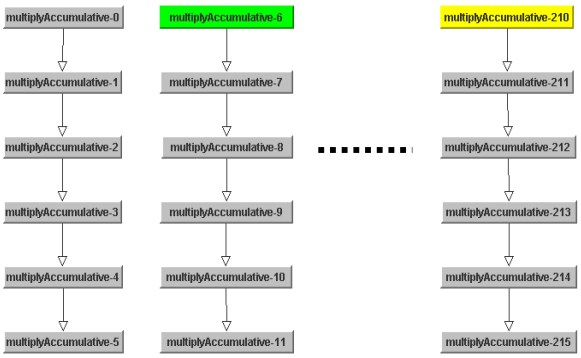

Figure 8. Matmul application experiments

required jar files that contain the GSWorker class as well as the application class, and the remote Grid runtime is responsible for executing the GSWorker program by using standard Java command with the target method name, class name, and arguments for that method. The only restriction on the Grid is that the Java 5.0+ runtime has to be installed. For example, the following shell command will be executed on the Grids:

java –cp GSWorker.jar:GSApp.jar. *GSWorker* Mean mean random.txt result.txt

It invokes the mean method on the Mean class which takes two arguments random.txt and result.txt.

6. Performance Evaluation

This section presents the results of experiment studies performed on GWFE-S. The experiments took place in three different Grid sites as shown in Table 1. Manjra cluster consists of 11 nodes, which is running in CSSE department at the University of Melbourne. Belle is a workstation containing 4 CPUs at the same site as manjra cluster. We have also used up to 18 nodes at State University of New York, Binghampton, USA, where each node contains 4 processors. We have decided to use the plain SSH adaptor provided by GSB which have the least overhead compared with other middleware support such as Globus. The main purpose of the test is to show the GWFE-S works within the context of the GSB as workflows on Global Grids, and meanwhile it provides reasonable performance gain via

Node	***Location***	***Grid Adaptor***	***No. Processors Per Node***	***CPU Info***
manjra.cs.mu.oz.au	The University of Melbourne, Australia	Plain SSH	4	Intel ® Xeon ™ CPU 2.00GHz
belle.cs.mu.oz.au	The University of Melbourne, Australia	Plain SSH	4	Intel ® Xeon ™ CPU 2.80GHz
node**.cs.binghamton.edu (8 different nodes)	State University of New York, USA	Plain SSH	4	Intel ® Xeon ™ CPU 2.66GHz

Table 1. Experiment Setup (start from 01 to 08)**

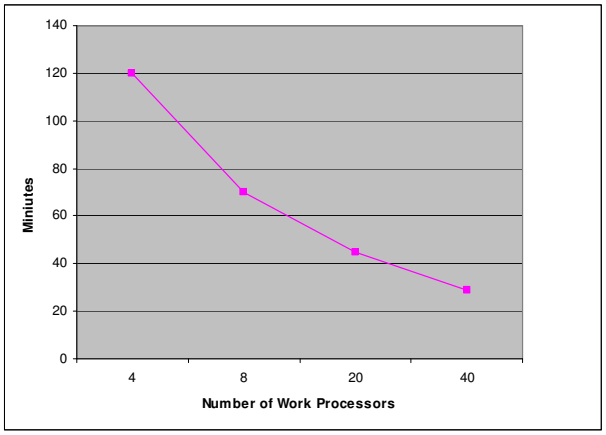

Figure 9. Execution time of Matmul

parallelism on Grids against the sequential program.

The experiment that we have adopted to demonstrate the objective is parallelizing an application that multiplies two matrices (Matmul). The matrices are divided into blocks, which are themselves smaller matrices of doubles. The tasks generated by Matmul work with blocks stored in files. In particular, we used matrices of 6 x 6 blocks, with 800 x 800 doubles in each block. With these input parameters, Matmul produces 216 coarse-grained tasks, each one multiplying two blocks. The corresponding dependency graph as shown in Figure 8 contains 36 groups of 6 pipelined tasks where each task reads the result of the previous one. For this test, the whole GWFE-S runtime (which plays the role of master) was deployed in the client machine, submitting the tasks to a variable number of other nodes (workers). Since not all of the nodes are in the same domain, and due to the security constrains on the local nodes as well, there is no way to provide a NFS-like setup so that the experiment assumes all the input files have to be transferred to the worker nodes which incurs quite large network overhead especially transferring files to the US nodes.

In Figure 9 appear the execution times of Matmul when parallelising it over different numbers of worker processors as the input parameters we mentioned. As can be seen from the results, for this particular execution of Matmul, which generates a medium number of tasks (216 in total), shows the reasonable speedup when the number of processors increases. The performance gain from 4 to 40 processors are reasonably good, the network overhead involved for transferring files between Australia and USA nodes is the main reason for degrading the performance in this application. As a result, the GWFE-S would perform quite well if the time for execution of one task is much longer than the network overhead caused by the file transfer.

7. Summary and Conclusions

This paper has presented GWFE-S superscalar, a new version of GRID superscalar which is an new implementation of the programming model to follow the principles of GWFE, a workflow model intended for the Grid. GWFE-S provides a straightforward programming model that keeps the Grid transparent to the user, who is only required to specify the tasks to be executed on the Grid, being free to leave the code of the Java application completely unchanged. In addition, through an operation example, we have explained how the runtime components of GWFE-S are individually concerned with different functionalities and how they collaborate to reach the common goal of remotely running the application. Finally, we have discussed some test results that explore the possible performance gain of running the application via GWFE-S.

Although in this paper we have focused on Grid-unaware applications, we believe that GWFE-S could offer an alternative way to develop Grid-aware applications as well. As the GSB is highly optimized to support economy-based scheduling policies, the GWFE-S can easily utilize the scheduling infrastructure provided by the GSB to support QoS-aware applications.

8. Acknowledgement

This work is partially supported through an International Science Linkage (ISL) project funded by the Australian Department of Innovation, Industry, Science and Research (DIISR).

9. References

[1]. R.M. Badia, J. Labarta, R. Sirvent, J.M. Pérez, J.M. Cela, R. Grima, "*Programming Grid Applications with GRID Superscalar*", Journal of Grid Computing, Springer, 2003 , pp. 151-170(20).

[2]. J. Yu and R. Buyya, "*A Novel Architecture for Realizing Grid Workflow using Tuple Spaces"*, Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society Press, Los Alamitos, CA, USA, Nov. 8, 2004.

[3]. I. Foster and C. Kesselman, *"Globus: A Metacomputing Infrastructure Toolkit"*, International Journal of Supercomputer Applications, 1997, pp: 115-128.

[4]. A. Bayucan, R. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, *"Portable Batch System: External reference specification"*. Technical report, MRJ Technology Solutions, 1999.

[5]. S. Venugopal, R. Buyya and L. Winton, *"A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids"*, Concurrency and Computation:

Practice and Experience, 18(6), Wiley Press, New York, USA, May 2006, pp: 685-699.

[6]. W. Gentzsch, *"Sun Grid Engine: Towards Creating a Compute Power Grid"*, Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia. IEEE CS Press, Los Alamitos, CA, USA, 2001.

[7]. M. Litzkow, M. Livny, and M. W. Mutka, *"Condor - a hunter of idle workstations"*, Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS 1988), San Jose, CA, USA, IEEE CS Press, Los Alamitos, CA, USA, 1988.

[8]. X. Chu, K. Nadiminti, C. Jin, S. Venugopal, R. Buyya, *"Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications"*, Proceedings of the 3rd IEEE International Conference on e-Science an Grid Computing (e-Science 2007), Dec. 10-13, 2007, Bangalore, India.

[9]. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D., L. Torczon, R. Wolski, "*The GrADS Project: Software Support for High-Level Grid Application Development*", International Journal of High Performance Computing Applications, 2001, pp. 327-344.

[10].J. Kommineni, D. Abramson, and J. Tan. *"Communication over a Secured Heterogeneous Grid with the GriddLeS runtime environment"*, Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing. Dec. 4- 6, 2006, Amsterdam, Netherlands.

[11].R. Lovas, R. Sirvent, G. Sipos, J. Perez, R.M. Badia, P. Kacsuk, *"GRID superscalar enabled P-GRADE portal"*. Proceedings of the Integrated Research in Grid Computing Workshop, Università di Pisa, Dipartimento di Informatica, Nov 2005, pp: 467-476.

[12].S. Venugopal, K. Nadiminti, H. Gibbins, R. Buyya, "*Designing a Resource Broker for Heterogeneous Grids",* Software: Practice and Experience, Wiley Press, New York, USA, July 10, 2008, pp: 793-825.

[13].G. Sipos, P. Kacsuk, "*Classification and Implementations of Workflow-Oriented Grid Portals"*. High Performance Computing and Communications, First International Conference, HPCC 2005, Sorrento, Italy, Sept 21-23, 2005, pp:684-693.

[14].E. Tejedor, R.M. Badia, "COMP Superscalar: Bringing GRID superscalar and GCM Together". Cluster Computing and Grid 2008, Lyon, May 2008, pp: 185-193.

[15]. *" Basic Features of the Grid Component Model (assessed)"*, CoreGRID Deliverable D.PM.04, 2007.

[16].D. Caromel, W. Klause, J. Vayssiere, *"Towards seamless computing and metacomputing in java"*, Concurrency Practice and Experience, vol. 10, no. 11–13, 1998, pp. 1043–1061.

[17].JBoss AOP, http://www.jboss.org/jbossaop/.

[18].TSpaces, http://www.almaden.ibm.com/cs/TSpaces/