MODEL FOR A GRID INFRASTRUCUTRE BASED ON JAVA

Radu Adrian NICULIȚĂ[1], Nicolae ȚĂPUȘ[2]

În această lucrare este prezentat un model de infrastrucutra GRID bazat pe avantajele pe care le prezintă utilizarea Java în Grid computing. Modelul se află la baza dezvoltării unui middleware care poate crea un sistem Grid bazat pe o paradigmă hibridă. Modelul este construit pornind de la o arhitectura pe trei nivele. Ideea fundamentala este incercarea de a utiliza puterea de calcul nefolosită a fiecareuia dintre calculatoarele noaste, anume a tuturor PC-urilor din întreaga lume care sunt conectate la internet.

This paper presents a model for a GRID infrastructure based on the advantages of using Java in the field of GRID computing. The model is the foundation of the development and implementation of a middleware which can create a GRID system based on a hybrid paradigm. The model is build starting from three tier architecture. The basic idea is to benefit from the unused computing power of each one of our computers, namely of all the PCs connected to the Internet in the world.

Keywords: Grid computing, middleware, Java

1. Introduction

The scientific community around the field of computational research has been long in the vanguard of advanced computing, because of the persistent need to resolve problems that require resources that cannot be supplied even by the most powerful computers available. Examples of such applications can be found in many areas, from the financial modelling and simulations of vehicles up to computational genetics and weather. For years, these considerations have led researchers to conclude that a more aggressive should be used and have given rise to innovative ideas, such as vector computers, parallel systems, clusters and other innovative computational technologies.

In recent years, the availability on a large scale of a high-performance high-speed network infrastructure, as well as the increasing awareness of the existence of new ways of solving problems, has made it possible to use these networks for coupling systems and resources distributed over a large geographic area and have stimulated interest in so-called Grid computing. The term "Grid" refers to a type of infrastructure with an increasing presence in several scenes,

[1] PhD. student, Universitatea POLITEHNICA of Bucharest, Romania
[2] Professor, Universitatea POLITEHNICA of Bucharest, Romania, e-mail:ntapus@cs.pub.ro

which offers security, access to resources and information and other services that make possible the sharing and management of distributed resources in a controlled and coordinated fashion. The collaboration takes place between the so-called "virtual organizations" made up dynamically of individuals or organizations pursuing a common interest. At the same time, there are a large number of ambitious projects that apply concepts of Grid computing to solve complicated problems, such as distributed analysis of data from physical access engineers in the field of seismological studies. At the same time we see "scientific portals" being created, which are accessible by small client applications and provides access to collection of information sources and to simulation systems which support a scientific discipline.

At the foundation of the progress of both the Grid computing and parallel calculation in general, there is a common need for coordination and communication mechanisms that can allow multiple use of resources in a concerted manner in order to solve complex problems. Engineering and scientific applications have largely addressed these needs especially using ad hoc approaches, most of them in low-level and specific manners, using message passing libraries inside of parallel computers and various ways of communication between computers in the network.

But, unfortunately, although these low-level approaches have allowed users to achieve their purpose in terms of performance criteria, an unintended consequence has been that much of the scientific community does not benefit much from the remarkable positive developments in terms of the software engineering, developments that were significant in the last period. In particular, the many benefits of Java, which seems to be the ideal medium of communication in a multi paradigm environment, are not exploited at all [1]. Java binary code is platform independent and can be executed in any secure virtual machine from almost any operating system, making Java an attractive environment for Grid computing. In addition, the performance of Java in relation to the implementation of the sequential code translating increased substantially in the last period of time; the development of "Grande" applications is much closer. Moreover, Java provides a framework for sophisticated graphical interfaces, as well as a paradigm for method innovation over remote objects. These features are of great interest in the handling of scientific instruments from a distance.

The rapid development of Java technology now makes it possible to support various structures of communication and coordination necessary for scientific applications in a single object-oriented framework. In the next part of this paper, we will present an approach that can be achieved and paradigms of communication in Java.

2. How Can Grid Computing Benefit From Using Java?

We can say that Java is an environment suitable to be used in a Grid computing environment, as the basic paradigm of such middleware [3]. Some reasons leading to this conclusion are presented in this section.

A GRID infrastructure is fundamentally based on the parallelism of the application it is running. Java programming language includes features beneficial for engineering projects of large scale software, such as packages, object orientation, inheritance, single formats, garbage collection and unified data formats. Since threads of execution and concurrency control mechanisms are part of the language, it is possible to express the parallelism at user level.

The development of GRID applications needs to benefit from pre-existent libraries in order to make the programming of such applications less complex. Java offers a wide variety of libraries containing additional classes including functions essential for Grid computing, such as the opportunity to make a secure transfer of data through sockets or through message passing. On the other hand, Java provides a architecture based on the JavaBeans component hence allowing the development of programs based on components.

Application distribution is always a burden in GRID environments. Java binary code allows easy distribution of software through Web browsers and through automatic installation features.

When we are talking about GRID computing, portability is the key. In addition to the unified data format, binary format Java ensure full portability following the popular concept of "write once, run anywhere". On the other hand, Java implementations exist on many types of devices. The workplace of scientists will be extended at many devices that can run Java, such as PDAs, mobile phones or smart cards.

Performance was always regarded as the major setback of using Java. Recent research results show that the performance of many Java applications may be a lot closer to the C or C + +.

Last but not least, scientific projects sometimes require longevity evaluation of a technology before it can be used. The high level of support that Java enjoys among producers makes this technology actual and very future-proof. Also, universities throughout the world are teaching Java to their students.

3. A Java-based Grid Middleware Model

A distributed system, especially a Grid one, consists of several components that are present in different locations and working together for a common purpose. Each of these components has its own role and responsibilities. They form the Grid middleware and should provide a working environment and performance.

Our intention is to create an architectural model for a middleware which is building on the ideas presented in the above paragraphs. This model is at the heart of developing a middleware that can create a GRID system based on a hybrid paradigm. Thus, there will be some central elements that will facilitate the exchange of messages, which will make the planning and will oversee the entire infrastructure.

But the actual computing power, the machines that will run the computational part, will be developed on a peer-to-peer model. More specifically, just like in a few cases already famous (SETI @ Home [8], Genome @ Home), the basic idea is to develop a system in which everyone that has a computer connected to the Internet can participate, offering both available processor cycles on his computer, and as the beneficiary, being able to send their applications to run.

3.1. The Architecture of the Model

Three-tier architecture

In all GRID environments, we are dealing with three entities that take part in the system: the consumer that submits application, the central part of the system receives, schedules and handles the application and the producers that are doing the actual computational part.

The model has three basic functions:

- It allows users (developers of Grid applications) to send applications that they want to run into the system;
- It allows users to contribute to the computational resources with their own computers;
- It manages the available resources, as well as their entry and exit from the system.

The proposed model is based on architecture with three layers, which consist of:

- *Consumers*, who send applications into the system;
- *The resource broker and the system core*, which deals with the management of the resources and scheduling, with a central view over the entire system;
- *Producers*, which execute the code sent into the system.

The structure of the entire application is based on the existence of a blocking message passing mechanism over the network between machines that are part of the system [2]. This implies that one of the parties can wait for an incoming message without consuming CPU time and can be notified of the arrival of such a message by the messaging system. Building on the variety of existing Java libraries (born due to the intrinsic portable nature of Java), a programmer can choose between several such libraries available both in the free land, as well as in

the commercial one (e.g. Jini, JavaSpace, GigaSpace etc.). From this point forward we call this central element (library) the *communication system*.

A general view of the system is shown in fig. 1; the functions and purpose of each component will be detailed in the following.

The connection between the components is provided through the existing network, either through a LAN environment type (if the system will run on a cluster) or through the Internet (if the system will run over a vast geographical environment).

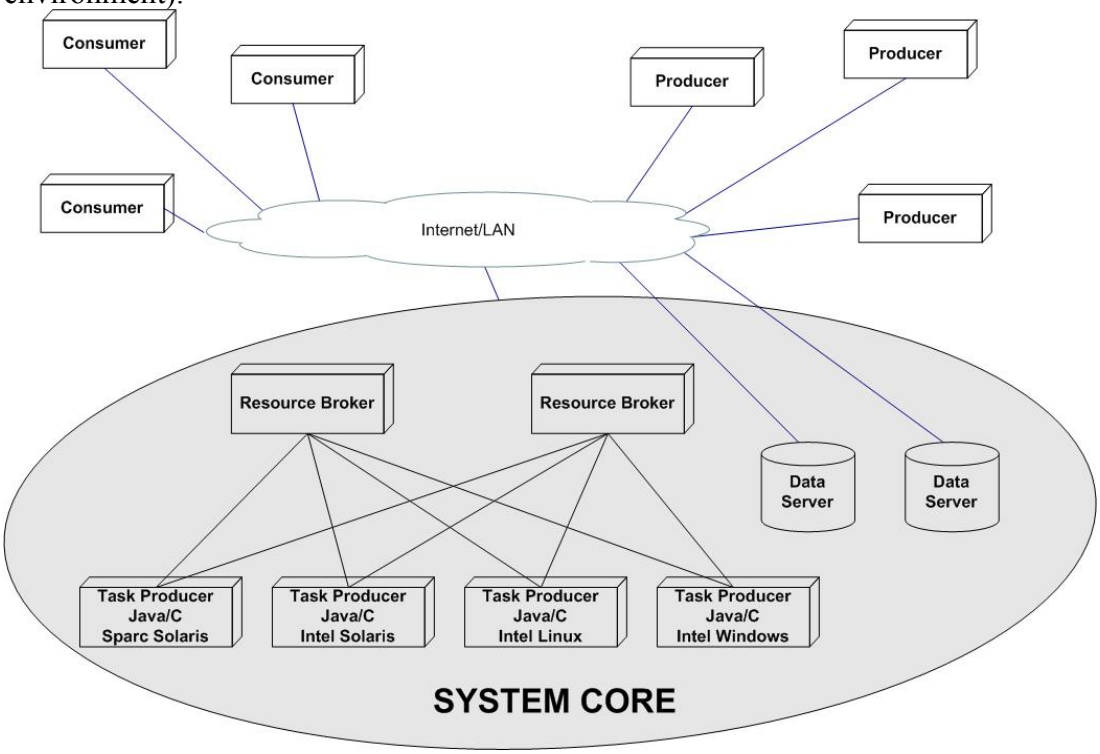

Fig. 1. Components of the model

The components of our middleware are:

The Consumer

The consumer is the one who sends applications in the system. It can be any system connected to the network running the needed components and a graphical interface. Any computer connected to the Internet can play this role, using a Java application that runs for example in a web browser or in a screen saver. The user can send into the system a file containing the applications, using a web interface. File format depends on the programming language used (for Java JAR / ZIP for C, etc.). This application will need to extend some existing methods, one of which is the task producer. This will be subsequently carried on the machine where it will generate tasks, which in turn will then be sent to the producers. The Result Collector will run on the same machine with the consumer.

This component will receive the results generated by the task execution on producer machines.

At the consumer layer one will also be able to submit new modules into the system, such as new communication protocols. These modules will also need to extend on existing components.

The System Core

The System Core is the centre of the model and it contains three type of elements:

The Resource Broker

The Resource Broker is the core of the system. At the most general level, it is only intended to do planning. This is necessary for several reasons. The first and most obvious is the need to control the presence and usage of the resources, especially taking into account that we are dealing with multiple applications running simultaneously in the same system.

Also, there are some objective needs deriving from the need to run non-Java applications on our infrastructure. Taking into account that a middleware, even created in Java, should allow to run code written in other programming languages, the resource manager must keep a list of resources processing that can run native code. This is necessary taking into account that, unlike Java, most programming languages produce platform-dependent code or even library-dependent code. The scheduler must choose the right platform for the producer who will run the sent code. At the resource broker's level, there are two types of scheduling to be done: application scheduling (applications that are sent by the user) and task scheduling (the computational components that each provides one part of the final result).

Even though there are many approaches related to planning in distributed systems, our model here focuses on a centralized scheduler (however planning can be influenced by performing pattern-matching at consumer level when objects are restored via the communication system).

The Task Producer

The task producer is a machine which is part of the central elements in our system; it is part of the "master" component of the hybrid paradigm used. This component generates tasks, which are then planned by the resource broker. An application presented to the system contains an entry point that will subsequently start to generate, at the runtime, computational tasks in the form of object code, which then will be serialized, packed and sent to the producers. They will perform the task, and will return the results. The model is exactly the one in the divide and Imperia algorithm; a big problem is divided into sub-problems that in this case will be executed on other machines.

Taking into account that we should support running non-Java code, the task producer will need to be run on the same type of platform as the producer

who will run the tasks written in non-Java code (the actual code that will be run by producers is generated at the task producer level). Thus, each task producer will run either Java coded applications (which can run on any platform) or code compiled for the platform-specific task producer.

The Data Server

The data server is a machine dedicated to file storage. Any data file used by any application developed on top of our middleware can be sent to the data server. From inside any task, the programmer can obtain access to a data file specific to the applications that generated the task. With reference they obtained, the task may write / read chunks of the file or even the entire file.

The Producer

The producer is a machine that has volunteered part of its processing power to run the Grid applications. It will receive task objects from the system in the form of serialized objects; they will be dynamically loaded than started. The result obtained for each task will then be sent to the consumer that submitted the original application.

Producers and consumers can share the same physical machine, which is the usual case in a wide geographical environment. A totally different approach is usual in a cluster or campus type work environment, where there could be computers that run only producers, as computing power, and a few that will run the consumer to send applications to process.

3.2. The Communication Inside the Model

In this section we will present the algorithm that will be the foundation for the application that will run on our system. All the steps that an application will follow are presented in Fig. 2, following the references that are placed inside the communication system. Many additional messages existing in the lifecycle of the system will not be covered here as we will concentrate on the basic functionality of the system.

To better understand how everything should work, let's take a look at the general algorithm that an application will follow with respect to object transfer through the network.

1. A user sends a new application in the system from a consumer machine in the form of a JAR archive containing all the necessary classes, including at least one task generator class and a results collector class; for a functional application, we will also need at least one task class and one result class;
2. The result collector is dynamically loaded, instantiated and started on the consumer machine;
3. A new reference signalling the presence of a new application to be ran is placed into the communication system;
4. The referenced placed at point 3 is retrieved by a resource broker that calls the scheduling function which will in turn choose the task producer machine that

will run the application. A new reference destined to that machine which contains a link to the new application is created; this reference will indicate the application archive that is waiting to be retrieved on the consumer machine that initially sent the application;

5. The chosen task producer retrieves the reference through for it from the communication system and it downloads the archive file that contains the application, from the consumer machine;

6. The task producer dynamically loads the task generator class of the application and all the other necessary classes, it instantiates them and runs the entry point in the application. This action will start the task generation process;

7. Each time a new task is generated and sent into the system this is first serialized and added to a JAR archive that will also contain all the classes that are referred by that object (including the result class). A new reference to that task is then placed into the communication system, without a clear destination;

8. The resource broker takes the reference and calls the scheduler to choose a producer that should run the task that is referred by the reference; this producer can be changed later without affecting the reference. A new reference is placed into the communication system, pointing to the file that contains the serialized tasks on the task producer machine; this reference will be addressed to the chosen producer;

9. The producer that was chosen at stem 8 takes the reference from the communication system, downloads the file from the task producer and then dynamically loads the object and all the other necessary classes. The new instance is ran through a pre-established entry point;

10. The execution of the task will return an object which is the result of this task. The result is serialized and placed in a JAR archive and a new object reference is created, pointing to that file and having the destination set according to the result delivery mode chosen when the application was first submitted. The result will be sent to the resource broker that scheduled the application if the results should be handled through the resource broker, or directly to the result collector of the application, running at the consumer, if the result delivery mode is direct delivery;

11. If the result delivery mode is direct delivery, the result manager running at the consumer that submitted the application takes the reference to the result from the communication system, downloads the file containing the result and dynamically loads the result object, storing it locally in memory. When the result collector requests a new result, this object is returned to it.

12. If the results are sent back through the result broker:

a. The resource broker takes the reference from the communication system and downloads the file containing the serialized result pointed by the reference and stores it on the secondary storage.

b. The result collector running at the consumer decides to check for results and sends a message to the resource broker asking for the number of results that are stored there for the application that this result collector runs for. If there are some results ready, the result collector can get them one by one. In order to get a new result, the result collector puts a request for it in the communication system;

c. The result manager running on the resource broker's machine takes the request (in form of a message) from the communication system and finds a file containing a serialized result that belongs to the respective application. A new object reference is created pointing to that file located on the resource broker's machine, with the destination the consumer machine that runs the result collector and is written in the communication system;

d. The result manager running at the consumer that submitted the application takes the reference to the result from the communication system downloads the file containing the result and dynamically loads the result object, storing it locally in memory. When the result collector requests a new result, this object is returned to it.

13. When the application's result collector returns, all the references that are related to the application that has just terminated are removed from the communication system.

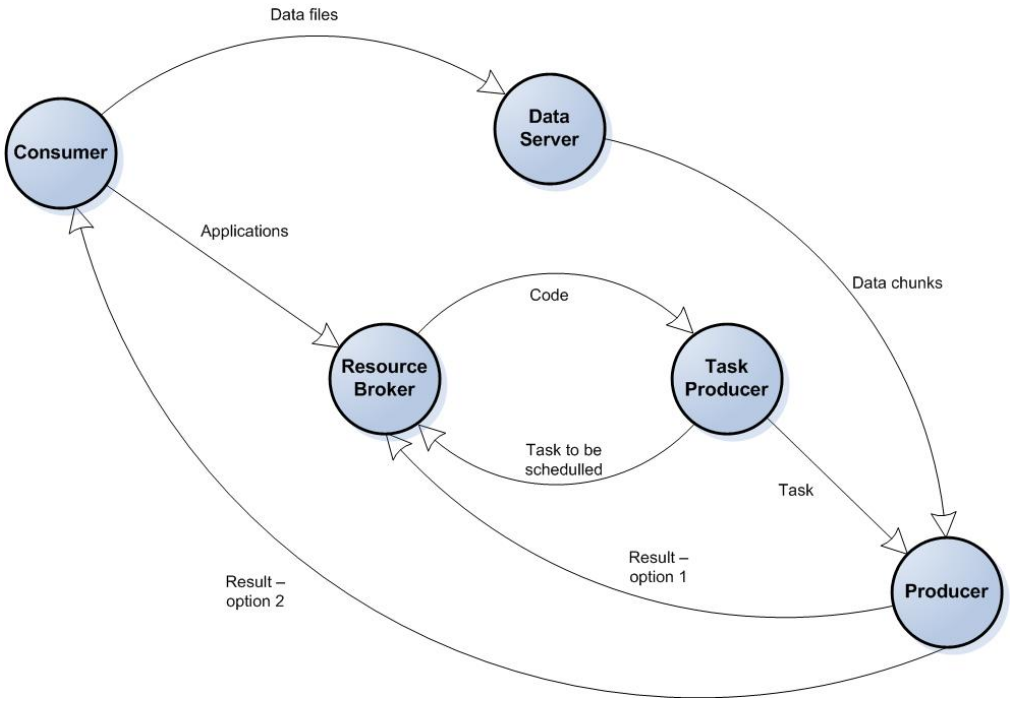

Fig. 2. Data and information transfer inside the proposed architecture

4. The Life of an Application in the System

In this paragraph we will present the live cycle of an applications in the system, from the launch until the getting of the final result.

For starters, it is necessary that the application being developed is one that is specific for our model; more exactly it needs to be constituted of several components. Those components consist of an implementation of a task producer class, which needs to return objects of type task. In turn, those constitute the computational intensive and parallel part of the application, who will return results in the form of objects that extend a predefined class.

The first step is that the developer or the user of the application to send the application into the system in the form of an archive through a GUI interfaces. At the interface level, the system is predefined through the indication of the IP address of the resource broker, the core of the entire infrastructure. Applications will be sent to the resource broker, who will download the archive in the form of a file. From this point, the client applications may choose to remain on-line and to expect the results as they are generated, or to return from time to time to reap results that are available. The first approach is appropriate in an academic environment or in a cluster type infrastructure, in which computers work together to reach a result as soon as possible, and the machine that sent the application has the sole purpose of finding the final result. By contrast, the second approach is more appropriate in a widely distributed environment, such as the Internet, where the search for the result is a very long term work and it is not desirable to the expected results on-line (in applications such as the SETI or Human Genome search).

After downloading the file containing the applications, the resource broker will begin searching for a task producer who can meet the requirements related to the platform and operating system on which to run that specific application. These limitations exist when the application is depending on the platform (in the case it is not developed in Java). In case the application is developed in Java, it can run on any platform.

We should mention here that the task producer, as well as the producers, are applications that run on distant machines, either in the form of a process in the foreground (i.e. applications running) or in the form of a background process (a daemon) or even in the form a screen saver. These applications will sign themselves to the resource broker in order to start; one of the crucial functions of the resource broker is to keep track of all producers and task producers that exist in the system. The resource broker will also check their availability and will mark as such in case they do not respond.

After finding an appropriate available task producer, the resource broker will send an instance of the task producer class to that specific machine. The task

producer will get this object and will run a certain method which is the entry point. As a result of running this method will be obtained a multitude of instances of task classes. All of these instances shall be submitted in live form (instantiated), by serialization, to resource broker.

The resource broker will run the scheduler for each one of the received tasks for the respective applications. Each such task will be planned to a particular available producer, according to the planning algorithm.

Producers will run each task object and will get in the end, after performing the computational part, a result type object. This item will be serialized and then sent to the customer that sent the initial application (for the online collection of results) or to the resource broker; later, the results available until some point will be taken by the customer, on the case of off line result handling.

All these operations will be executed until all the tasks generated by the task producer class of a certain application will have been returned a result and until all these results will reach the client who sent the application. Thereafter, the resource broker will consider the application as being finished and it will free up all the resources allocated to it.

5. Conclusions

Today, the Grid systems focus on effectiveness and on the existence of scalable powerful middleware that can allow running the applications with optimal performance [15]. However there is a possible unexplored side of what can be and can become Grid computing; this side is comprise of the enormous computing power that sits unused on the computers of each of us, on all the PCs in the world, which, on the other hand, are increasingly frequently connected to the Internet. However, in order to benefit from this power, there is a need for a middleware that is, on the one hand, very simple to install and use, and on the other hand totally independent of platform, operating system or installed libraries on a personal computer. Computers that are part of the grid managed in academic environments have a characteristic that is not valid for PCs of individual from the Internet, namely that they benefit the possibility that a specialized administrator installs and configures the system in such a way that it can run a dedicated Grid middleware. The usual Internet user does not have enough knowledge to carry out such at task, nor will he or she install and use a system that is complicated to use. Aspects such as compiling a library are totally out of the reach of this user category. Resolving these issues can be done easily through the use of the most widely known and world spread portable platform that everyone has heard about: Java. Building on a portable language that has become a legend can lead to results that can bring Grid into anyone’s house. What would this mean? That anyone

could provide their unused computing power of its PC and in turn be able to turn such applications; users could be able to be part of the millions of heroes who solved an extremely complex problem for which nobody could ever find a way of financing.

REFERENCES

[1] *Vladimir Getov, Gregor von Laszewski, Michael Philippsen, Ian Foster*, Multiparadigm Communications în Java for Grid Computing, Communications of the ACM,Volume 44, Issue 10, Pages: 118 – 125, 2001

[2] *W. Gropp, E. Lusk, A. Skjellum*, Using MPI: Portable Parallel Programming with the Message Passing Interface, The MIT Press, 1999

[3] *M. Philippsen, B. Haumacher, C. Nester*, *More* efficient serialization and RMI for Java. Concurrency: Concurrency: Practice and Experience, 12(7):495–518, 2000

[4] *Johan Prawira, ALiCE,* Java-based Grid Computing System, Honours Thesis, School of Computing, National University of Singapore, 2002

[5] *Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane*, *A* Java Commodity Grid Kit, Concurrency and Computation: Practice and Experience, 13(89):643- 662, 2001. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf.

[6] *J. Frey, I. Foster, M. Livny, T. Tannenbaum, S. Tuecke, G. Condor,* A Computation Management Agent for Multi-Institutional Grids, University of Wisconsin Madison, 2001

[7] *L.F.G. Sarmenta,* Volunteer Computing. Ph.D. thesis. Department of Electrical Engineering and Computer Science, MIT, March, 2001

[8] SETI@home: http://setiathome.ssl.berkeley.edu

[9] Distributed.Net: http://www.distributed.net

[10] Globus: http://www.globus.org

[11] The GLOBE Project: http://www.cs.vu.nl/~steen/globe/

[12] IEEE High Performance Distributed Computing (HPDC) symposium, 2008: http://www-2.cs.cmu.edu/~hpdc

[13] High Performance Computing Symposium, 2002 http://wwwteo.informatik.unirostock.de/HPC

[14] 9rd International Workshop on Grid Computing: http://www.gridcomputing.org/grid, 2008

[15] *Sang Boem Lim, Hanku Lee, Bryan Carpenter, Geoffrey Fox*, Runtime support for scalable programming in Java, The Journal of Supercomputing archive Volume 4, Issue 2, February 2008.