Thilo Kielmann, Vrije Universiteit
Editor

14-11-03

Workshop on Grid Applications and Programming Tools
Held in conjunction with GGF8, June 25 2003, Seattle, USA
Proceedings

Status of This Memo

This memo provides information to the Grid user community. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This is the proceedings of the Workshop on Grid Applications and Programming Tools that has been organized jointly by the Applications and Testbeds Research Group (APPS-RG) and the User Program Development Tools Research Group (UPDT-RG) of the GGF. It contains the papers that have been accepted for presentation by the programme committee.

Contents

1. Foreword

The Applications and Testbeds Research Group (APPS-RG) seeks to facilitate the use of Grid technology by application developers, and to attract new application domains to the Grid. The User Program Development Tools for the Grid Research Group (UPDT-RG) seeks to simplify the process of programming on the Grid by facilitating the development and deployment of Grid-enabled tools such as debuggers and performance tuning tools.Recent experience shows that Grid users run their applications using various kinds of additional (and frequently tailor-made) tools, ranging from simple wrappers around Globus commands up to web-based application portals.

The goals of this workshop were

- to provide a forum for (prospective) applications utilizing Grids and to disseminate "lessons learned" from Grid-enabling application codes
- to spread information about tools, toolkits and other instruments for users and application programmers
- to encourage users and application programmers to make use of existing Grid infrastructure

We were looking for presentations on the workshop's subject. Suggested topics for talks were the following:

1. Experience in converting an existing application to run on the Grid
2. Experience in designing a new application to run on the Grid
3. Experience in using a Grid-enabled application
4. Experience in using or creating tools that support the development or use of Grid applications
5. Thoughts on what the Grid should provide to effectively support applications
6. Thoughts and/or experience on the characteristics of applications or particular application software that make them well suited for the Grid, or make them ill-suited to run on the Grid.

Despite the short time frame that was available to announce the workshop, the response on submitted papers was very positive. From the submissions, the programme committee selected the 11 papers included in this report. The organizers were very pleased by the interesting presentations and the discussions they spawned off . We gratefully accepted the offer to publish extended versions of the strongest papers in an upcoming special issue of the *Journal of Grid Computing*.

2. Security Considerations

There may be security issues related to the individual solutions presented at the workshop. These need to be considered on a per-paper basis.

Author Information

The workshop has been organized by the chair persons of the two GGF research groups which were involved:

- Thomas Hinke, NASA Ames, USA, *Thomas.H.Hinke@nasa.gov*
- Thilo Kielmann, Vrije Universiteit, The Netherlands, *kielmann@cs.vu.nl*
- Ed Seidel, AEI – MPG, Germany, and LSU, USA, *eseidel@capital.lsu.edu*
- Susanne Balle, HP, USA, *susanne.balle@hp.com*
- Robert T. Hood, NASA Ames, USA, *rhood@nas.nasa.gov*

The authors of the individual papers can be reached according to the information given in the respective papers.

Intellectual Property Statement

Full Copyright Notice

Programme Committee

The following people kindly agreed to review the submitted papers:

Gabrielle Allen	(AEI - MPG)
Susanne Balle	(HP)
Simon J. Cox	(University of Southampton)
Thomas Hinke	(NASA Ames)
Robert T. Hood	(NASA Ames)
Shantenu Jha	(University College London)
Thilo Kielmann	(Vrije Universiteit)
Andre Merzky	(Zuse Institute Berlin)
Matthias Müller	(HLRS Stuttgart)
Ed Seidel	(AEI - MPG)
Yoshio Tanaka	(AIST)

Workshop Papers

Development of Grid Applications on Standard Grid Middleware

Hiroshi Takemiya, Kazuyuki Shudo, Yoshio Tanaka, Satoshi Sekiguchi
National Institute of Advanced Industrial Science and Technology

Abstract

The effectiveness of standard grid middleware has been evaluated through the work of “gridifying” a legacy program. As a case study, we have gridified a typical parameter survey program called the barotropic S-model which aims to predict middle- to long-term climate change accurately. Ninf-G was used to gridify the system. By using Ninf-G, the program could be easily gridified without worrying about the complicated structure of the grid itself.

The program has been deployed on 8 clusters on the APGrid Testbed which spreads over various Pan-Pacific countries. Using 100 processors on these clusters, we succeeded in giving a demonstration of a 10 day weather prediction at the CCGrid Conference 2003. The simulation was finished in 400 seconds which is about 20 times shorter than the sequential execution time.

1. Introduction

As a consequence of the pervasive permeation of the grid concept, many researchers in the field of natural science as well as computer science have started to pay attention to the concept, and expect it to solve problems which are too large or too complicated to process on a single computer. Although many kinds of grid middleware have been proposed to support the construction of grids and grid applications, there is little information on how to “gridify” legacy programs using this middleware. “How is the performance of gridified programs?”, “what should be done for attaining good performance?”, and so on. As a result, there is still a high barrier for application programmers wanting to construct a grid application.

In order to cross the barrier and accelerate the construction of grid applications, we have examined and evaluated the effectiveness of standard grid middleware such as Ninf-G[1] and the Globus Toolkit[2] through the work for gridifying “real” applications and executing them on the international Grid testbed. The Ninf-G library is a reference implementation of GridRPC[3] which was proposed to the GGF as a standard RPC protocol on a grid environment. In addition, Ninf-G is implemented on the Globus Toolkit. Globus is one of the most popular grid middleware suites and is installed on various grid test beds. It’s, however, too difficult for non-computer scientists to gridify their applications using Globus directly. Ninf-G provides familiar semantics, similar to those of RPC, so that the complicated mechanisms of the underlying grid infrastructure can be hidden behind the RPC interface.

Using Ninf-G, we have gridified a legacy FORTRAN program called the Barotropic S-model which is designed to predict middle- to long-term climate change. This program is expected to attain good performance on a grid environment, because it executes hundreds to thousands of independent climate simulations.

In this paper, we describe the process of gridifying a typical parameter survey program using Ninf-

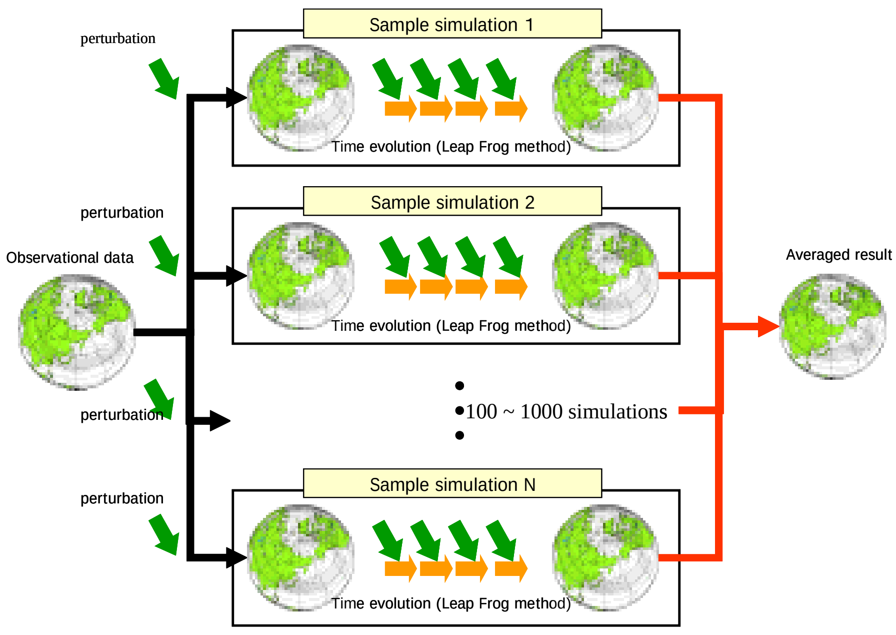


Fig.1 Schematic execution flow of a climate simulation

G, the techniques for attaining good performance, and lessons learned from the construction of a demonstration system on a grid test bed. Such information is invaluable for application programmers in constructing their own grid applications. At the same time, the information helps grid middleware developers to improve their middleware.

First, we will give an overview of the Barotropic S-model program and the Ninf-G library. In chapter 4, we will explain how to gridify the legacy program using Ninf-G and evaluate the ease of use of the tool. Chapter 5 is devoted to describing the demonstration system developed on the ApGrid Testbed and summarizing the lessons learned from the demonstration performed at the CCGrid Conference.

2. Climate simulation program

Our climate simulation system is designed to predict middle- to long-term, global climate change such as the wind patterns of the Jet-Stream and blocking phenomena of high atmospheric pressure.

The core part of the system is based on the Barotropic S-model which was originally proposed by Tanaka[4].

It is well known that long term weather prediction is very difficult due to the chaotic nature of the system. The S-model program tries to extend the predictability by adopting two methods.

The first is to solve a spectral equation on vertically averaged quantities, the averaged barotropic component of the atmosphere, in place of solving full, nonlinear, fluid equations. It was pointed out that the averaged barotropic component is only weakly influenced by the chaotic nature of the system[4]. By solving averaged equations for barotropic components, the model has succeeded in reproducing the blocking phenomenon of high atmospheric pressure that took place in Alaska at the beginning of 1989[4].

Another method is to take a statistical ensemble mean of the simulation results. This method is expected to suppress the growth of errors included in the initial observational data and those generated during the simulation. The program executes hundreds to thousands of sample simulations while introducing perturbation for each simulation. The result of each sample simulation is gathered and included in an average to generate the final statistical result. The execution flow of the system is shown in Figure 1.

From the point of view of grid computing, this program is quite suitable for grid computing, because it is a typical parameter survey. So if we gridify the program, we can expect a quick response from the system.

The system consists of the following 4 parts.

(1) Initialization part

The program reads initial spectral data generated from the observational data, and sets simulation parameters such as a simulation period and the number of sample simulations.

(2) Simulation part

Sample simulations are executed using perturbed initial spectral data.

(3) Statistical processing part

The result of each simulation is gathered and averaged to generate the ensemble data.

(4) Visualization part

All the simulation results are visualized to generate the contour maps of physical quantities in the northern hemisphere.

3. Ninf-G library

Ninf-G[1] is a reference implementation of the GridRPC API. GridRPC is a programming model based on client-server-type remote procedure calls on the Grid. Client programs can “call” server programs on remote resources using the client API provided by GridRPC systems. The GridRPC Working Group of the GGF has just been approved and will make efforts to draft the standardization of the GridRPC API[2].

Implemented on top of the Globus Toolkit. Ninf-G provides user-level API specifications that retain the simplicity, but generalize call contexts, for more flexible Grid-level programming, as well as high inter-operability with other Globus-based Grid systems.

Ninf-G provides a Client Component and Remote Executables. The Client Component consists of a client API and GridRPC client libraries for RPC invocations. The client API provides GridRPC call functions for RPC invocations. The GridRPC call functions may be categorized into blocking (synchronous) calls and non-blocking (asynchronous) calls. Asynchronous calls enable the implementation of task-parallel applications. On the other hand, synchronous calls are useful to execute compute-intensive library items such as matrix solvers on a high performance computing resource.

A Remote Executable is an entity of a callee routine, that is, it consists of a stub main routine and system-supported wrapper functions. The stub is automatically generated by the Ninf compiler from an IDL file in which the interface of a remote executable is described. It should be noted that no IDL handling is necessary on the client side, as opposed to traditional RPC systems such as CORBA. The client program dynamically gets the interface information from a network directory service in place of using a statically generated client stub. This mechanism is useful for utilizing remote executables

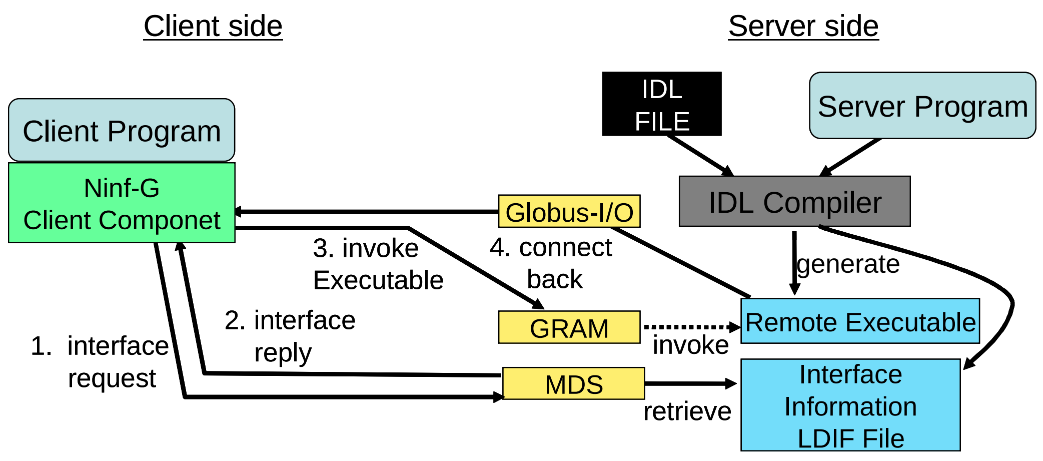


Fig.2 Software architecture of Ninf-G

developed by other providers. In this case, a user can realize an RPC call merely by inserting a Ninf-G function, without worrying about the stub information.

As for the execution of a gridified program using Ninf-G, the client component and the remote executable communicate with each other using the functions of the Globus Toolkit. The software architecture of Ninf-G is depicted in Figure 2. When the client program is executed, the client component accesses MDS to get interface information for a server stub which includes two kinds of information; that for invoking a remote executable such as path information and that for encoding/decoding arguments such as type and size of arguments. Next, the client passes the former information to GRAM and asks to invoke a server program. In this phase, authentication is performed using GSI. After the invocation, the remote executable connects back to a client to establish a connection. Finally, the client component dynamically encodes the arguments according to the interface information, and transfers them using Globus I/O and GASS.

Although a client and a server program have to deal with the complicated mechanism of the Globus Toolkit, Ninf-G can completely hide the mechanism from a user. Gridifying a legacy program using Ninf-g, therefore, requires no detailed knowledge on the Grid infrastructure. The feature can greatly help non-computer scientists to gridify their own applications.

4. Gridification of a program using Ninf-G

4.1 Gridification of a legacy program

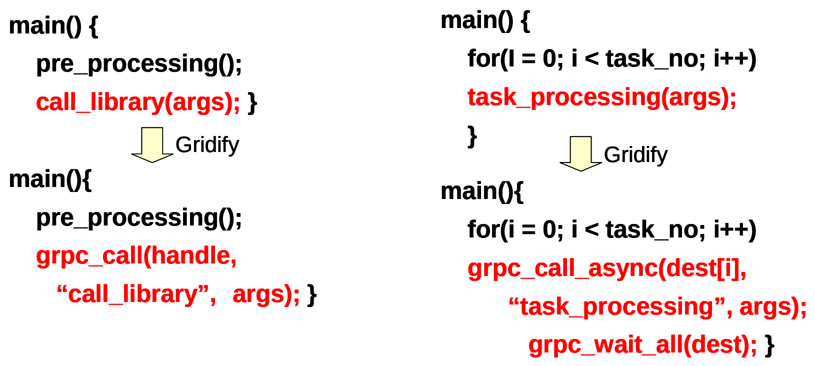
In general, a user can gridify a sequential program using Ninf-G by following the four steps below.

(1) Specifying the interface of a server program

a user should identify a client part and a server part in the original program. If a server part is not specified as a function in the original program, it should be modified to be a function. Because the server part will be executed on the remote site, it is desirable that the granularity of the server part is large and data transferred between a client and a server part is small..

(2) Eliminating data dependence between a client and a server part, or among server parts

The main cause of data dependence between a client and a server part is variables shared by both parts. Candidates of such variables are common variables in FORTRAN program and global variables



```
main() {
   pre_processing();
   call_library(args); }
main(){
   pre_processing();
   grpc_call(handle,
     “call_library”,  args); }
```

```
main() {
   for(I = 0; i < task_no; i++)
   task_processing(args);
   }
main(){
   for(i = 0; i < task_no; i++)
   grpc_call_async(dest[i],
        “task_processing”, args);
          grpc_wait_all(dest); }
```

Fig.3. Typical examples of gridifying a legacy program
(left:using synchronous calls, Right: using asynchronous calls)

in C program. These variables should be passed as arguments to a server part.

If data dependence among server programs exists, server parts must be executed sequentially. By eliminating the dependence, the performance will be drastically increased, because many tasks can be simultaneously executed by using asynchronous calls.

After checking the program modified in the step (1) and (2) works well on a single computer, a user should extract the client part and the server part from the modified program, and tailor them as a client component and a remote executable respectively.

(3) Inserting Ninf-G functions into the client program

The client component is created by inserting Ninf-G functions into the client program. Ninf-G has functions for initialization (grpc_initialize, grpc_function_handle_init) and termination (grpc_function_handle_destruct, grpc_finalize) as well as asynchronous and synchronous calls. RPC can be realized by inserting these functions. Typical examples of inserting Ninf-G functions are shown in Figure 3.

(4) Creating remote executables from a server program and a server stub

Remote executable is created on each server in the last step. This step can be divided into the following sub-steps. First, a user describes the interface for the server program with Ninf IDL (Interface Description Language). The created IDL file is used to generate a server stub. This work is automatically performed by a Ninf compiler. Finally, the user should register the information in MDS.

It should be noted that steps (1) and (2) can be performed on a single computer. Only steps (3) and (4) must be executed in a grid environment.

In addition to the work described above, it is a good idea to optimize the client program to attain good performance. There are two methods for optimization.

(1) Implementing a dynamic task scheduling method

The original parameter survey program usually calculates independent tasks by using a loop structure. The easiest way of gridifying the program using Ninf-G is to allocate these tasks cyclically to each server, one by one. Unfortunately, it is difficult to attain good performance using such static scheduling methods, because the grid environment is usually quite heterogeneous and dynamic. For example, computing resources are different and their loads vary from moment to moment. In order to adapt the gridified program to such an environment, a dynamic scheduling routine will be required.

(2) Multi-threading a client program

The Ninf-G library accesses servers and a directory service over a network. The cost for these access operations becomes very expensive under a large latency and low bandwidth network environment. One solution for reducing the cost is to parallelize the client program. Then, the client program can execute other tasks during the long and slow Ninf-G call.

4.2 Gridification of the climate simulation program

According to the method described in the previous section, we have gridified our climate simulation program as follows.

(1) Specifying the interface of a server program

As described in chapter 2, the original program consists of 4 parts. Among them, we have gridified a visualization part as well as a simulation part, because both parts have numbers of independent tasks, that is, sample simulations and visualizations of each result. In the original program, the visualization part was implemented to start after the completion of all sample simulations. In order to improve the performance, we have modified the original program so that both parts can be executed in parallel and process the data in a pipelined fashion.

(2) Eliminating data dependence between a client and a server part, or among server parts

As the simulation part had common variables shared between a client and a server part, we specified them as arguments of the remote executables. In addition, the simulation part itself had data dependence in the random number generator. As described in chapter 2, the original program uses random numbers to add perturbations. Sample simulations use random numbers successively, namely, each sample simulation generates a new random number using the seed generated by the previous sample simulation. In order to eliminate the data dependence, we have implemented the Leap Flog algorithm[5] which is one of methods for parallel random number generation.

The visualization part calls many visualization libraries which generates files with a fixed name. When this part will be executed in parallel on a cluster which has a shared file system, the visualized results will be mingled with each other. In order to avoid the contamination of the data, we implemented a routine which generates the unique temporal directory and stores data files in it.

(3) Inserting Ninf-G functions into the client program

```
Module S-model;
Define servmain(IN int N1, IN int IBUF[N1],
                IN int N2, IN double DBUF[N2],
                IN int N3, IN double WW[N3],
                IN int N4, OUT double TARRAY[N4],
                IN int N5, OUT double WTOT[N5])
Required  “S-model_serv.o”
Calls ”Fortran”  servmain(N1, IBUF, N2, DBUF, N3, WW, N4, TARRAY, N5, WTOT);
FortranFormat “%s_”;
```

Fig.4 IDL file for executing a sample simulation

We used an asynchronous function (grpc_call_async()) in both parts, so that tasks were able to be executed simultaneously.

(4) Creating remote executables from a server program and a server stub

In order to generate a server stub, we specified interface information for both parts using the IDL. The interface specified for a simulation part is shown in Figure 4. The simulation part had originally many scalar parameters. In order to reduce the number of arguments, we have put them into array variables, because the data transfer time of Ninf-G is proportional to the number of arguments.

The visualization part requires input and output files. In order to transfer these files, we used the file handling function of Ninf-G which automatically sends a file to the remote executables. The function can be used by just defining a file as an argument with the type “IN filename” in the IDL file. The transfer to the opposite direction takes place by specifying the type as “OUT filename”.

In addition to the above mandatory steps, we optimized the program for better performance.

(1) Implementing a dynamic task scheduling method

We implemented a pure self scheduling mechanism which allocates a single task dynamically to an idle processor.

(2) Multi-threading a client program

Both the simulation and the visualization program are multi-threaded based on the master-worker model. Each program has a single master thread which puts all tasks in a task pool. Each worker thread manages a connection to the server node and allocates a task to the node.

4.3 Cost evaluation of gridifying the climate simulation system

In gridifying the climate simulation system, we modified and newly generated the code of totally about 300 lines. It took 13 days for one person to accomplish the work. 10 days were spent for the work on a single computer, while only 3 days for the work in a grid environment. The reason why the working time spent in a grid environment is shorter than that in the sequential environment is as follows;

(1) Inserting the Ninf-G library was intuitive and straightforward. In addition, the typical parameter survey program could be programmed by inserting only a small amount of Ninf-G calls. In fact, we could accomplish the work by coding only 10 lines for step (3) and 50 lines for step (4).

(2) The IDL compiler automatically generated a server stub. As a result, we could concentrate our attention on the modification of the client part.

(3) Input and output files for the visualization part could be transferred using the file handling mechanism of Ninf-G. We, therefore, did not need to write any code for the file transfer.

(4) The most time consuming work was the analysis of common variables and multi-threading the client program both of which could be performed on a single computer.

Much more efforts would be required if the program were gridified using other tools such as the Globus Toolkit and MPICH-G2[7]. For example, when gridifying a program using MPI, a user has to insert the MPI library calls in both a client and a server part. The work raises the possibility of creating bugs in both programs. Finding bugs in a grid environment is incredibly difficult. In addition, after fixing the bugs, a user has to spend much time to distribute the program over a grid environment.

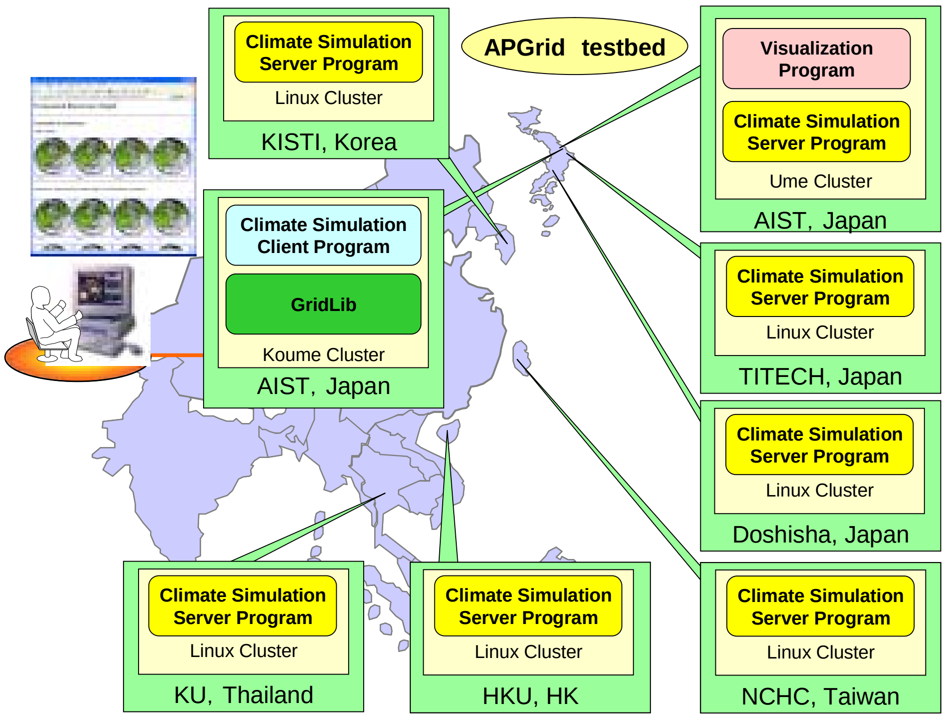

APGrid testbed
Climate Simulation Server Program
Linux Cluster
KISTI, Korea
Climate Simulation Client Program
GridLib
Koume Cluster
AIST, Japan
Visualization Program
Climate Simulation Server Program
Ume Cluster
AIST, Japan
Climate Simulation Server Program
Linux Cluster
TITECH, Japan
Climate Simulation Server Program
Linux Cluster
Doshisha, Japan
Climate Simulation Server Program
Linux Cluster
KU, Thailand
Climate Simulation Server Program
Linux Cluster
HKU, HK
Climate Simulation Server Program
Linux Cluster
NCHC, Taiwan

Fig.5 Configuration of the climate simulation system on the ApGrid Testbed

On the other hand, in the case of using Ninf-G, a user need not modify a server program because of the automatic stub generation function in Ninf-G. In addition, creating a client component from the original program is intuitive and the amount of coding is less than that would be required to gridify the program using the MPI library. These features helps to reduce the number of bugs. And furthermore, finding bugs and fixing a program is not so difficult, because most of bugs are restricted in a client program. As a result, a user can spend little time for the work in a grid environment.

5. Demonstration at the CCGrid conference

The system was demonstrated at the CCGrid 2003 Conference held in Tokyo. We used computing resources on the ApGrid for the demonstration.

ApGrid is a partnership for Grid computing in the Asia Pacific region[6]. It is an open community encouraging collaboration and is not restricted to just a few developed countries. As of the end of March 2003, 41 organizations from 15 countries were participating in ApGrid. One of the most impor-

Table 1 Status of the ApGrid Testbed (As of May, 2003)

Organization (country)	No. of CPUs	CA
AIST (Japan)	92	AIST GTRC CA
Doshisha (Japan)	16	Globus CA
TITECH (Japan)	16	TITECH CA
KISTI (Korea)	80	KISTI CA
Kasetsart U. (Thailand)	15	AIST GTRC CA
AIT (Thailand)	6	AIST GTRC CA
KMITNB (Thailand)	11	AIST GTRC CA
Hong Kong U. (Hong Kong)	32	HKU CA
NCHC (Taiwan)	8	NCHC CA
SDSC (USA)	32	NPACI CA
Osaka-U (Japan)	156	Osaka-U CA

tant objectives of ApGrid is to develop an international Grid testbed called the ApGrid Testbed which can be used for the evaluation of the middleware developed in the community, as well as for the execution of real large-scale applications. As of the end of May, 2003, approximately 450 processors in over 11 cluster systems from 6 countries are available on the ApGrid Testbed. Table 1 gives a list of contributing organizations and their countries, the number of CPUs of contributing resources, and the certificate authority which issues the server certificate of each resource.

We deployed both a simulation and a visualization program to 8 clusters on the testbed to construct the demonstration system. As shown in Table 1, these clusters are in different countries and different security policies. In addition, the number of CPUs of each cluster, and data transfer speed of networks are different. We, therefore, had to construct the system on a typical heterogeneous virtual organization.

The client program was installed on a cluster (the Koume cluster) in AIST. The climate simulation program was installed on 8 clusters and executed on them using a total of 80 CPU's. On the other hand, we decided to install the visualization program on another cluster (the Ume cluster) at the same site (AIST) in which the client program was executed, because heavy network traffic was required for the execution of the program. 30 CPUs were used to execute the program.

In order to access the system securely from a remote site, we constructed a portal for the climate simulation on a cluster (the Koume cluster in AIST) using a GridLib portal kit which was developed in our center[8]. The portal invokes the client program by using a globus-run command. The simulation results are retrieved by the portal using a globus-url-copy command, and finally downloaded to a PC in the CCGrid conference room. The overall system configuration is depicted in Figure 5.

We executed a 10 day climate simulation using 200 sample simulations. In this case, the simulation program receives an initial data item of 3.4KB and sends back results totalling 135KB. The visualization program requires the simulation result and returns a visualized data of 125KB. It takes about 20 seconds for each simulation and visualization. Therefore, if all the programs are executed on a single computer, it takes about 8000 seconds for the total simulation. We have succeeded in reducing the elapsed time to about 400 seconds by executing all of the programs on the ApGrid Testbed.

Through the construction of the demonstration system, we have found the following problems which result in a performance degradation.

(1) Long initialization time caused by a long MDS access time

As described above, Ninf-G accesses MDS to get interface information for a server program in the initialization phase. The access time is proportional to the amount of registered interface data. From our experience, the access time amounted to more than 20 seconds, when the amount of interface data increases to a few thousands items. As a result, the total time for getting interface information for all servers exceeds 150 seconds, when we use 8 clusters.

In order to reduce the initialization time, we implemented the function so as to get interface information from a local file. By using this function, the cost for getting interface information is reduced to a few milliseconds.

(2) Long termination time caused by long default polling interval of GRAM

In the termination phase, Ninf-G asks GRAM to check the termination of a server program. GRAM has a polling mechanism to check the program termination. The time it takes to check sometimes becomes unacceptably long, because the polling period is set to 30 second by default. As a result, it sometimes takes more than 10 minutes to check the termination of all programs, when we use 100 nodes.

Although the problem is solved by modifying the polling period in the GRAM source code to a few seconds, this is not a good way to resolve the problem. GRAM should provide an interface for setting the polling period, because the optimum polling period will depend on each application.

(3) Very long invocation time due to the use of a batch system

Most clusters of ApGrid are managed by a batch system such as PBS, SGE, and so on. When the number of available nodes is fewer than the number of server programs we want to invoke, some of programs have to wait to start running in the batch queue. The waiting time sometimes amounts to more than an hour and results in a very long invocation time. In order to avoid long blocking, we are now investigating the possibility of introducing a time-out mechanism in the Ninf-G function.

Another problem in executing the system was caused by the instability of the ApGrid Testbed. We have often experienced that target nodes were down or network connection was broken under the execution of a program. The dynamic task allocation mechanism was very useful to resolve the problem. We could easily enhance a fault tolerance of the system by implementing the mechanism in which the failed task was thrown again in the task pool and assigned to other available nodes. It is important to implement a mechanism to cope with the accidents, because it is usual rather than rare to meet such accidents in executing a program in a grid environment.

6. Summary

In this paper, we have evaluated the effectiveness of standard grid middleware and described a method for gridifying a typical parameter survey program using Ninf-G, techniques for attaining good performance, and lessons learned from the construction of a demonstration system on the ApGrid Testbed.

It was found that a typical parameter survey program could be easily gridified using Ninf-G without worrying about the complicated grid infrastructure. We were able to spend most of the time required to modify the original program on a single computer. The amount of work which had to be performed in a grid environment was very small, only a few days, because inserting the Ninf-G library was straightforward and stub generation was automated by the Ninf compiler.

Using the gridified program, we have succeeded in giving a demonstration of a 10 day climate simulation on the ApGrid Testbed. It took 400 seconds to execute 200 sample simulations on 100 CPUs spread over 8 clusters. We found several problems which might degrade the performance. In order to solve them, it is necessary to improve the function of both Ninf-G and the Globus Toolkit.

Currently, our system is running on totally 9 clusters on the ApGrid Testbed. A cluster in Osaka-U has become newly available. Using 150 processors on these clusters, we have succeeded again in giving a demonstration at the PRAGMA workshop held in Melbourne at the beginning of June this year.

Future work will focus on improving the functions of Ninf-G based on lessons learned from these experiences. We are now designing a new Ninf-G library in which new mechanisms such as time-outing, simultaneous multiple handle creation, and so on, are implemented.

Acknowledgments

We would like to give thanks to Dr. Tanaka in Tsukuba university. He was kind enough to provide his S-model program for our research work.

We are also pleased to acknowledge our ApGrid partners, especially TITECH, Doshisha University, KISTI, NCHC, HKU and KU for allowing us to use their computing resources. We were able to give a demonstration of our system at the CCGrid conference with the help of these institutions.

References

[1] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and M. Matsuoka: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, Journal of Grid Computing, 2003 (to appear)

[2] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee and H. Casanova: GridRPC: A Remote Procedure Call API for Grid Computing, GWD-I, APM Research Group, http://www.eece.unm.edu/~apm/docs/APM_GridRPC_0702.pdf (2002)

[3] I. Foster and C. Kesselman: Globus: A Metacomputing Infrastructure Toolkit, Proceedings of Workshop on Environments and Tools, SIAM (1996)

[4] H. L. Tanaka and D. Nohara: A Study of Deterministic Predictability for the Barotropic Component of the Atmosphere, Science Reports of the Institute of Geoscience, University of Tsukuba, section A, Vol. 22, pp. 1-21 (2001)

[5] I. Foster: Designing and Building Parallel Programs, Addison Wesley (1991)

[6] http://www.apgrid.org/

[7] N. Karonis, B.Toonen, and I. Foster: MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface, Journal of Parallel and Distributed Computing (2003), http://www3.niu.edu/mpi/

[8] M. Hirano, Y. Tanaka and S. Sekiguchi: Grid PSE Builder: Design and Implementation of a Grid-enabled Problem Solving Environment Builder, Proceedings of SWOPP, (2003) (to appear) (in Japanese)

Multivariate Minimization Using Grid Computing

Kandle Kulish Jerry Perez Phil Smith
High Performance Computing Center
Texas Tech University

Abstract

In November 2002, Texas Tech implemented a production grid powered by Avaki software. This paper addresses a new, simple approach to parallel programming which requires little or no actual knowledge of parallel programming structures, such as MPI, enabling even the novice user to take advantage of this resource. We describe and test this feature of grid computing called multiple input files. To demonstrate and test this feature, we will solve a toy multivariate global minimization problem. We would like to solve $\min\{f(x) : a_i \leq x_i \leq b_i\}$, where a and b are the vectors which define the boundary of our domain. Using the multiple input file feature we subdivide the region, and each subregion is solved on a different processor on the grid yielding linear speed ups and more accuracy for the same amount of compute time.

1 INTRODUCTION

As technology advances, so does our scientific curiosities and capabilities. We are now at the point that even with our fastest supercomputers some calculations would take months or years to compute. To help speed up these computations, parallel computing developed, which could partition the workload of a program onto multiple processors. But with the rising costs to purchase and maintain these multiprocessor supercomputers, a more efficient solution had to be found. One such solution is the Beowulf cluster, which links inexpensive Linux machines together and supports parallel computing. But what about the Windows personal computers in offices and labs that have unused compute cycles? Thus the concept of grid computing emerged to link together not only Windows machines, but Beowulf clusters and supercomputers as well. Grids [1] link all your resources together into one giant heterogeneous compute and data grid. Texas Tech implemented a production grid powered by Avaki software in November 2002 called TechGrid. In this paper, we will discuss a feature of grid computing : multiple input files. This feature allows novice users to access significant computing power with little or no knowledge of parallel computing methodologies. Basically, this is a way of forcing parallelism without having to use any parallel coding structures such as MPI. We will illustrate this feature via a toy global minimization problem. The actual technique of minimization currently used in this example is rather simplistic since we are focusing mainly on the new grid feature. By using the multiple input files to subdivide the domain into regions, a more accurate global minimum is probable. Further sophistication in the minimization algorithm is currently a work in progress.

2 MULTIVARIATE MINIMIZER

2.1 The Problem

One problem of interest in the mathematical world is the minimization of functions. Functional minimization is needed in many real world applications such as solving nonlinear ordinary and partial differential equations, financial modeling, plant optimization, and resource allocation. Our goal is to write a computer program which will minimize a function over an n-dimensional domain. For simplicity we will define our n-dimensional domain by two vectors of length n. The first, a, will define the lower left corner of our domain and the second, b will define the upper right corner of our domain. For the purpose of testing, we chose a function for which we know the true minimum. For this model problem, we define the functional value to be the sum of the squares of the x_i's over all i, i.e.

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} (x_i)^2. \tag{1}$$

Thus, for a function of this nature, the true minimum would occur at the lower left corner defined by vector a, when considering $a \geq 0$. So the problem can be stated as

$$\min\{f(x) : a_i \leq x_i \leq b_i,\ i = 1, \ldots, n\}. \tag{2}$$

2.2 The Coding Approach

For a complete copy of all programs used in this experiment, see the appendix. The main program, *mini_no_p.c* begins by reading in from an input file the dimension of the domain, the number of iterations, and the vectors a and b which will define the lower and upper limits of the domain over which our function will be minimized. After echoing the input data into the output file, the function *mini_me*() is called and the approximate minimum and where it occurs is written to the output file. The function *mini_me*() is a prototype of all function minimizers. It trivially uses function values to produce an approximate minimum by generating a random vector within the domain and testing the functional value of the vector. If the functional value being tested is less than the current stored minimum, it copies the minimum and stores the vector where this minimum occurs. The program will repeat this process of generating a random vector and testing as many times as was specified in the input file.

3 GRID COMPUTING

3.1 What is a Grid

A Grid is a dynamic network of computing resources that work together as a single, uniform operating environment. The term grid computing originated in the early 1990's as a metaphor for making computer power as easy to access as an electric power grid. Grids can span locations and administrative domains, and can flexibly support dynamically changing organizations and computing requirements. The grid distributes compute jobs among compute nodes within the grid using middleware to facilitate distributed computing. TechGrid, which uses Avaki grid middleware, is the union of networked Texas Tech University desktop Windows lab machines, office machines, and Linux Beowulf clusters.

3.2 Why We Care

Grid computing is an amazing alternative to classical distributed computing. Rather than running parallel or serial jobs on a lone desktop pc, a Beowulf cluster, or even a supercomputer, grid computing unifies all your available resources which allows for more options. You can even run your multiprocessor jobs on an entire lab of windows machines completely in the background, taking advantage of unused compute cycles. In the case of this multivariate minimizer serial example, we are able to use the many processors at our fingertips to our advantage. Instead of iterating and finding minima and vectors across the entire domain on a single processor, we used the program *create_em.c* to generate multiple input files which partition the domain into regions. Each one of these regions may then be farmed out to different processors for local minimization. In doing this, you are dividing and conquering the problem, and a global minimum closer to the actual minimum is more likely. Also, our serial code now becomes embarrassingly parallel without having to use any parallelized coding structures such as MPI.

The Grid handles all the coding changes for us. We can still use the same serial code which asks for input file input.txt and writes to output.txt. We subdivide our domain into N partitions, making N input files :

$$input1.txt,\ input2.txt,\ \ldots\ ,\ inputN.txt.$$

A simple, one line command is then executed to run the job on the grid. The grid first identifies available compute resources. Next, the grid determines that you have N input files which match the requirements for the program and executes the following loop N times :

1. Locate an available node.
2. Move the executable and input file input*.txt onto the available node.
3. Rename the input file to input.txt
4. Run the executable, which will compute the minimum and write the results to the file output.txt.
5. Rename output.txt to output*.txt to match the pattern of the original input file.
6. Remove the temporary files such as the executable, input.txt, output.txt (clean up).

4 RUNNING THE CODE

4.1 Example Problem

For the purpose of comparison, we chose to test our code both on a single pc and as a grid application using the pc's connected to the grid to see which could find a better minimization. On both, we chose the number of dimensions to be 3 on a domain defined by $a = (1.0, 1.0, 1.0)$ and $b = (7.0, 7.0, 7.0)$, with the number of random vectors generated and tested to be 3000. Since the grid will support multiple input files, we again chose to subdivide our domain into thirds and thus read in the following as our $a's$ and $b's$ to define the three subdomains :

$$\begin{aligned} a_1 &= (1.0, 1.0, 1.0) \qquad b_1 = (7.0, 7.0, 3.0) \\ a_2 &= (1.0, 1.0, 3.0) \qquad b_2 = (7.0, 7.0, 5.0) \\ a_3 &= (1.0, 1.0, 5.0) \qquad b_3 = (7.0, 7.0, 7.0) \end{aligned}$$

For this data then, our input files were as follows:
input1.txt looked like the following :

```
3
3000
1.0 1.0 1.0
7.0 7.0 3.0
```

input2.txt looked like the following :

```
3
3000
1.0 1.0 3.0
7.0 7.0 5.0
```

input3.txt looked like the following :

```
3
3000
1.0 1.0 5.0
7.0 7.0 7.0
```

4.2 Results

Due to the nature of the function, the true minimum for this example would occur at (1.0, 1.0, 1.0), yielding 3.0 as the minimum. After running on a pc for the described example problem from (1) and (2), we found the minimum of the test function to be 4.114095 having occurred at the vector (1.359719, 1.001212, 1.123759). The output files from the grid execution were as follows :
output1.txt looked like the following :

```
3
3000
1.0 1.0 1.0
7.0 7.0 3.0

3.935468
1.359719 1.001212 1.041253
```

output2.txt looked like the following :

```
3
3000
1.0 1.0 3.0
7.0 7.0 5.0

12.100481
1.359719 1.001212 3.041253
```

output3.txt looked like the following :

```
3
3000
```

```
1.0 1.0 5.0
7.0 7.0 7.0

28.265493
1.359719 1.001212 5.041253
```

After computing on the grid, the three local minima found were 3.935468, 12.100481, and 28.265493 having occurred at the vectors (1.359719, 1.001212, 1.041253), (1.359719, 1.001212, 3.041253), and (1.359719, 1.001212, 5.041253) respectively. So, the approximate global minimum found using the grid was 3.935468. In this case, there was only a minor improvement in the global minimum. Of course, this is due to the simple nature of the example and not a limitation of the method.

5 CONCLUSIONS AND FUTURE RESEARCH

By running this application on our grid and taking advantage of its multifile capabilities, we were able to find a minimum which was much closer to the true minimum of our function. We would like to continue our exploration of functional minimization in several ways. We would like to construct a grid application which would read through all the output files generated, by searching the subdomains and calculate the global minimum found over all of the domain. This would allow us to then implement an adaptive search technique which would take the minimum found and draw a "box" around it and further subdivide this domain and run the multifile search again to look for a better minimum. Additionally, we would like to explore implementing more sophisticated techniques to search for our minimum such as simulated annealing or the Nelder Mead Downhill Simplex Method [2].

More importantly there are many computational areas that have "embarrassingly" parallel applications that fit the multifile paradigm. Examples of this would be : solution of stochastic differential equations, factorization of large integers, numerical integration, and genomic computing (BLAST).

References

[1] Foster I. , and Kesselman C., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, (1999).

[2] Press W. H. , Tenkolsky S. A. , Vetterling W. T. , and Flannery B. P. , *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, (1992).

[3] Salamon P. , Sibani P. , and Frost R. , *Facts, Conjectures, and Improvements for Simulating Annealing*, SIAM, Philadelphia, (2002).

6 Appendix

6.1 *create_em.c*

```
// Kandle Kulish
// 4-14-2003
// create_em.c
// The purpose of this program is to generate the
// input files to use with the program mini_no_p.c
// As it is written, three files will currently be
// created as is defined by numfiles.  I have
// the program written in such a way that it
// is dividing the domain into subregions in the
// last dimension.

#define numfiles 3
#define dimension 3
#define iterations 30000

#include <stdio.h>
#include <string.h>

main()
{
   char filename[10];
   int i,j;
   FILE *fp;
   double a[dimension] = {1.0, 1.0, 1.0};
   double b[dimension] = {7.0, 7.0, 7.0};
   double a_new[dimension];
   double b_new[dimension];

   double jump = (b[dimension-1] - a[dimension-1])/numfiles;

   for (i = 0; i < dimension; i++)
   {
      a_new[i] = a[i];
      b_new[i] = b[i];
   }
   b_new[dimension-1] = a_new[dimension-1] + jump;

   for (i = 1; i <= numfiles; i++)
   {
     sprintf(filename,"input%d.txt",i);
     fp = fopen(filename,"w");
     fprintf(fp,"%d  %d\n",dimension,iterations);
```

```
      for (j=0; j<dimension; j++)
         fprintf(fp,"%f ",a_new[j]);
      fprintf(fp,"\n");
      for (j=0; j<dimension; j++)
         fprintf(fp,"%f ",b_new[j]);
      fclose(fp);
      a_new[dimension-1] = a_new[dimension-1] + jump;
      b_new[dimension-1] = b_new[dimension-1] + jump;
   }
}
```

6.2 *mini_no_p.c*

```
// Kandle Kulish
// Multivariate Minimizer non parallelized version
// 4-9-2003h
// mini_no_p.c
// The purpose of this program is to find the minimum of a function defined
// in the subroutine funky.  It solves for the minimum by generating a random
// vector within the boundary box of [a,b] and testing the functional value
// of that vector.  If the functional value of the generated vector is less
// than the current minimum, it stores the new minimum and keeps track of
// the vector that produced it.  The program will generate M such vectors
// for testing.

// --------------------------------include files--------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "randomlib.h"
#include "randomlib.c"

// ----------------------------function prototypes-------------------------------

double funky (double *x, int n);

double mini_me (double a[],double b[],double mini_where[],int
n,int m);

double (*p)(double *x,int n);               // pointer declaration

// -------------------------------main program-----------------------------------

main()
```

```
{ // BEGIN main
  // The main program defines the boundary within which the function is to be
  // minimized with a, b.  It then calls the subroutine mini_me to minimize the
  // function and prints out the minimum and where it occurred.

  int i,N,M;                                  // looping variable
  double temp;

  double *a;                                  // lower left of boundary box
  double *b;                                  // upper right of boundary box
  double *mini_where;

  FILE *fp1, *fp2;

  fp1 = fopen("input.txt","r");
  fp2 = fopen("output.txt","w");

  fscanf(fp1,"%d",&N);
  fscanf(fp1,"%d",&M);

  a = (double*) calloc(N,sizeof(double));
  b = (double*) calloc(N,sizeof(double));
  mini_where = (double*) calloc(N,sizeof(double));

  for (i = 0; i < N; i++)
  {
     fscanf(fp1,"%lf",&temp);
     a[i] = temp;
  }
  for (i = 0; i < N; i++)
  {
     fscanf(fp1,"%lf",&temp);
     b[i] = temp;
  }
  fprintf(fp2,"%d\n%d\n",N,M);
  for (i = 0; i < N; i++)
     fprintf(fp2,"%f ",a[i]);
  fprintf(fp2,"\n");
  for (i = 0; i < N; i++)
     fprintf(fp2,"%f ",b[i]);

  fprintf(fp2,"\n\n\n%f\n\n",mini_me(a,b,mini_where,N,M));
  for (i = 0; i < N; i++)
     fprintf (fp2,"%f ",mini_where[i]);

  fclose(fp1);
  fclose(fp2);

} // END main

// ------------------------------------------------------------------------------------

double funky (double *x, int n)
{ // BEGIN funky
  // INPUT :  a pointer to vector x (of length n) to find the functional value of,
  //          and an integer n to store the length of the vector
  // OUTPUT:  the functional value of the input vector
  //
  // This subroutine defines another function we want to minimize.  In this case,
  // we define the functional value to be the sum of the squares of the x[i]'s
  // over all i (length of vector).  The functional value is returned once computed.

   double sum = 0.0;
   int i;

   for (i = 0; i < n; i++)
      sum = sum + x[i]*x[i];

   return sum;

} // END funky

// ------------------------------------------------------------------------------------

double mini_me(double a[],double b[],double mini_where[],int n,
int m)

{ // BEGIN mini_me
  // INPUT : vectors a,b of doubles to store the bottom left corner and upper
  //         right corner, respectively, of the boundary box to find the minimum
  //         within, a vector mini_where to store the location the minimum occured,
  //         an integer n to store the length of the vector, and an integer m to
  //         control how many random vectors will be tested in search of a minimum.
  // OUTPUT: the minimum functional value
  //
  // This subroutine generates a random vector m times and tests the functional
  // value of each random vector generated.  If the functional value being tested
  // is less than the current stored minimum, it copies the minimum and stores the
  // vector where this minimum occurs. The minimum is returned when finished.

   double mini_what;                                  // initialize minimum
   double *x;                                         // temporary vector storage
   double rmin, rmax;
   int i,j,k;

   x = (double*) calloc(n,sizeof(double));
   p = funky;                                  // points to funky

   for (j=0; j < n; j++)                        // initialize mini_what and where
   {
      rmin = a[j];
      rmax = b[j];
      mini_where[j] = RandomDouble(rmin,rmax);
   }
   mini_what = p(mini_where,n);

   for (i = 0; i < m; i++)
   {
      for (j = 0; j < n; j++)                   // generates random vector to test
      {
         rmin = a[j];                            // lower bound for random number
         rmax = b[j];                            // upper bound for random number

         x[j] = RandomDouble(rmin,rmax);   // calls random number generator prog
      }

      if (p(x,n) < mini_what)          // tests if min is less than current min
      {
         mini_what = p(x,n);      // if so, copies into min and stores the
         for (k = 0; k < n; k++)  // vector where this minimum occurs.
            mini_where[k] = x[k];
      }
   }
   return mini_what;

} // END mini_me
```

GridSuperscalar: a programming paradigm for GRID applications[1]

Rosa M. Badia, Jesús Labarta, Raül Sirvent, José M. Cela, and Rogeli Grima
CEPBA-IBM Research Institute, UPC, Spain
{rosab, jesus, cela}@ciri.upc.es, {rsirvent, rgrima}@ac.upc.es

Abstract

The goal of this paper is to present the GridSuperscalar ideas, a programming paradigm that eases the development of GRID applications in such a way that writing an application for a computational GRID may be as easy as writing a sequential application. In these applications, the tasks will be of the granularity of simulations or programs, and the objects will be files. An underlying run-time will exploit the parallelism existent at the task level, select the machine to send the task, send and receive the required files, etc. In this paper we present a GridSuperscalar prototype based on Globus Toolkit 2.x.

1 Introduction

Grid computing is becoming a very important research and development area in this decade. However, one of the important concerns of Grid computing is if a killer application will appear or not. This concern comes partially because of the difficulty of writing applications for a computational Grid. The goal of this paper is to present GridSuperscalar, a programming paradigm that eases the development of Grid applications to the point that writing such an application can be as simple as programming a sequential program to be run on a single processor and the hardware resources remain totally transparent to the programmer.

GridSuperscalar is based on how superscalar processors execute assembler codes [1]. Even though the assembler codes are sequential, the implicit parallelism of the code is exploited in order to take advantage of the functional units of the processor. The processor explores the concurrency of the instructions and assigns them to the functional units. Indeed, the execution order defined in the sequential assembler code may not be followed. The processor will establish the mechanisms to guarantee that the results of the program remain identical. While the result of the application is the same, or even better, if the performance achieved by the application is better than the one that would have been initially obtained, the programmers really do not care.

Another mechanism exploited by processors is the forwarding of data generated by one instruction that is needed by next ones. This reduces the number of stall cycles.

All these ideas are exportable to the Grid application level. What changes is the level of granularity: in the processors we have instructions lasting in the order of nanoseconds and in computational Grids, functions or programs that may last from some seconds to hours. Also, what it changes is the objects: in assembler the objects are registers or memory positions, while in GridSuperscalar we will deal with files, similar to scripting languages. This paper presents these ideas and a prototype that have been developed over Globus Toolkit 2.x [2].

The structure of the paper is the following: section 2 presents the main ideas of GridSuperscalar, section 3 presents the current prototype over Globus Toolkit 2.x, section 4 presents the file forwarding mechanism, section 5 presents some examples that have been developed over the current prototype, section 6 presents the previous and related work, section 7 present the ideas for future work and some conclusions.

[1] This work has been funded by the Ministry of Science and Technology of Spain under CICYT TIC2001-0995-CO2-01.

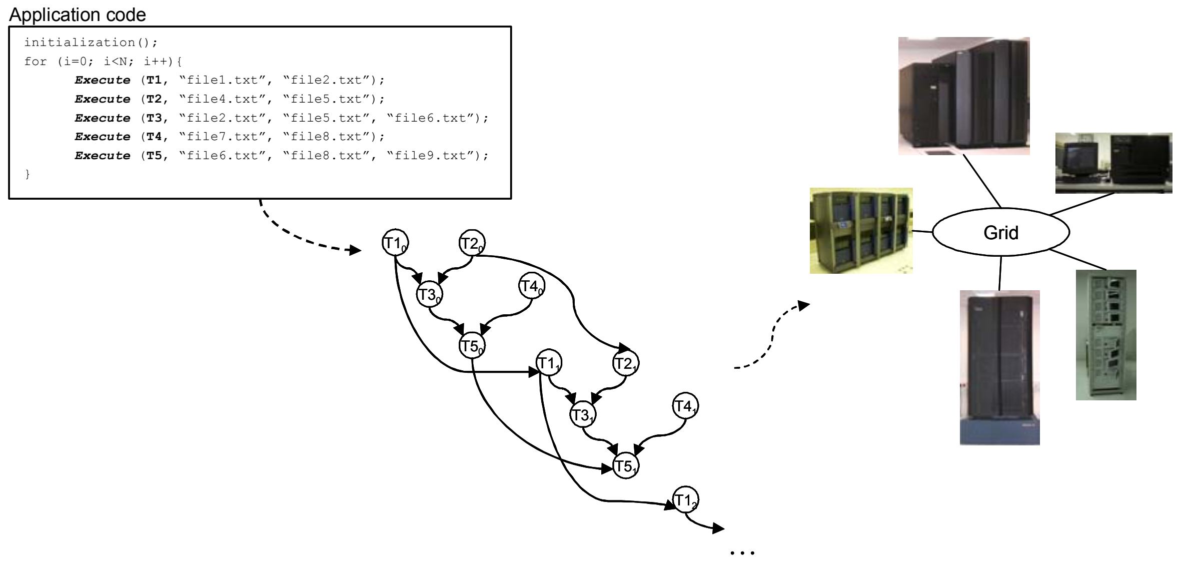
2 GridSuperscalar behavior and structure

To adapt a user application to use the GridSuperscalar only some slight modifications have to be performed in the code, mainly, the programmer selects those functions or programs of the application that should be executed in the grid by calling them inside the *Execute* primitive. Any sequential code may benefit from the GridSuperscalar.
Then the user code is linked with the GridSuperscalar run-time library and applications can be started in the usual way. The behavior of the application is then the following: those functions or programs called within instances of the *Execute* primitive are not run directly. Instead, a node of a task graph is created for each instance of the *Execute* primitive. Then, the GridSuperscalar run-time system seeks for data dependencies between the different tasks of the graph. These data dependencies are only defined by those input/output of the tasks that are files.
If a task does not have any dependence with previous tasks which have not been executed or are still running (i.e., the task is not waiting for any data that has not been already generated), it can be submitted for execution to the Grid. If that occurs, the GridSuperscalar run-time requests a Grid resource to the broker and if provided, it submits the task. Those tasks that do not have any data dependence between them can be run on parallel on the grid. This is automatically controlled by the GridSuperscalar run-time, without any additional effort for the user.
The GridSuperscalar is notified when a task finishes. Then, the data structures are updated and if there exists tasks than have now their data dependencies resolved, can be submitted for execution. Figure 1 shows an overview of the behavior that we have described above.

The rest of this section describes with more detail the GridSuperscalar. The section has been decomposed in two parts: the user view or interface that is composed of a set of primitives that should be used by the programmer in order to exploit the features of the run-time, and the run-time, which is a library that automatically *gridifies* the application while it is been run.

2.1 User interface

To develop an application in the GridSuperscalar paradigm, a programmer must do the



```
initialization();
for (i=0; i<N; i++){
       Execute (T1, "file1.txt", "file2.txt");
       Execute (T2, "file4.txt", "file5.txt");
       Execute (T3, "file2.txt", "file5.txt", "file6.txt");
       Execute (T4, "file7.txt", "file8.txt");
       Execute (T5, "file6.txt", "file8.txt", "file9.txt");
}
```

Fig. 1 Overview of GridSuperscalar behavior

following three steps:

1. Define the tasks: identify those subroutines/programs in the application that are going to be executed in the computational Grid.
2. Define the parameters of each task: identify which parameters are input/output files and which are input/output generic scalars. These two first steps are equivalent at the architecture processor level to defining the instruction set.
3. Write the sequential program invoking the sequence of tasks to be computed using calls to the GridSuperscalar primitives.

These three steps are explained here in reverse order. The basic primitive of the current prototype is the *Execute* function. The general syntax of this function is:

```
Execute (OP_NAME, parameters, ...);
```

OP_NAME is the identifier of the operation/function/program to be executed, and the rest is a list of parameters. Initially, two main types of parameters are considered: files and generics. Files can be input or output, and are the basic objects of the paradigm, and define the data dependencies between the tasks. Generic parameters can be also input or output. The input/output files are names of files that are used by the operation. The input/output generic can be any type of scalar by value parameter. Every operation can have an undefined number of parameters of each type.

To understand better the use of this function, we will give an example. Imagine a code where the function "*simulate*" is called. The function has three parameters, a scalar, an input file and an output file, for example:

```
simulate (input_file, 3, output_file);
```

Using the syntax of the GridSuperscalar, this call should be substituted by:

```
Execute (SIMULATE, input_ file, integer_parameter, output_file);
```

Somewhere, the programmer will specify that task SIMULATE has one input file parameter, one output file parameter and one input generic parameter.

So, for the programmer represents a change in the syntax and some small differences in the way the parameters have to be passed to the function *Execute*.

The programmer also needs to write the code of the tasks, the worker. The worker is the part of the code that will execute the tasks specified in the *Execute* functions on a machine in the Grid. In the current prototype, the worker is mainly a *case* structure, were given the OP_NAME the function is called. For example:

```
switch(OP_NAME){
      case SIMULATE: simulate(infile, outfile, par);
                     break;
      case ...: ...
}
```

Additionally, as the objects of the tasks are files and the task dependencies are defined by the files, in order to be able to control correctly the dependencies we need to re-define the *open()* and *close()* functions, which are substituted by *GS_open* and *GS_close*.

Also, the *GS_Barrier* function has been defined to allow the programmers to explicitly control the tasks flow. This function waits till all tasks called with Execute finish.

2.2 GridSuperscalar run-time

The core functionality of the GridSuperscalar is based in the *Execute* function. When an *Execute* with a given OP_NAME is called, an instance of the operation called is going to be executed at any time. We will call each instance of the operations specified in the Execute function a *task*. When a program calls the *Execute* function the GridSuperscalar run-time does some actions that are explained in next paragraphs.

Data dependency analysis

The data dependency analysis is performed inside the *Execute* function and a task dependency graph is automatically and dynamically built. A task dependency graph is such that the vertices of the graph denote tasks and the edges data dependencies between a pair of tasks. The analysis is performed only for the parameters that are input/output files, not for the generics.

The meaning of the task dependency graph is that those tasks that do not depend between them (it does not exist any path between them in the task dependency graph) can be **concurrently executed**.

The data dependency analysis [4] between tasks is performed based in the file parameters of the tasks. Three types of dependencies can be detected:

- Read after Write (RaW): exists when a task reads a parameter that is written by a previous one. For example:

  ```
  Execute (BT, ControlFile, fileIn1, genIn1, fileOut );
  Execute (MF, MFControlFile, fileOut, genIn2, file);
  ```

  The first *Execute* function calls a BT task, with input parameters *ControlFile*, *fileIn1* and *genIn1*, and as output parameter *fileOut*. The second *Execute* calls a MF type task with input parameters *MFControlFile*, *fileOut* and *genIn2* and as output *file*. So, the result of task BT is written in fileOut, and MF use that file as input. Therefore, task MF should be executed after task BT because needs an output of this task.

- Write after Read (WaR): exists when a task writes a parameter that is read by a previous one. For example:

  ```
  Execute (LU, LUControlFile, file, genIn1, fileOut2);
  Execute (MF, MFControlFile, fileOut, genIn2, file);
  ```

  In this case the problem is with the parameter **file** which is read by task LU and written by task MF. Initially, if the order of execution is sequential, no problem should occur. However, if MF is executed before task LU it may overwrite some data that is needed by LU.

- Write after Write (WaW): exists when a task writes a parameter that is also written by a previous one. For example:

  ```
  Execute (MF, MFControlFile, fileIn1, genIn1, fileOut);
  Execute (SP, SPControlFile, fileIn2, genIn2, fileOut);
  ```

  The problem with this data dependency may arise if task SP is executed before task MF. Although the execution of both tasks will not be affected, if there is any task that later reads *fileOut* from SP, instead it would read the resulting *fileOut* from MF.

As we have said before, the data dependency analysis is performed for the input/output files, but not for the generic parameters. In case any dependency between generics should be taken into account in the task flow execution, it can be forced by the programmer by the use of the *GS_Barrier* function. This function will wait till all the previous called tasks in the graph have finished.

Renaming

Although the program will finish correctly if all type of task dependences are respected, only the RaW dependencies are unavoidable. That is the reason why RaW dependencies are also called *true dependencies*. However, WaR and WaW dependencies can be eliminated by means of renaming. The renaming is just, as the word say, the change of name of the parameter. For example, in the WaR example instead of using the same **file** as parameter, if the system *renames* the file parameter for the MF task, then the WaR dependency disappears. Of course, those tasks that later read the MF output file as input should also have their input file *renamed* with the same name as the output file of the MF task. The WaW case is almost identical. As with the data dependency analysis, the renaming is performed **automatically** inside the Execute function.

Resource brokering

The resource broker is not included in the GridSuperscalar. However, an interface between a resource broker and the GridSuperscalar should exist. This interface is called from the *Execute* function when the task is going to be submitted for execution and a hardware resource where a task should be executed is obtained. In the current prototype, the broker is very simple, but the in further versions it should be able to interface with much more powerful brokers.

Task Submission

The task submission is again performed inside the Execute function. If the task that has been called in the current Execute function, does not have any unresolved data dependence, and a hardware resource has been provided by the broker, then the task is submitted for execution. If any of the previous conditions is not accomplished, then the task remains pending till all the conditions to be executed are met.
The task submission is composed of two steps: file submission and task submission. In the file submission step, those files that are input of the task are sent to the destination machine where the task is going to be executed. The generic input parameters are passed to the destination machine too. In the submission step, the task itself is called to be executed in the destination machine.

End of task notification

When a task finishes, the run-time of the GridSuperscalar should be notified. Then, the task dependency graph is modified according to the data dependencies that have been accomplished by the end of this task. So, tasks that depend in the task that has finished and that do not have any more data unresolved dependency can now be submitted for execution.

Results collection

Once a task ends, the results (files or generics) should be transferred to the original machine, since the philosophy of the GridSuperscalar is that the remote execution of the tasks is transparent to the programmer. So, the output files should be sent back to the initial machine. Also, the output generics should be returned to the original machines.

2.3 Other aspects

Task synchronization

In some cases the programs are organized in different parts of the code, and before beginning a part, the tasks from previous ones should have end. In those cases, the programmer can use the *GS_Barrier()*.

File modifications

As the dependency task graph is based on the file dependencies, the run-time needs to control when a local file is modified. If the main program does not directly modify the input files to the tasks, then there is no problem. However, if the main program modifies the input file of a given task (for example, if the task has an input file with execution parameters and the main program modifies this file before each task execution), then a

mechanism should be provided to control this modifications. If not, errors in the execution should arise.
This control is performed through the *GS_open()* and *GS_close()* functions. The functionality of these functions is partially the same as the *open()* and *close()* functions. Under the assumption that if a file is being opened with the write option is because its going to be modified, if the file is part of any WaR or WaW dependency, the same renaming techniques explained before are applied.

3 Globus-based implementation

In current prototype, the base programming language to write the main program and workers of an application is C++. The implementation of the prototype uses as underlying middleware the Globus Toolkit 2.x APIs.
As usual, some of the implementation decisions are bound to the used underlying middleware. For example, a previous prototype version of GridSuperscalar was implemented over the MW [17], a C++ API defined over Condor-PVM [18]. Some restriction on the task scheduling and brokering make us think on other possible basic middlewares.
However, the core of the GridSuperscalar is independent of the Grid middleware and we expect that future versions could be used with other software. This section will detail some of the aspects of the implementation.

Definition of tasks and parameters

In the current prototype, those tasks that a programmer wants to execute in the Grid should be defined in a text input file. The syntax of the file is:

```
OP_NAME n_in_files n_in_generic n_out_files n_out_generic
```

Where n_in_files is the number of input files, n_in_generic is the number of input generics and so on. For example, for the application described in section 5.1, the following file is defined:

```
FILTER 1 2 1 0
DIMEM 2 0 1 0
EXTRACT 1 0 1 0
```

For writing the worker, the programmer may program the code as it would do in a sequential code, with the difference that the parameters will be passed in the *argv*, *argc* parameters of the worker.

Resource brokering implementation

Current version interacts with a very simple version of a broker. The *Execute* function asks for a hardware resource on demand (when the task is ready to be sent). The broker has a list of machines and a maximum number of tasks allowed to be sent to each machine. It uses a Round Robin policy to assign the machines to the tasks and always avoid that the maximum number of tasks running on each machine is exceeded.

Task submission implementation

As we have specified in section 2 the task submission is composed of two steps: file submission and task submission. Current version of the Resource Specification Language (RSL) used in Globus allows to specify which files should be transferred to the destination machine, when a new scratch directory should be created or not and which files should be transferred as output to the original machine. The GridSuperscalar takes advantage of this feature to transfer the input and output files. The scheme followed initially regarding the file management is that one temporal directory is made for each task in the destination machine where the task is going to be executed. All the

files of a task are sent from the starting machine (where the main code is begun) to the destination machine temporal directory. Once the task finishes, those output files generated by the task are sent back to the starting machine and the temporal directory and input files are erased from the destination machine.
The task submission is based in the Resource Management Client API, by use of the calls *globus_gram_client_job_request* for task submission, the *globus_gram_client_callback_allow* call for asynchronous end of task synchronization, and *globus_poll_blocking* for end of task synchronization at the end of the program.

End of task notification implementation

To notify the GridSuperscalar run-time when a submitted task has finished the asynchronous state-change callbacks monitoring system provided by the Resource Management Client API [6] is used. The *globus_gram_client_callback_allow*() function opens a TCP port which listens for messages from the Globus job manager (one of the parameters of this call is a *callback_func* function, provided by the programmer). It returns a *callback_contact* that can then be used for globus*_gram_client_job_request*() calls, for which the *callback_func* will be called for state changes of the jobs submitted. In the GridSuperscalar run-time when a task enters the done state, the provided callback function is executed. The data structures are updated with the information of what task has finished, and then the broker is notified. Also, job manager error handling is performed. These are the only functionalities included in the callback function.
The initial implementation was such that inside the callback function we submitted the next tasks that were then ready. However, due to problems with the Globus callbacks the functionality was reduced to the one mentioned above (some callbacks were loosed if we call the globus*_gram_client_job_request*() from the code of the callback function).

Task scheduling implementation

The task scheduling mechanism is distributed between the Execute call, the callback function and the *GS_barrier* call. Each task enters into the run-time data structures when it is instantiated with an Execute call. Different possibilities may arise in this moment. One of the cases, which has already been explained, is the case when the task can be submitted immediately after being created. If the task has not been scheduled because there exists some data dependency that should be solved before, then it has to wait. Once the tasks that are responsible of those data dependencies end, the data structures will be updated. The callback function only marks the tasks that have finished. The data structure update is performed inside the following instances of the Execute function. After the data structures have been updated, those tasks that have now their data dependencies solved are submitted for execution. This last step is always bound to machine availability.
The remaining case is when no more Execute calls are going to be called but some tasks are still pending in the graph. Then, the GridSuperscalar performs a non CPU consuming wait inside the *GS_Barrier* primitive before ending the program.

Results collection implementation

Similarly to the input file submission, the output files should be resent to the original machine. This is as before specified in the RSL of the task.

4 File forwarding

One of the factors that reduce the concurrency of the execution of the tasks is the RaW dependencies. As it has been explained before, these dependences are unavoidable.

Also, the file that is output of a task has to be sent to the machines were the receiving task is going to be executed, adding more latency.
In this section we present a way to reduce the impact of this fact in the performance of the application. The idea is related with the forwarding mechanism used on the processors pipelines, where data produced by one instruction and needed by the following one is directly *forwarded* in such a way that cycle stalls are reduced.
Once a task has begun to write its output files, the tasks that are waiting can start their execution and can start to read the results. Therefore, a mechanism has been implemented using a socket between the tasks that writes and read the file. To give an example of the idea, the functionality of the two following commands:

```
> simulator1 <file_in.cfg >file_out.txt
> simulator2 <file_out.txt >file_out2.txt
```

is equivalent to:

```
> simulator1 <file_in.cfg | simulator2 >file_out2.txt
```

Instead of using the pipe, a socket is opened when a file is opened for write inside a task. When the same file is opened for read in another task, the other side of the socket is opened. The write/read operations do not need to be substituted, since the writing task writes into the socket and the reading task reads from the socket. However, in the implementation the writing task also writes the data into the file, since this forwarding mechanism is intended to be transparent to the programmer.
This mechanism has been implemented by means of dynamic interception of the open, close, read and write operations by using Dyninst [7].
The scheduling schema is slightly modified with the use of this mechanism. Now, a task (T2) that has a RaW data dependency with a running task (T1) is started when the task T1 opens the file that is responsible of the data dependency. The two tasks will run concurrently then. Although this will enlarge the degree of concurrency of the application, care has to be taken since deadlock situations may arise.
This forwarding mechanism is currently under development and has not been used in the experiments detailed in next section.

5 Usage examples

Several examples have been implemented with the GridSuperscalar since now, although the current version of the GridSuperscalar can still be considered a prototype. We have selected two examples for the paper: a very simple example that allows us to show the details of an application written with the GridSuperscalar paradigm and the NAS Grid Benchmarks.

5.1 Simple optimization search example

In this example, a set of N parametric simulations performed using a given simulator (in this case, we used the performance prediction simulator Dimemas [8]) are launched, varying some parameters of the simulation. Later, the range of the parameters is modified according to the simulation results in order to move towards a goal. The application runs until a given goal is reached.
The body of the program in a regular programming environment would have been:

```
Range = initial_range();
while (!goal_reached() && (j<MAX_ITERS)){
      for (i=0; i<ITERS; i++){
            L[i] = gen_rand_L_within_current_range(range);
            BW[i] = gen_rand_BW_within_current_range(range);

            filter (“bh.cfg”, L[i], BW[i], “bh_tmp.cfg”);
            dimemas_funct(“bh_tmp.cfg”,“trace.trf”, “dim_out.txt”);
            extract("dim_out.txt", "final_result.txt");
        }
        generate_new_range("final_result.txt", &range);
        j++;
}
```

Figure 1Simple optimization search example

Each called function performs the following:

- *filter*: substitutes two parameters (L and BW) of the configuration file *bh.cfg*, generating a new file *bh_tmp.cfg*
- *dimemas_funct*: calls de Dimemas simulator with the *bh_tmp.cfg* configuration file. *trace.trf* is the input tracefile and the results of the simulation are stored in the file *dimemas_out.txt*
- *extract*: greps the result of the simulation from the dim_out.txt file and it stores in final_result.txt file
- *generate_new_range*: from the data in the final_result.txt file generates the new range for parameters L and BW.

This code will be written as follows in the GridSuperscalar paradigm (see Figure 2):

```
Range = initial_range();
while (!goal_reached() && (j<MAX_ITERS)){
        for (i=0; i<ITERS; i++){
                L[i] = gen_rand_L_within_current_range(range);
                BW[i] = gen_rand_BW_within_current_range(range);
                Execute (FILTER, "bh.cfg", L[i], BW[i], "bh_tmp.cfg");
                Execute (DIMEM, "bh_tmp.cfg","trace.trf", "dim_out.txt");
                Execute (EXTRACT, "dim_out.txt", "final_result.txt");
        }
        GS_Barrier();
        generate_new_range("final_result.txt", &range);
        j++;
}
```

Figure 2 Code of the search example using the GridSuperscalar

The programmer also has to write the worker. In this example, the worker would be:

```
                switch(atoi(argv[2]))
                {
                        case FILTER: res = filter(argc, argv);
                                     break;
                        case DIMEM:  res = dimemas_funct(argc, argv);
                                     break;
                        case EXTRACT:res = extract(argc, argv);
                                     break;
                        default:     printf("Wrong operation code\n");
                                     break;
                }
```

Figure 3 Code of the worker for the search example

Additionally, the *filter*, *dimemas_funct* and *extract* have to be adapted, since the parameters are passed in the *argc* and *argv* parameters of the worker main program.
Some preliminary results of this example are shown in Table 1. Two different machines have been used: Khafre, an IBM xSeries 250 with 4 Intel Pentium III, and Khepri a node of an IBM Power4 with 4 processors. In each case, a maximum number of tasks that could be sent to each machine was set. The poor gain of the cases with more concurrent tasks was supposed to be due to the latency in transferring the input tracefile.

Machine	# max tasks	Elapsed time	Machine	# max tasks	Elapsed time
Khafre	4	4 min 51 s	Khepri	4	22 min 36 s
Khafre	8	4 min 46 s	Khepri	8	16 min 25
Khafre +	4+4	9 min 59 s			

Khepri					

Table 1 Preliminary results

A second set of experiments was performed, but in this case the input tracefile was not sent to the target machine, modeling some enhancements that we will add to the run-time which are partially explained in section 7. In this case, however, we were not able to measure the case with the two machines together (the simulation times are larger, because another test case was used for these experiments).

Machine	**# max tasks**	**Ellapsed time**	**Machine**	**# max tasks**	**Ellapsed time**
Khafre	8	14 min 10 s	Khepri	8	21 min 50 s
Khafre	4	17 min 17 s	Khepri	4	29 min 23 s
Khafre	3	22 min 26 s	Khepri	3	34 min 18 s
Khafre+Khepri	8+8	14 min 35 s			
Khafre+Khepri	4+4	16 min 40 s			
Khafre+Khepri	2+2	24 min 54 s			

Table 2 Results when the tracefile is not sent to the worker

5.2 Nas Grid Benchmarks

The NAS Grid Benchmarks (NGB, [1]), which are based on the NAS Parallel Benchmarks, have been recently specified. Each NGB is a Data Flow Graph (DFG), where each node is a NPB instance (BT, SP, LU, MG or FT). As even within the same problem class there are different mesh sizes for the different benchmarks, to allow that the output of one benchmark can be used as input to another, an additional interpolation mesh filter has been added (MF).

These benchmarks have been implemented with GridSuperscalar prototype and correctly run in the CEPBA machines. With the development of this example we have been able to find the weaknesses of the programming model and validate its advantages. Using GridSuperscalar, the main program of each NGB benchmark is much simpler than in the original version. Also, less file names should be used, as the system automatically generates each version of the files with the renaming. The implementation of the NGB benchmarks has allowed us validating GridSuperscalar as an operative system that eases to develop grid applications. Table 3 shows some results obtained for the S class of some of the NGB benchmarks. All benchmarks have been run on Khafre.

# max tasks	ED.S	HC.S	MB.S	VP.S
3	99.99 s	33.07 s	248.48 s	509.21 s
4	98.2 s	33.6 s	248.9 s	508.21 s
8	65.28 s	33.35 s	248.04 s	508.13 s
16	39.81 s	33.71 s	248.64 s	508.39 s

Table 3 Results for the NGB

It is observed that only ED scales with the number of tasks. In the case of HC this is reasonable, since for HC, MB and VP at most three tasks can be executed concurrently. Also, as MB and VP transfer large files its elapsed time is much larger than the other two cases. We expect to improve those results with new features of the GridSuperscalar. However, we consider that the use of that benchmarks has allowed us to test the prototype and also we have obtained a much more clean and simple code for that benchmarks.

6 Related work

Some of the ideas presented in this paper are related to previous work developed by the group in EU projects PEMPAR, PARMAT, and ASRA-HPC. In those projects the PERMAS code was parallelized by means of PTM, a tool that totally hides parallelization from higher level algorithms [9][10][11]. With PTM an operation graph was asynchronously build and executed on top of blocked submatrix operations. A clustering algorithm distributed the work, performing a dynamic load balancing and exploiting data locality such that the communication on the network was kept at a minimum. Furthermore a distributed data management system allowed free data access from each node. Above PTM the sequential and parallel code was identical. When an application runs on a parallel machine, PTM does an automatic run-time parallelization.
Also, the work presented may have similarities with the workflow language BPEL4WS [12]or other similar, as it is proposed in the Web Services Choreography Working Group [13]. However, in these languages what we can define is the graph, with the dependencies already described. Also, it is oriented to medium size graphs, while our system may handle really huge task graphs that are automatically generated.
Some similarities can be found with the commercial tool ST-ORM [14], although this last is mainly oriented to parametric studies. In ST-ORM you can define a task graph with dependencies. Each task can be anything, from a script to a crash simulation. ST-ORM handles the job submission and results collection into heterogeneous grids. Again, the difference is that the graph and the dependencies should be defined by the user.

7 Open issues, future work and some conclusions

This paper presents the ideas of GridSuperscalar, a programming paradigm for Grid applications. We have demonstrated that exists a way to easy the programming of Grid applications. GridSuperscalar allows making use of the resources in the Grid by means of exploiting the existent parallelism of the applications at the program level. However, this paper does not present a finalized work, but the GridSuperscalar ideas and an initial prototype based in Globus.
One optimization that we want to implement is to assign tasks that share input or output files into the same computer. This will reduce the amount of files that would have to be sent from one machine to another. In this sense, an initial step is currently being developed, where the input files of each task are sent to the destination machine of the task from its initial location. But, when the task finishes, all the files related to the task (input and output files) remains on the machine where the task has been executed, and the GridSuperscalar run-time maintains the information about the location of each file in its data structures. Then, when another task that is using any of these files is going to be executed, those are sent from the current location to the machine where the new task is going to be executed. This reduces some of the transfers, but when the scheduling would be also modified to take into account the location of the files, more improvement will be achieved.
Also, we foresee the GridSuperscalar as something not bound to a single underlying middleware, as in this case is bound to Globus 2.x, but as something that can interact with workers specified in different underlying systems: as services specified with Globus Toolkit 3, or BPEL4WS, CORBA components [15] or others. In those cases, the behavior of the tasks would be described by the user in a given language, and the worker will be a service in WSDL or others automatically generated by GridSuperscalar.

Another possibility of GridSuperscalar is its usage on top of a Grid RPC system as for example Ninf-G [16]. In this case the programmer will not have to program the worker, since the worker will be a Ninf-G remote library itself. If this library already exists and it is installed, the programmer will not have to do any extra work in this sense.
The user application code will also be the same as when the GridSuperscalar run-time is over Globus. What will change is the code of the GridSuperscalar run-time itself. In the GridSuperscalar code, when submitting the tasks, instead of using the Globus API primitives (i.e, *globus_gram_client_job_request)* the Ninf-G Grid RPC API will be used (i.e., grpc_call_async). Also, the *grpc_wait_** primitives will be used inside the GridSuperscalar code to program the synchronization mechanisms.
As when used directly on top of Globus, GridSuperscalar would be able to parallelize the calls to the tasks, which are calls to the remote libraries.
Similarly, we understand that the GridSuperscalar run-time could also be programmed on top of the Grid Application Toolkit (GAT) [19]. In this case, the worker tasks will be implemented as GAT Adaptors, and the GridSuperscalar code would be programmed with calls to the GAT Engine.

8 References

[1] J.E. Smith and G.S. Sohi, The microarchitecture of superscalar processors, Proc. of the IEEE, Dec. 1995, 1609-1624.
[2] The Globus project, www.globus.org
[3] R. F. Van der Wijngaart, M Frumkin, *NAS Grid Benchmarks: A Tool for Grid Space Exploration*, Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01) , 2001.
[4] J. L. Hennessy, D. A. Patterson, D. Goldberg, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 2002.
[5] globus gass copy Documentation 2.7, http://www-unix.globus.org/api/c/globus_gass_copy/html/index.html
[6] Resource Management Client API 3.0, www-unix.globus.org/api/c-globus-2.2/globus_gram_documentation/html/index.html
[7] Dyninst : An Application Program Interface (API) for Runtime Code Generation, http://www.dyninst.org/
[8] Dimemas, http://www.cepba.upc.es/dimemas/
[9] Markus Ast, Hartmut Manz, Jesús Labarta, A. Perez, J. Solé, Uwe Schulz: A general approach for an automatic parallelization applied to the finite element code PERMAS, HPCN Europe 1995, 844-849
[10] Uwe Schulz, Markus Ast, Jesús Labarta, Hartmut Manz, A. Perez, J. Sole, Experiences and Achievements with the Parallelization of a Large Finite Element System, HPCN Europe 1996, 82-89
[11] Markus Ast, Cristina Barrado, José M. Cela, Rolf Fischer, Jesús Labarta, Óscar Laborda, Hartmut Manz, Uwe Schulz, Sparse Matrix Structure for Dynamic Parallelisation Efficiency, Euro-Par 2000, 519-526
[12] Business Process Execution Language for Web Services version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
[13] Web Services Choreography Working Group, http://www.w3.org/2003/01/wscwg-charter
[14] ST-ORM, Stochastic optimization and robustness management, http://www.easi.de/storm/
[15] CORBA Components Model, http://www.j2eeolympus.com/J2EE/CCM/CCM.html
[16] Ninf-G, http://ninf.apgrid.org
[17] “MW Homepage”, http://www.cs.wisc.edu/condor/mw/
[18] “Condor Project Homepage”, http://www.cs.wisc.edu/condor/
[19] GridLab: A Grid Application Toolit and Testbed, http://www.gridlab.org

MPI Development Tools and Applications for the Grid

Rainer Keller, Bettina Krammer, Matthias S. Müller, Michael M. Resch
High Performance Computing Center Stuttgart (HLRS), Stuttgart, Germany
{keller,krammer,mueller,resch}@hlrs.de

Edgar Gabriel
Innovative Computing Laboratory, Computer Science Department,
University of Tennessee, Knoxville, TN, USA
egabriel@cs.utk.edu

June 20, 2003

Abstract

The message passing interface (MPI) is a standard used by many scientific applications. It has the advantage of a smoother migration path for porting applications to the Grid. In this paper Grid-enabled tools and libraries for developing MPI applications are presented. The first is PACX-MPI, an implementation of the MPI standard optimized for Grid environments. The second is MARMOT, a tool that checks the adherence of an application to the MPI standard. Besides the efficient development of the program, an optimal execution is of paramount importance for most scientific applications. We therefore discuss not only performance on the level of the MPI library, but also several application specific optimizations, e. g. for a TFQRM solver and an RNA folding code, like latency hiding, prefetching, caching and topology-aware algorithms.

1 Introduction

Applications running on several machines can use various methods for data exchange between the processes. File transfer (e. g. FTP, SCP, GridFTP), socket-based communication or RPC-like systems are just some examples. The Grid programming primer [18] explains in details the different concepts and available implementations for various of these methods. For scientific applications, the MPI specification [19, 20] is the most widely used standard. Therefore, it is not surprising that several MPI-implementations optimized for Grid-environments are currently available (MPICH-G2 [16], PACX-MPI [10], Stampi [14], MPI_Connect [8], and MagPIe [17]). The widespread use of MPI, together with the availability of several Grid-enabled MPI-implementations and the smooth migration path offered by these libraries are among the reasons why more and more applications are ported to Grid environments using MPI. The efficient and reliable execution of real scientific applications is far from commonplace. First, the dynamic nature of the Grid imposes high requirements on the flexibility of an application. Second, application developers are used to existing standards, tools and libraries. Most of them don't exist for Grid environments. Often the adoption of an application to Grid environments is not a smooth transition, but a steep learning curve has to be mastered. In this paper we present tools and techniques that are useful to achieve the goal of a smooth migration and efficient, reliable execution.

The paper is organized as follows: In section 2 we describe the Grid-enabled MPI-implementation PACX-MPI together with the optimizations for collective communication and the network layer. In the following section the MPI checking tool MARMOT and its usage in conjunction with PACX-MPI is presented. In section 4 we describe several optimizations that can be done on the application level and have been successfully applied to a TFQMR solver and an RNA folding application.

2 PACX-MPI, a Grid-enabled MPI Library

The characteristics of clustered systems in Grid environments show two different levels of quality in the communication. Between MPI processes on the same host, latencies typically are in the range of microseconds and bandwidth in the range of several hundred Megabytes/second, while communication between MPI processes on different hosts shows high latencies (in the range of tens of milliseconds) and small bandwidth (ranging from a few Kilobytes/second to a few Megabytes/second).

Taking the characteristics of clustered system into account, we define the following terms, which will be used throughout the rest of the paper:

- A **local operation** may be executed on a process without requiring any communication with other processes.
- An **internal operation** requires communication between processes which are located on the same host or machine.
- An **external operation** requires communication between processes which are located on different hosts/machines in the Grid.

PACX-MPI is an implementation of the message-passing standard MPI, which aims to support the coupling of high performance computing systems distributed in a Grid. Regarding the communication characteristics of the communication subsystem described above, PACX-MPI relies on three main concepts:

- Two level hierarchy: In clustered systems, a message-passing library has to deal with two different levels of quality of communication. Therefore, the library uses two independent layers, one to handle internal operations and one to handle external ones.
- Usage of the optimized vendor-MPI library: Internal operations are handled using the vendor-MPI environment on each system, thereby exploiting the capacity of the underlying communication subsystem in a portable manner.
- Usage of communication daemons: on each host two additional local MPI processes, so-called daemons, take care of communication between systems. This makes it possible to bundle communication, thus avoiding situations in which there would be thousands of connections open. This approach also enhances security.

The current functionality of PACX-MPI includes the full MPI-1.2 standard as well as some parts of MPI-2 and parts of the MPI-2 Journal of Development. Additionally, several features make PACX-MPI well suited for heterogeneous, clustered systems, including optimized collective operations, optimizations for the handling of derived datatypes, encryption of external communication and optionally data compression to reduce the size of data transferred between machines.

In the following, we describe in more detail two new improvements in PACX-MPI, one introduced for improving the achievable bandwidth on Gigabit network environments, and a new optimization technique for the `MPI_Gather/MPI_Scatter` operations of MPI.

2.1 Network Optimizations within PACX-MPI

With latencies and bandwidth as described above, the external network performance will always be the bottleneck for heavily communicating applications.

To improve external network performance, multiple concurrent network connections were implemented into PACX-MPI. Every new connection opened is associated with the creation of a new thread by the communication daemons. The authors are aware that this may harm other concurrent traffic because of overloaded routers dropping packets of other connections. Since tests at iGrid have shown that several concurrent network users are not capable of exploiting a high-bandwidth connection, it is believed that a few connections are not overwhelming the routers inbetween. With this concept, it is easily possible to incorporate any reliable stream-oriented network protocol into the threaded connections of PACX-MPI.

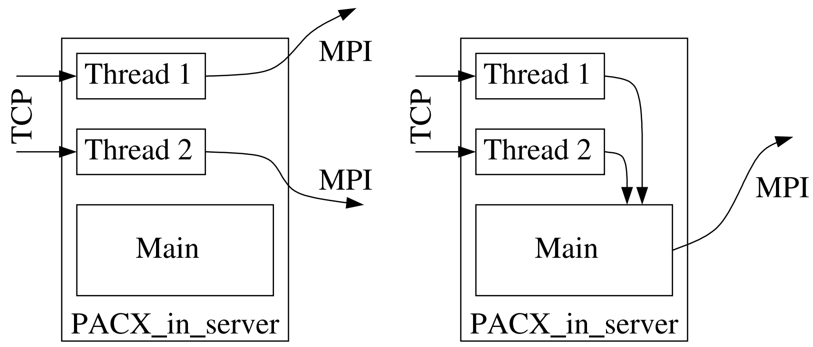


Figure 1: Implementation for thread-safe (left) and non-thread-safe (right) native MPI-implementations.

For the daemon, handling the incoming network traffic, two implementations exist, taking the thread-safety of the underlying MPI-implementation into account. If the native MPI-implementation is thread-safe, e. g. according to the naming convention of the MPI-2 standard either `MPI_THREAD_MULTIPLE` or `MPI_SERIALIZED`, the threads may not only concurrently receive packets from the network, but may also relay them concurrently to the receiving process(es). For non-thread-safe MPI-implementations, i. e. `MPI_FUNNELED`, the threads enqueue the messages for the main thread to send, as may be seen in Fig. 1. Care has been taken not to impose any unneeded copy-operations, as well as reducing the number of synchronization points, i. e. for MPI-implementations guaranteeing `MPI_THREAD_MULTIPLE`, only one mutex is being locked when relaying a packet.

2.2 Optimization of Collective Operations

Optimizing collective operations has been an active research recently at various institutions [10, 15, 17, 22]. In this section, we present the approach taken in PACX-MPI. As an example, we demonstrate the usage of the Gather operation. In the Gather operation a root process collects data from all processes within a given communicator. A variant of this function is the Gatherv operation, in which processes send data of varying lengths and only the root process has information about all of the processes and their data. The Scatter operation is the inversion of the Gather operation, therefore similar optimizations apply.

Two basic algorithms are used within PACX-MPI, which can be described as follows:

- Linear algorithm: In this algorithm the root node receives from each node its part of the message in a linear order.
- Host-based algorithm: This algorithm is split into two parts: a local part where a dedicated process on each machine, the so-called local root node, collects data from all other nodes on its machine, and a global part where the global root node collects data from the local root nodes.

From the performance point of view, we would expect the following behavior of the two algorithms described above: for short messages, the host-based algorithm should be faster than the linear one, since it minimizes the number of messages over the 'weak' link between the machines. The total execution time of a Gather operation for short messages is dominated by the wide area latencies, and therefore minimizing the number of messages helps to reduce the overall execution time of the operation. With increasing message sizes, the overall execution time of the operation is however not dominated by the communication latency between the machines, but by the 'wide area' bandwidth. In this case, the linear algorithm will be faster than the host-based algorithm, since it avoids the additional internal communication steps (which are not negligible anymore). The expected behavior is also shown in the tests in Fig. 2 (right). For this test we coupled a Cray T3E and a Hitachi SR8000 using 64 processors on both machines.

Regarding the results of the performance comparison the problem is now to determine which algorithm to use under which conditions. The IMPI-Standard [7] proposes to determine the global sum of all individual messages. If this sum is under a certain limit, the host-based algorithm shall be employed, if it is above the limit the linear algorithm shall be used. However, this solution for determining the cross-over point between both algorithms has two problems:

- For the Gatherv operations, only the root node has the required information to determine the global sum. Therefore, the global sum has to be distributed by a broadcast operation to all other processes (which includes expensive external communication).
- The global sum does not really distinguish between the requirements of each machine. For example, if one of the machines has a much lower number of processes than the other participating machines, it will be useful from a performance point of view to use to this machine the host-based algorithms, while to the other machines the linear algorithm is already used. Unfortunately, this is not possible with the IMPI approach.

Therefore, another method for determining the cross-over point between the linear algorithm and the host-based algorithm has been developed:

- Determine on each machine separately the local sum of all individual messages. This can be done using only internal communication (e. g. an internal `MPI_Allreduce`).
- Depending on whether this local sum is underneath or above a certain limit, use the host-based or the linear algorithm **to these machines**.

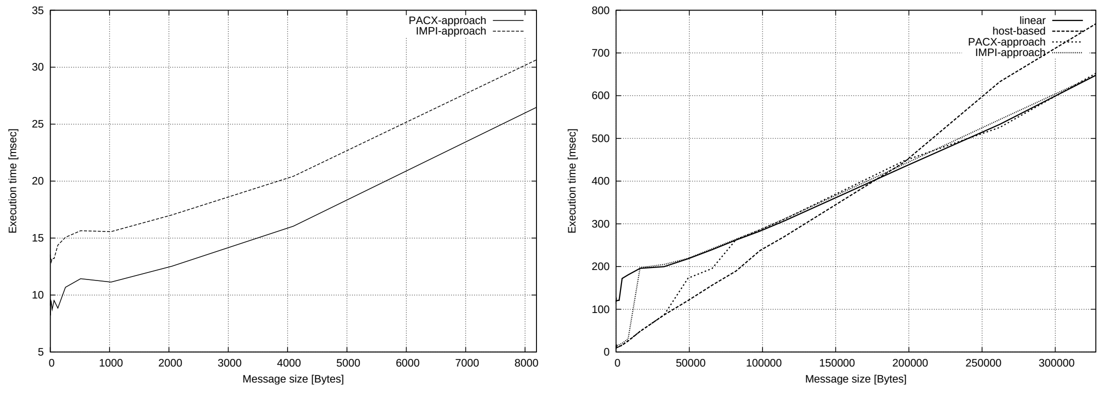


Figure 2: Comparison of the execution times for an `MPI_Gatherv` between the IMPI-approach and the PACX-MPI approach for short messages (left) and long messages (right).

Figure 2 shows the first advantage of the PACX-MPI approach compared to the IMPI approach. Especially for short individual message length, the avoidance of the global broadcast operation leads to reduced overall execution times.

The results shown in the right part of Fig. 2 were achieved using a Metacomputer consisting of three machines. The first one was the Hitachi SR8000 using 32 processes, the second one was a Cray T3E using 32 processes as well. The third partition was also running as a separate machine on the Cray T3E using only 16 processes. For a multi-site configuration like this, the PACX-MPI approach enables a 'softer' convergence towards the optimal cross-over point, by using not just a single but several switching points.

3 MARMOT, an MPI Analysis and Checking Tool

MARMOT is a tool to verify the conformance of an MPI program to the MPI standard. This is done automatically at run-time and thus may be used to help to debug the program in case of problems. Due to the complexity of parallel programming there is a clear need for debugging of MPI programs. Running

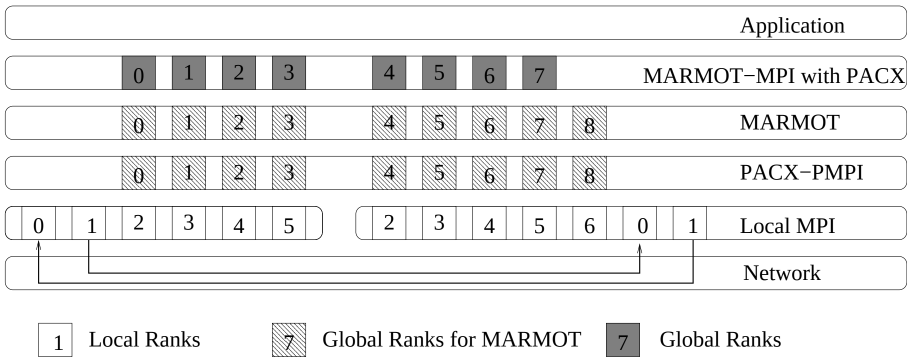


Figure 3: MARMOT used to check an application running on the Grid.

a parallel program with a Grid-enabled MPI library significantly increases the number of possible problems. According to our experience with different testbeds, there are several reasons for this: First, the MPI standard allows implementation-defined behaviour, e. g. whether or not a communication is blocking. Because most Grid-enabled MPI libraries make use of the native MPI library for local message delivery, the application has to run smoothly on several, different implementations at the same time. Second, the heterogeneous network with its high latencies and low bandwidth for external messages will not only change the performance but also alter the overall execution. For example, the amount of unexpected messages and necessary buffering might increase significantly. Last but not least the number of processors are typically larger than on any single computer where the application has been developed or tested.

Debugging MPI programs has been addressed in different ways. Classical debuggers have been extended to address MPI programs. This is done by attaching the debugger to all processes of the MPI program. This concept can also be extended for distributed MPI programs running on the Grid [13, 21]. Another approach is to provide a debug version of the MPI library (e. g. MPIch). This version is not only used to catch internal errors in the MPI library, but also detects some incorrect usage of MPI by the user, e. g. a type mismatch of sending and receiving messages [11]. Neither of these approaches address portability or reproducibility, two of the major problems when using MPI on the Grid.

3.1 Design of MARMOT

MARMOT uses the MPI profiling interface to intercept the MPI calls of the application for analysis. This profiling interface is part of the MPI standard, therefore any MPI implementation that conforms to the MPI standard needs to provide this interface. Like PACX-MPI, MARMOT adds an additional MPI process for all tasks that cannot be handled within the context of a single MPI process, like for example deadlock detection. Information between the MPI processes and this additional debug process are transferred using MPI. Another possible approach is to use a thread instead of an MPI process and use shared memory communication instead of MPI [23]. The advantage of the approach taken here is that the MPI library does not need to be thread-safe. Without the limitation to shared memory systems the tool can also be used on a wider range of platforms. Running on top of PACX-MPI it can be used to debug an application running distributed on the Grid. Figure 3 shows the configuration that is used in this case. Since MARMOT uses the profiling interface, the MPI implementation has to support the PMPI interface as required by the standard. In the case of PACX-MPI the MARMOT library has to be compiled with PACX-MPI in order to make use of the PACX-PMPI interface.

Currently MARMOT supports the full MPI-1.2 standard, i. e. both the C and the Fortran language binding. The MPI standard includes the concept of communicators, groups, datatypes and operators. These are mapped to corresponding MARMOT communicators, groups, datatypes and operators (and vice versa) and can thus be scrutinized.

- Communicators: MARMOT checks whether the communicator is valid, e. g. in cartesian calls.

- Groups: MARMOT checks whether the group is valid and, e. g. when using `MPI_Comm_create`, whether it is a subset of the communicator.
- Datatypes: MARMOT checks whether the datatype is valid, i. e. whether has been committed or whether it is `MPI_DATATYPE_NULL` etc.
- Operators: MARMOT checks whether the operation is valid, e. g. when calling `MPI_Op_free`, whether the operation to be freed exists or has already been freed.
- Miscellaneous: Apart from communicators, groups, datatypes and operators, other parameters are passed to MPI calls, e. g. tags or ranks lying within a valid range. MARMOT also verifies if requests are handled properly when using non-blocking send and receive or wait and test routines, e. g. if unregistered requests are used or if active requests are recycled.

MARMOT detects deadlocks in MPI programs by making the additional debug process survey the time each process waits in an MPI call. If this time exceeds a certain user-defined limit the process is reported as pending. If finally all processes are pending, the debug process issues a deadlock warning. In the case of a deadlock, the last few MPI calls can be traced back for each process.

The possibility to introduce race conditions in parallel programs with the use of MPI is one of the major problems of code development. While it is possible to write correct programs containing race conditions and sometimes even beneficial there are many occasions where they must be avoided. The possible locations of races can be identified by locating the calls that are sources of race conditions. One example is the use of a receive call with `MPI_ANY_SOURCE` as source argument. A second case is the use of `MPI_Cancel`, because the result of the later operation depends on the exact stage of the message that is the target of the cancellation. If strong sequential equivalence is required the use of `MPI_Reduce` is another potential source of irreproducibility. By inspecting all calls and arguments, MARMOT can detect dangerous calls and issue warnings.

3.2 Developing MPI Implementations with MARMOT

A Grid-enabled MPI implementation like PACX-MPI is itself a complex piece of software with more than 130.000 lines of source code. A tool like MARMOT is therefore also useful to check the GRID-MPI library itself. Theoretically this is possible for all libraries that make use of the native MPI for local message delivery, like PACX-MPI, Stampi or MPIch-G2. In practice, the local profiling interface needs to be available. Since PACX-MPI uses source code modification and supports a PACX-PMPI interface the native PMPI interface is still available for tools like MARMOT. Figure 4 shows the required setup. The application is compiled with the PACX-MPI header, PACX-MPI uses the MARMOT-MPI library to deliver the local messages and MARMOT uses the native PMPI library to transfer the messages after inspection. The respective libraries have to be linked to the application in this order. In addition to the two processes for the communication the user has to add one MARMOT debug process on every host. Each host is checked independently, including the two processes invisible to the application. This is possible because PACX-MPI has to make sure that the application calls are mapped in a consistent way to the local MPI environment.

4 Application Issues

Despite the large number of optimizations that can be integrated into the MPI library on the level of the network or collective operations, a large number of issues have to be considered at the application level. Tools like Vampir [5, 6] help to recognize inefficient communication patterns or find and eliminate excessive messages being sent to processes on external hosts.

In [4] a new communication pattern for a tightly coupled CFD-application has been introduced for Grid environments, which reduces the execution time for an iteration of the solver, but having the drawback that the number of iterations for convergence has to be increased. In [2] completely new numerical algorithms have been developed for solving certain classes of partial differential equations (PDE), which are less

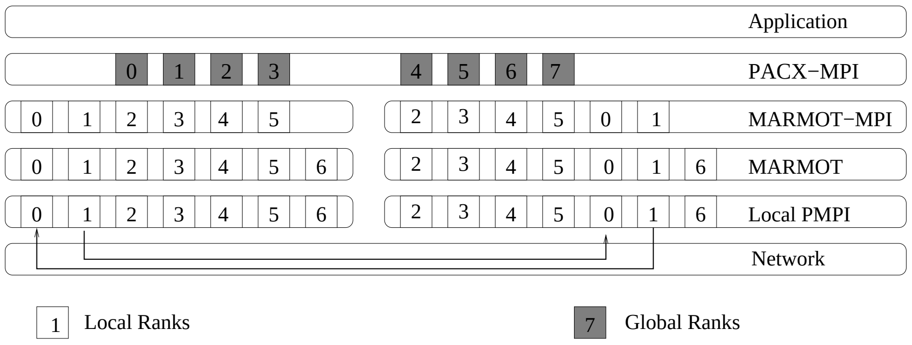


Figure 4: MARMOT used to check the PACX-MPI library.

tightly coupled then previous algorithms. The Cactus group improved the performance of their application at SC2001 by introducing a variable number of ghost-cells [1], and thus reducing the number of required communication steps depending on the quality of the network. The GrADS project [3] aims to develop a toolkit that enables the application developers to write their application such that the application is (at least partially) able to choose the best machine (or set of machines) and adapt to the new environment during runtime.

4.1 Optimizations for a TFQMR Solver

In the following, we will demonstrate some optimizations done for a parallel transpose-free quasi-minimal residual equation solver (TFQMR) [9]. For analyzing the effect of different communication patterns on the performance of the equation solver, we implemented five different versions, some of them being extensions of algorithm developed originally for optimizing collective operations. Thus, optimizing the communication patterns on this level is the next logical step beyond collective operations. The implemented algorithms are:

- **blocking**: The standard implementation of the matrix-vector product used originally in the solver consisted of several blocking communication steps. To avoid deadlocks when using blocking MPI-operations, the algorithm has to enforce a certain order for the messages, e. g. each process with an odd rank has to receive the message from the right neighbor, each process with an even rank has to send a message to the left neighbor.
- **first-come first-serve**: This implementation initiates the send and receive operations to all neighbors and then waits for completion of all operations. This communication pattern has the advantage that a certain order for incoming messages is not enforced, but is rather a first-come first-serve (fcfs) algorithm. However, a blocking communication step is done in the sense that the whole communication between all processes has to be finished, before the computation can be started.
- **overlapping communication and computation**: For calculating the matrix-vector product, the algorithm has to be modified in the following way for overlapping communication and computation:
  - Initiate the sending and receiving of the data to/from the neighbors (using e. g. `MPI_Isend` and `MPI_Irecv`).
  - Calculate the matrix-vector product for all elements that do not require any data from the neighbors. In our implementation this means that the outermost row/column could not be calculated, but all inner elements could.
  - Wait for the communication to be finished (e. g. using `MPI_Wait` or one of its derivatives).
  - Calculate all elements of the matrix-vector product which could not be calculated in the second step.

- **topology aware**: Moving the concept of topology aware communication from the implementation of collective operations to numerical operations, e. g. a matrix-vector product, one may avoid sending each message between the machines separately, but instead one process collects all messages and then sends one big message. This minimizes the number of external latencies, but is a lot more complex than the optimizations in the previous section.

- **all together**: Taking the topology aware algorithm as starting point, this version overlaps additionally communication and computation by using non-blocking communication for the external communication step.

For showing the efficiency of each of these versions, we increased the problem size according to the number of processors used. Starting with a $64 \times 64 \times 64$ mesh on a single processor, the size of the mesh is doubled if the number of processors is doubled. This test represents more the real goals of using such a tightly coupled application in Grid environments, since one is generally most interested in solving on a Grid problems that are larger than would be solved on an individual machine. In this way it is possible to solve problems that could not otherwise be solved.

Each processor is working on a constant part of the domain. The size per subdomain is getting close to the available main memory per processor. The test was executed on the virtual metacomputer up to 128 processors. For an optimal algorithm one would expect to have a constant execution time for all problem sizes.

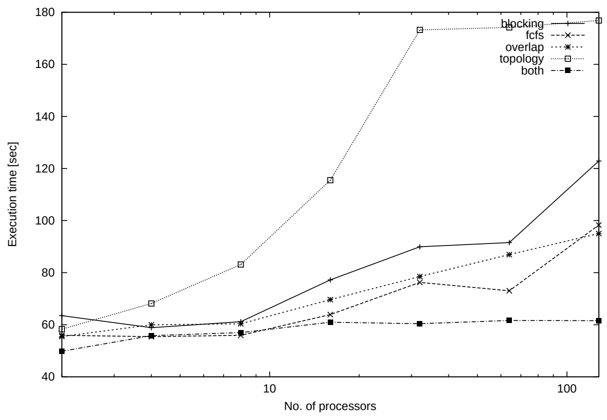


Figure 5: Comparison of the execution times for a scaleup test for all presented equation solvers.

Figure 5 shows the performance achieved with all equation solvers. The best performance is achieved by the solver that combines topology aware operations and the overlapping of communication and computation. This implementation shows a very good scaleup for the analyzed problem and nearly constant execution times up to 128 processors. The fcfs algorithm and the overlap algorithm show a good performance for a smaller number of processors. The execution time is however increasing with the number of processors.

The pure topology aware implementation shows a good performance for a smaller number of processes, but does not scale for a higher number of processors. The major reason for this is that the combination of two internal collective operations and a blocking external communication will outperform the benefits of the algorithm if no useful work can be done during these steps. Additionally, this test is using a big domain on each processor compared to the main memory available per node, and thus, the messages sent between each pair of processors are comparably large with 32K. This algorithm is supposed to perform well if a large number of small messages can be concatenated to a single big one. However, this does obviously not apply for a smaller number (max. 16 in this test) of quite large messages.

The performance of the presented algorithms is further depending on the MPI library used. Using the daemons for external communication, PACX-MPI can `really` overlap communication and computation,

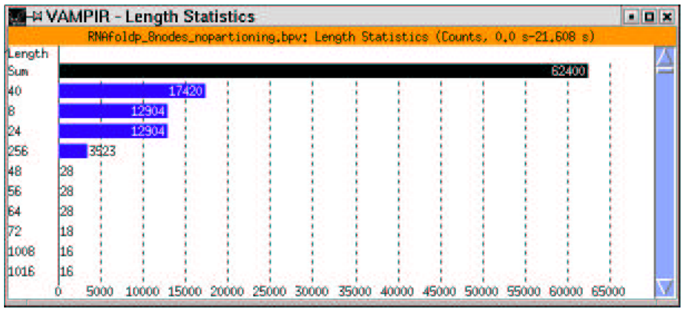


Figure 6: Number of messages sent, sorted by length.

and thus showing exceptionally good results for the algorithms using this feature. These results may in fact be different when using an MPI-implementation, which does not provide this feature.

A drawback of most of the presented algorithms are the increased complexity of the code. Compared to the original version of the solver, the number of lines in the source code have been doubled. These stem from the fact, that information, which is available in every Grid-enabled MPI library (e.g. which process/rank is on which machine), is not easily accessible in a portable manner from within the application. Since the MPI standard does not offer any portable solution for this problem, isolated solutions have been developed up to now from various implementors (e.g. the usage of the Cluster Attributes from the MPI2 Journal of Development within PACX-MPI). For providing the end-user a portable solution, a standardized interface could/should be developed within the Advanced Programming Model Working Group of the GGF.

4.2 Optimizations in an RNA-Folding application

Similar to DNA, single-stranded RNA forms into a stable three dimensional shape through a process called folding in a semi-chaotic way. The application RNAfold[12] uses an MPI-parallel dynamic programming algorithm to compute the two dimensional folding of long single-stranded RNA-sequences. Although only two dimensional, this process is very processor and memory consuming, for sequences of length n, CPU time is in $\mathcal{O}(n^3)$, while memory consumption is in $\mathcal{O}(n^2)$. Here, we present several optimizations to improve the computation/communication ratio of this application.

- **Omitting messages not needed**: At the beginning of every iteration two calls to collective operations `MPI_Barrier` and `MPI_Bcast` were issued, which were not required for production runs.

- **Coalescing of messages**: Vampir showed, that several messages in a row were being sent to the same process in a row. These were coalesced into one single message, therefore decreasing the overhead induced by function calls and ommitting transmittal of control data for each message. In this particular case, MPI's derived datatypes proved to be useful.

- **Latency hiding**: The application makes use of MPI's non-blocking communication. The programmer should be aware, that PACX-MPI internally uses a non-blocking `MPI_Ibsend` to the network daemons even for blocking calls. However, the semantics of `MPI_Ssend` require waiting for acknowledgment, thus increasing the execution time by twice the latency of the connecting network even on the sender's side.

- **Prefetching & caching of communicated data**: Another technique not being obvious at first in some applications is prefetching and caching for applications which do collect results in a non-regular pattern. At the reassembly-step of the folding process, values were gathered in a point-to-point operation, requesting one value a time from two matrices F^M and F^B, which are stored in a distributed fashion among several processes. This resulted in a high number of one-integer and three-integer-sized messages, as may be seen in Fig. 6

  After examining the access pattern of the two matrices it turned out, that both were accessed in regular patterns. Matrix F^M, as may be seen in Fig. 7, was accessed in a linear fashion (colors encode

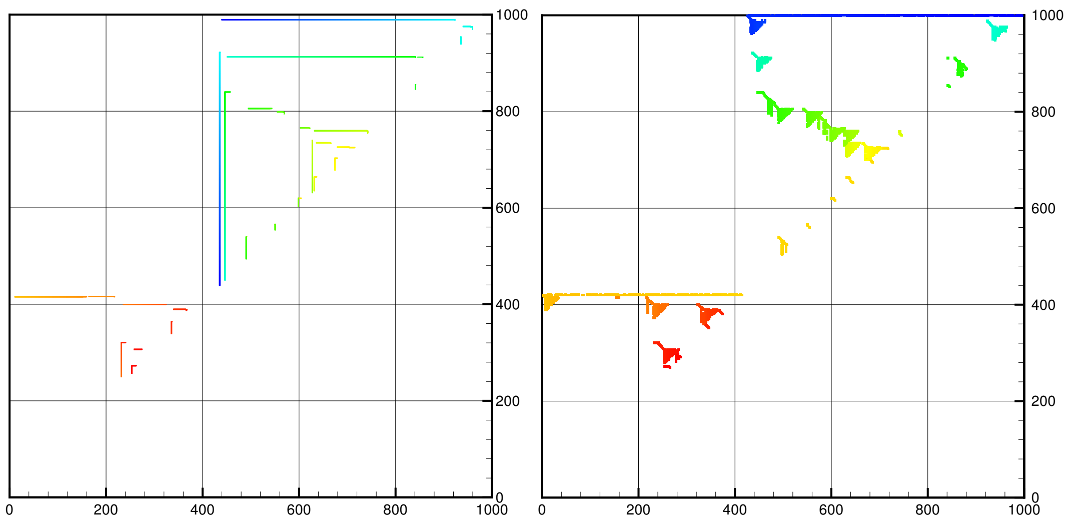


Figure 7: Access pattern of the F^M-matrix (left) and the F^B-matrix (right)

the time of access from blue to red). With only four variations for accessing this matrix, a heuristic was implemented to detect a pattern and prefetching the values in a line soon to be requested.

The F^B-matrix was accessed in a different way, still the values accessed consecutively were in regular, triangular regions. For this matrix, for the sake of simplicity, a square region of adjustable size was prefetched.

The optimizations altogether improved the time even on single HPC machines. While the first set of optimizations is generally applicable to any parallel program, the last optimization may only be used after close examination of the code.

5 Conclusion and Outlook

Different tools to develop MPI applications on the Grid have been presented. The first is PACX-MPI, a Grid enabled MPI implementation that supports the complete MPI-1.2 standard. It includes optimization for the hierarchical communication network present in the Grid. An example are the presented improvements for collective communications like Gather or Gatherv. Another important issue is the optimization of the network layer in conjunction with the external communication done by the MPI library. A multi-threaded implementation of the communication daemons of PACX-MPI handles several network connections between two machines to maximize the throughput for high-bandwidth connections with high-latency. Another tool is the MPI checking tool MARMOT. This tool is not only useful to detect non-standard conformant usage of the MPI library but is also helpful to develop a Grid-enabled MPI library on top of native MPI.

We have also presented some examples of optimizations that have to be made in the application itself. Examples are message prefetching or data caching in RNAfold or latency hiding techniques or topology aware algorithms in the TFQRM solver. With the higher complexity of applications that are adopted to the Grid systems with their deep hierarchy a tool like MARMOT will be essential. For Grid applications to perform optimally overall, it is necessary to optimize at the network layer, the MPI library layer, and on the individual application layer. The combination of PACX-MPI and MARMOT aids significantly in optimizations at all of the three layers.

Acknowledgments

This work was supported by the European projects DAMIEN (IST-2000-25406) and CROSSGRID (IST-2001-32243).

References

[1] G. Allen, T. Dramlitsch, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *Supercomputing 2001*. IEEE, Denver, 2001.

[2] N. Barberou, M. Garbey, M. Hess, M. Resch, J. Toivanen, T. Rossi, and D. Tromeur-Dervout. Aitken-schwarz method for efficient metacomputing of elliptic equations. In *Proceedings of the Fourteenth Domain Decomposition meeting*. Cocoyoc, Mexico, January 6-11 2002.

[3] F. Bernman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, L. T. D. Reed, and R. Wolski. The GrADS project: Software support for high-level grid application development, 27. August 2001. `http://hipersoft.cs.rice.edu/grads/publications/grads_project.pdf`.

[4] T. B. Bönisch and R. Rühle. Adaptation of a 3-d flow-solver for use in a metacomputing environment. In A. E. C.A. Lin, N. Satofuka, P. Fox, and J. Periaux, editors, *Parallel Computational Fluid Dynamics, Development and Applications of Parallel Technology, North Holland*, pages 119–125. North Holland, 1999.

[5] H. Brunst, M.Winkler, W. E. Nagel, and H.-C. Hoppe. Performance optimization for large scale computing: The scalable vampir approach. volume 2, pages 751–760. LNCS, May 2001.

[6] H. Brunst, W. E. Nagel, and H.-C. Hoppe. Group-Based Performance Analysis of Multithreaded SMP Cluster Applications. In R. Sakellariou, J. Keane, J. Gurd, and L. Freeman, editors, *Euro-Par 2001 Parallel Processing*, pages 148–153. Springer, 2001.

[7] I. S. Committee. IMPI - interoperable message-passing interface. `http://impi.nist.gov`.

[8] G. E. Fagg, K. S. London, and J. J. Dongarra. MPI_Connect: managing heterogeneous MPI applications interoperation and process control. In V. Alexandrov and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1497 of *Lecture Notes in Computer Science*, pages 93–96. Springer, 1998. 5th European PVM/MPI Users' Group Meeting.

[9] R. W. Freund. A transpose free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM Journal of Scientific Computing*, 14(2):470–482, 1993.

[10] E. Gabriel, M. Resch, and R. Rühle. Implementing MPI with optimized algorithms for metacomputing. In *Message Passing Interface Developer's and Users Conference (MPIDC'99)*, pages 31–41, Atlanta, March 10-12 1999.

[11] W. D. Gropp. Runtime checking of datatype signatures in MPI. In J. Dongarra, P. Kacsuk, and N. Podhorszki, editors, *Recent Advances In Parallel Virtual Machine And Message Passing Interface*, pages 160–167. Springer, 2000.

[12] I. L. Hofacker, W. Fontana, L. S. Bonhoeffer, M. Tacker, and P. Schuster. Vienna RNA Package. Internet, October 2002. `http://www.tbi.univie.ac.at/~ivo/RNA`.

[13] R. Hood. Debugging computational grid programs with the portable parallel/distributed debugger (p2d2). In *The NASA HPCC Annual Report for 1999*. NASA, 1999. `http://hpcc.arc.nasa.gov:80/reports/report99/99index.htm`.

[14] T. Imamura, Y. Tsujita, H. Koide, and H. Takemiya. An architecture of Stampi: MPI library on a cluster of parallel computers. In J. Dongarra, P. Kacsuk, and N. Podhorszki, editors, *Recent Advances in Parallel Virutal Machine and Message Passing Interface*, volume 1908 of *Lecture Notes In Computer Science*, pages 200–207. Springer, Sept. 2000. 7th European PVM/MPI Users' Group Meeting.

[15] N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *Proceedings of the 14th International Parallel Distributed Processing Symposium (IPDPS '00)*, pages 377–384. IEEE, Cancun, Mexico, May 2003.

[16] N. Karonis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. accepted for publication in Journal of Parallel and Distributed Computing, 2003.

[17] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (Ppopp'99)*, pages 131–140. ACM, May 1999.

[18] C. Lee, S. Matsuoka, D. Talia, A. Sussmann, M. Müller, G. Allen, and J. Saltz. A Grid programming primer. Global Grid Forum, August 2001.

[19] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, June 1995. `http://www.mpi-forum.org`.

[20] Message Passing Interface Forum. *MPI-2: Extensions to the Message Passing Interface*, July 1997. `http://www.mpi-forum.org`.

[21] S. Reynolds. System software makes it easy. *Insights Magazine*, 2000. NASA, `http://hpcc.arc.nasa.gov:80/insights/vol12`.

[22] S. S. Vadhiyar, G. E. Fagg, and J. J. Dongarra. Performance modelling for self-adapting collective communications for MPI. In *LACSI Symphosium*. Springer, Eldorado Hotel, Santa Fe, NM, Oct. 15-18 2001.

[23] J. Vetter and B. de Supinski. Dynamic software testing of mpi applications with umpire. In *SC2000: High Performance Networking and Computing Conf.* ACM/IEEE, 2000.

DIMEMAS: Predicting MPI applications behavior in Grid environments

Rosa M. Badia, Jesús Labarta, Judit Giménez and Francesc Escalé
CEPBA-IBM Research Institute
{rosab, jesus, judit}@ciri.upc.es, fescale@cepba.upc.es

Abstract

Dimemas is a performance prediction tool for MPI applications. In project DAMIEN this tool has been extended to predict for Grid environments. This paper presents these new developments: the new target architecture, the extension of the communication model and some results.

1 Introduction

The performance analysis of parallel programs is a must to assure its efficiency. However, the execution of production runs with the objective of doing performance analysis may be very expensive. This fact gets even worst if the target architecture is a computational Grid instead of a parallel machine. In this sense, Dimemas is a valuable tool, since allows the programmers perform predictions of the execution of MPI programs without requiring the use of the target platform.
In the IST project DAMIEN [8] , Dimemas has been extended to predict for distributed or Grid architectures. This paper presents these new developments.
Other approaches to the performance prediction of applications in Grid environment has been previously presented, although not specifically targeted to MPI applications. Between those we can find Bricks, MicroGrid, SimGrid and GridSim. A taxonomy of those and other tools is presented in [10] .
The paper is organized as follows: section 2 presents an overview of Dimemas, section 3 presents the point to point communication model and section 4 the collectives operation model, section 5 presents the results of some experiments and section 6 conclude the paper.

2 Dimemas overview

Dimemas is a performance prediction simulator for message passing applications. It helps users developing and tuning parallel applications in any machine, even in a workstation, while providing an accurate prediction of their performance on a parallel target machine or even more, on a grid environment composed of several machines (parallel or not). Dimemas reconstructs the time behavior of a parallel application on that target architecture by modeling it with a set of performance parameters.
Dimemas is useful in two phases of the life of an MPI application: during its development, to perform analysis of the effect of different parameters in the performance without requiring the use of the target architectures; and before the production phase, to select the best Grid target architecture to run the application.
The inputs to Dimemas are: a tracefile and a configuration file. The first is a tracefile of a real execution of the application on a source machine that captures the CPU bursts versus the communication pattern information. This real execution can be done in any kind of machine, even a uniprocessor machine, mapping all the MPI processes into one processor. Although the performance of that run will be very low, the Dimemas tracefile

will be valid. The second input is a configuration file that contains a set of parameters that model the target architecture.
The output of Dimemas can be simply a text output with the prediction of the elapsed time that will be get when the MPI application is run on the target architecture or a visualization tracefile. The use of Dimemas together with a visualization tool as Paraver [5] or other similar [6] , it allows the user to gain insight into the application behavior.

2.1 Dimemas architecture

The Dimemas target architecture is a computational Grid composed of a set of machines connected through an external Wide Area Network (WAN) or through dedicated connections.
Each machine of the Grid can be a network of Symmetric Multi Processors (SMP) nodes, where each of the nodes has several processors with shared memory. Figure 1 shows the machines architecture. The interconnection network that connects the nodes is modeled with a set of B buses, with a given bandwidth, and L links between each

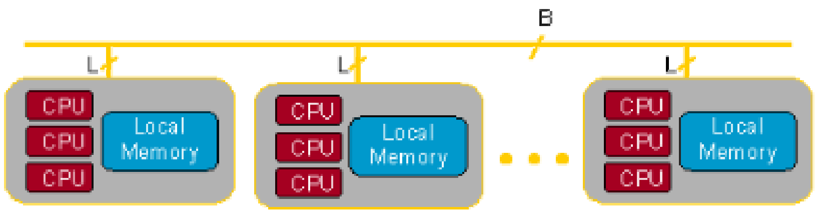


Figure 1 Network of SMP architecture

node and the interconnection network, with a given latency. Also, inside each node, a local latency and bandwidth between the processors and the memory is considered.
The Wide Area Network connecting the machines is characterized with a maximum bandwidth. This maximum bandwidth can be reduced due to traffic contention, which is modeled with a function of the traffic in the WAN. Also, the connection of the machines to the WAN is modeled with one link, with a given latency and a flight time. This latter value models a non-CPU consuming latency that represents the time invested in the transmission of the message to the destination. This flight time represents the propagation and transmission delay [11] .
Besides the WAN, pairs of machines of the target architecture can be connected through dedicated connections. The dedicated connections are considered to have a certain QoS, with guaranteed bandwidth, and similarly to the WAN, a given latency and a flight time is defined.
All these characteristics of the target architecture are defined on the Dimemas

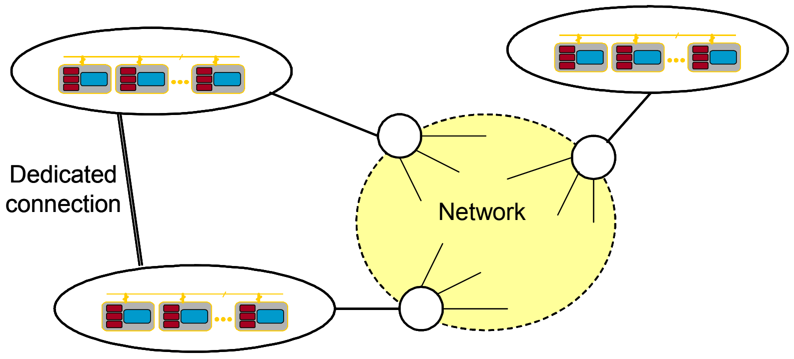


Figure 2 Dimemas architecture model

configuration file with a set of parameters. Figure 2 shows the Dimemas architecture model.

3 Point to point communication model

Dimemas has different models for the point to point communications and for the collective operations. Those communication models are user-configurable with a set of parameters allowing the user to model different communication library implementations and networks characteristics. For both point to point communications and collective communications, two levels of communications is taken into account. The local level for communications between processes mapped on processors of the same machine, and the remote level for communications between processes mapped on processors of different machines. Furthermore, we can classify the local level communications into two types: node level communications, for those that take place inside a node of an SMP, and SMP level communications, for those that take place in the SPM interconnection network of a given machine.
For point to point communications a very simple model expressed in Figure 3 is considered [5].
In this figure, we have one time line for each process (0 and 1). Each process has an associated state in each moment, represented with different colors[1]. The figure represents the different steps that a sending process goes through when a point to point communication takes place.

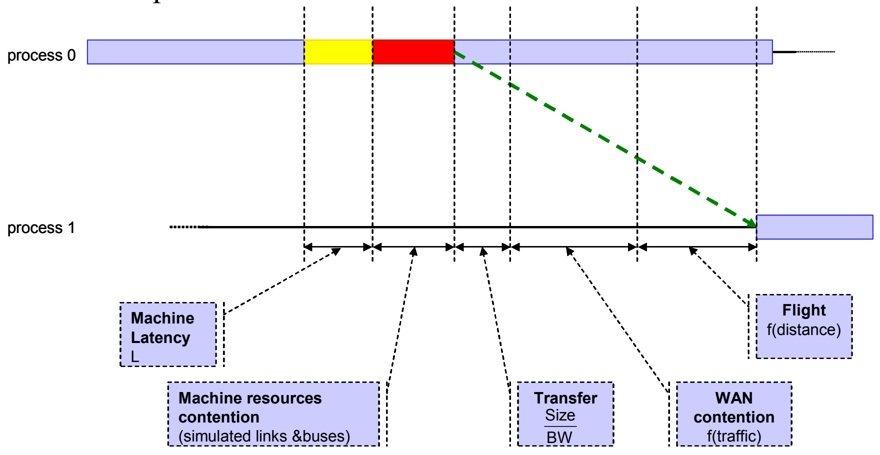

process 0
process 1
Machine Latency L
Machine resources contention (simulated links &buses)
Transfer Size BW
WAN contention f(traffic)
Flight f(distance)

Figure 3 Point to point communication model

3.1 Resource latency

The first step of the communication is the latency of the sending processor. Dimemas will select the appropriate value from the configuration file, according to the level of the communication (node, SMP or remote level).

3.2 Communication resources contention

The second step is the machines resources contention, which is simulated by Dimemas. Dimemas does not evaluates this time with a fixed value from the configuration file, but

[1] If printed in colors, blue color represents that the process is doing some calculations, yellow color represents that the process is busy with the latency startup and red color represents that the process is blocked for some reason. If printed in BW, the lightless grey represents when the process if busy with the latency startup, the middle grey represents that the process is doing some calculations and the darkness grey represents that the process is blocked for some reason.

with the time required until all the resources for the communication are available. For example, for a communication between two processes mapped on different nodes of the same machine, Dimemas will wait until one output link from the sending node, one bus in the interconnection network, and one input link to the receiving node are available. Dimemas marks the state of the process as blocked until all resources are available. Then, the required resources for this communication will be retained for that communication until Dimemas free them.
After this period, the sending process would eventually continue doing other calculations (this can vary, depending on facts as for example, if it is a blocking communication or not).

3.3 Transfer

The transfer step is calculated as the quotient of the bytes sent and the bandwidth. Again, depending on the communication level, the bandwidth value can vary. Equation (1.1) depicts the calculation that is performed to calculate this step, where BW denotes the bandwidth.

$$Transfer = \frac{Size}{BW} \tag{1.1}$$

3.4 WAN contention

For the remote level communications, a function of traffic is evaluated (*f(Traffic)*). The traffic is decomposed into two types: internal and external. The internal traffic is the one generated by the application and the external traffic is an estimation of the bytes sent to the external network by other applications. These estimated values of external and internal traffic are then weighted by factors γ and β, as it is depicted in equation(1.2).

$$Traffic = \gamma \cdot Traffic_{external} + \beta \cdot Traffic_{internal} \tag{1.2}$$

This value (Traffic) is then used to index a traffic function that returns a value between 0 (maximum traffic contention) and 1 (null traffic contention). Four different behaviors of the function of traffic can be modeled on the configuration file: constant, logarithmic, linear and exponential.
Finally, the WAN contention step is calculated as indicated in equation(1.3):

$$WAN\,contention = \frac{(1 - f(Traffic))}{f(Traffic)} \cdot \frac{Size}{BW} \tag{1.3}$$

3.5 Flight time

The flight time step is, again, only applied on the remote level communications. The flight time is a non CPU consuming latency (opposite to the latency, which is a CPU consuming latency), which represents the time invested on the transmission of the message to the destination. The flight time parameter will probably depend on the distance between sites and is defined for each pair of machines.

4 Collective operations model

To model the collective operations behavior, the following assumptions were considered:

- Synchronization: all collective operations are assumed to synchronize their execution before actually starting the communication.

- Resources: Collective operations use all the communication resources of the machines involved in the operation while the communication takes place. The model does not consider that the delay of the communications depends on the number of resources in the system.

Regarding the time invested on the operation, four phases of execution will be considered. Similarly to the point to point communications, these four phases are classified into two types: local and remote level. Local level phases will be those that take place inside a machine and remote level phases those that take place between machines. The expression that models the delay dedicated to each of such collective operations is, thus, the following:

$$T = FAN_{in}^{local} + FAN_{in}^{remote} + FAN_{out}^{remote} + FAN_{out}^{local} \qquad (1.4)$$

The FAN_{in} and FAN_{out} phases model the behavior that some collective operations have: a first phase, where some information is collected (fan in) and a second one, where the result is distributed (fan out).

The local phases are further decomposed into two parts: the node level part of the collective operation, and the SMP level part. The calculation of these phases is evaluated as follows:

$$FAN_{in/out}^{local} = \max_{all\,machines}\left(\max_{all\,nodes}\left(T_{node}\right) + \left(L_{SMP} + \frac{Size_{in/out}^{local}}{BW_{SMP}}\right)\cdot MODEL_{in/out}^{local}\right)$$
$$T_{node} = \left(L_{node} + \frac{Size_{in/out}^{local}}{BW_{node}}\right)\cdot MODEL_{in/out}^{local} \qquad (1.5)$$

$$FAN_{in/out}^{remote} = \left(L_{remote} + \frac{Size_{in/out}^{remote}}{BW_{remote}}\right)\cdot MODEL_{in/out}^{remote} + Flight\,time \qquad (1.6)$$

Where, L and BW are the latency and bandwidth at remote/SMP/node level. The MODEL is a parameter specified on the configuration file that defines the behavior of each $FAN_{in/out}$ phase for each collective operation. Similarly, the size that should be used is defined for each collective operation and phase. The values that MODEL parameter can take in the configuration file and the corresponding value that is used in the formulas is depicted in Table 1. This different values of the MODEL parameter, allow us consider different implementations of the collectives operations, from nice logarithmic trees to linear phases, and also to take into account the possibility that some phase does not exist at all.

Model	**Network level**			
	local (node)	**local (SMP)**	**remote**	
0	0	0	0	Non existent phase
CT	1	1	1	Constant time phase
LIN	P	N	M	Linear time phase
LOG	$\log_2$ P	Nsteps	$\log_2$ M	Logarithmic time phase

Table 1 Values of the Model parameter (being P the number of processors in a node, N the number of nodes in a machine and M the number of machines on the target architecture)

In case of a logarithmic model inside the SMP network (LOG value for MODEL parameter), the value used is Nsteps, which is evaluated as follows. Initially, to model a logarithmic behaviour, we will have $\lceil \log_2 N \rceil$ phases, being N the number of nodes in the SMP. However, the model also takes into account the network contention. In a tree-structured communication, several communications are performed in parallel in each phase. If there are more parallel communications than available buses, several steps will be required in the phase. For example, if in one phase 8 communications are going to take place and only 5 buses are available, we will need $\lceil 8/5 \rceil$ steps. In general we will need $\lceil C/B \rceil$ steps for each phase, being C the number of simultaneous communications in the phase and B the number of available buses. Thus, if steps_i is the number of steps needed in phase *i*, Nsteps can be evaluated as follows:

$$Nsteps = \sum_{i=1}^{\lceil \log_2 N \rceil} steps_i \qquad (1.7)$$

Finally, the values that Size can take on the configuration file and its evaluation in the formulas are shown in Table 2:

MAX	Maximum of the message sizes sent/received by root
MIN	Minimum of the message sizes sent/received by root
MEAN	Average of the message sizes sent and received by root
2*MAX	Twice the maximum of the message sizes sent/received by root
S+R	Sum of the size sent and received root

Table 2 Values of the Size parameter

For each machine in the GRID, we will have a table that configure each local (node and SMP level) phase. Table 3 shows an example of such table that was obtained with characterization of the native MPI on a SGI O2000 [12] :

Id_op	*MODEL_IN*	*SIZE_IN*	*MODEL_OUT*	*SIZE_OUT*	*Colletive operation*
0	LIN	MAX	LIN	MAX	/* MPI_Barrier */
1	LOG	MAX	0	MAX	/* MPI_Bcast */
2	LOG	MEAN	0	MAX	/* MPI_Gather */
3	LOG	MEAN	0	MAX	/* MPI_Gatherv */
4	0	MAX	LOG	MEAN	/* MPI_Scatter */
5	0	MAX	LOG	MEAN	/* MPI_Scatterv */
6	LOG	MEAN	LOG	MEAN	/* MPI_Allgather */
7	LOG	MEAN	LOG	MEAN	/* MPI_Allgatherv */
8	LOG	MEAN	LOG	MAX	/* MPI_Alltoall */
9	LOG	MEAN	LOG	MAX	/* MPI_Alltoallv */
10	LOG	2MAX	0	MAX	/* MPI_Reduce */
11	LOG	2MAX	LOG	MAX	/* MPI_Allreduce */
12	LOG	2MAX	LOG	MIN	/* MPI_Reduce_Scatter */
13	LOG	MAX	LOG	MAX	/* MPI_Scan */

Table 3 MODEL and Size parameters to model local stages of collective operations

Similarly, we will have one table with the same format for the remote communications.

5 Results

In this section we present some examples of use of Dimemas. The first example is with application RNAfold, used in the HPC-Challenge at SC02, in Baltimore. Next we present some quantitative analysis performed with Dimemas.

5.1 RNAfold application

The application RNAfold calculates the secondary structure of RNA. It was developed as part of the ViennaRNA package at the University of Vienna [1] and MPI-parallelized. For iGrid2002 and the HPC-Challenge at SC2002, this application was enhanced by Sandia National Labs and High Performance Computing Center Stuttgart (HLRS) [7] .

The experiments with the RNAfold application started at the HPC Challenge at SC2002, when HLRS used this application as one of the examples to be run on a metacomputer across the world. Unfortunately, HLRS was not able to measure those executions. So, later the experiments were repeated on a much more reduced testbed.

A first set of experiments was performed at CEPBA. The heterogeneous Metacomputer used in this case was composed of two machines: an SGI Origin O2000 server with 64 processors (Karnak) and an IBM RS-6000 SP with 8 by 16 Nighthawk Power3 processors (Kadesh).

Dimemas tracefiles of local executions on the SP3 were extracted with mpidtrace, the Dimemas tracefile generator. These tracefiles were then used to feed Dimemas simulator with different input configurations. To estimate the characteristics of the connection between the two machines the *ping* program was used. What we do is set several *pings* varying the size. Then, doing simple calculations we get estimations for the bandwidth and the flight time + latency. Then we assigned a fixed value to the external latency of 100μs and set the difference to the flight time. We set as external latency a value a little bit larger than the node latency but still on the order. At each moment, depending on the network situation, the nominal values of the WAN parameters can vary. Afterwards, measurements of distributed executions between these two machines were performed to validate Dimemas predictions. Table 4 shows some of the results obtained in this environment. The prediction for these cases was obtained by setting the network bandwidth to 0.5 MB/s and the flight time to 4.9 ms.

Benchmark	**# processors**		**Time**	
	Kadesh	Karnak	Predicted	Measured
Test 5000.seq	4	4	245 secs	289 secs
Test 5000.seq	14	14	107 secs	116 secs
Yeast DNA seq	4	4	338 secs	409 secs

Table 4 Results for RNAfold at CEPBA

Testing the prediction of Dimemas on links with high-latency was performed on a distributed heterogeneous Metacomputer consisting of a 512 processor Cray T3e900 (Hwwt3e) installed at HLRS and the IBM-SP3, as well as the Origin-2000 installed at CEPBA. Figure 4 summarizes the results of the tests performed.

In the x-axis there are the different benchmarks and configurations that we have used for these experiments. The benchmarks are Test 1000 to Test 6000, with 1000 and 6000 bases respectively, which are synthetic data sets to RNAfold. The configurations that have been considered are 4+4, with 4 processors on the O2000 and 4 on the T3E; and 6+6 and 14+14, with 6 and 14 processors on the IBM-SP and on the T3E respectively.

For all these cases, some of the measurements were done more than one time (Measure 1 and Measure 2) showing how influent the network state can be on the achieved performance.

To estimate the flight time and bandwidth of the network between Barcelona and Stuttgart, a combination of ping and traceroute tools were used. From the values obtained with those tools, a range of flight times and bandwidths were identified. These

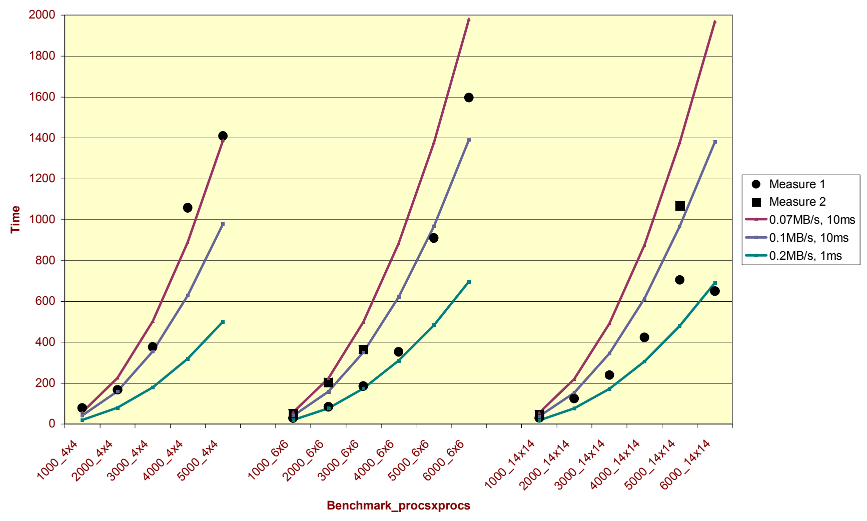


Figure 4 Results for RNAfold between Barcelona and Stuttgart

ranges of values were then used to feed Dimemas and obtain different prediction lines: for the leftmost (of each set of three), the bandwidth was set to 70KB/s and the flight time to 10ms; for the center line, to 100KB/s and 10ms and for rightmost line to 200KB/s and 1ms respectively. It can be observed how Dimemas is able to capture the behavior of the application under this range of values.

Although these results were also presented in [7] , we reproduce it here to completeness of the paper. Other experiments have been performed with other benchmarks, as with the PMB-MPI benchmarks [9] , the URANUS application or the DAMIEN project application, although this last is still under development.

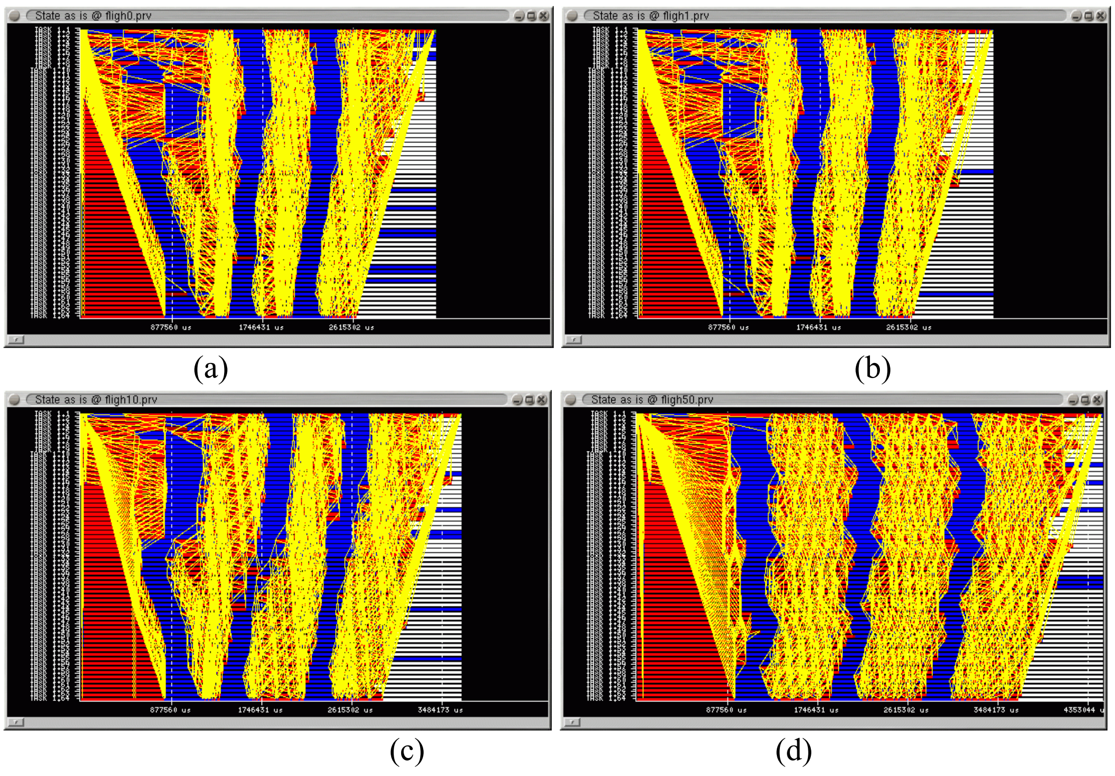


Figure 5 Effect of flight time on Uranus (a) Flight time = 0 ms; (b) 1 ms; (c) 10 ms; (d) 50ms

5.2 Qualitative experiments

What follows are some qualitative experiments that have been performed using Dimemas and ST-ORM for the analysis of different applications. ST-ORM is a metacomputing tool that allows performing intensive parametric studies in a Grid [3] . The experiments explore the influence of different parameters on the behaviour of a set of applications.

Influence of flight time

In this case was used the URANUS application with 64 processes. URANUS is a Navier-Stokes solver developed at the Institute of Space Systems of Stuttgart University (IRS) for the simulation of non-equilibrium flow around re-entry vehicles in a wide altitude-velocity range. To compute large 3-D problems the URANUS code was parallelized at the RUS/HLRS [4] . CEPBA has been using this code to perform some

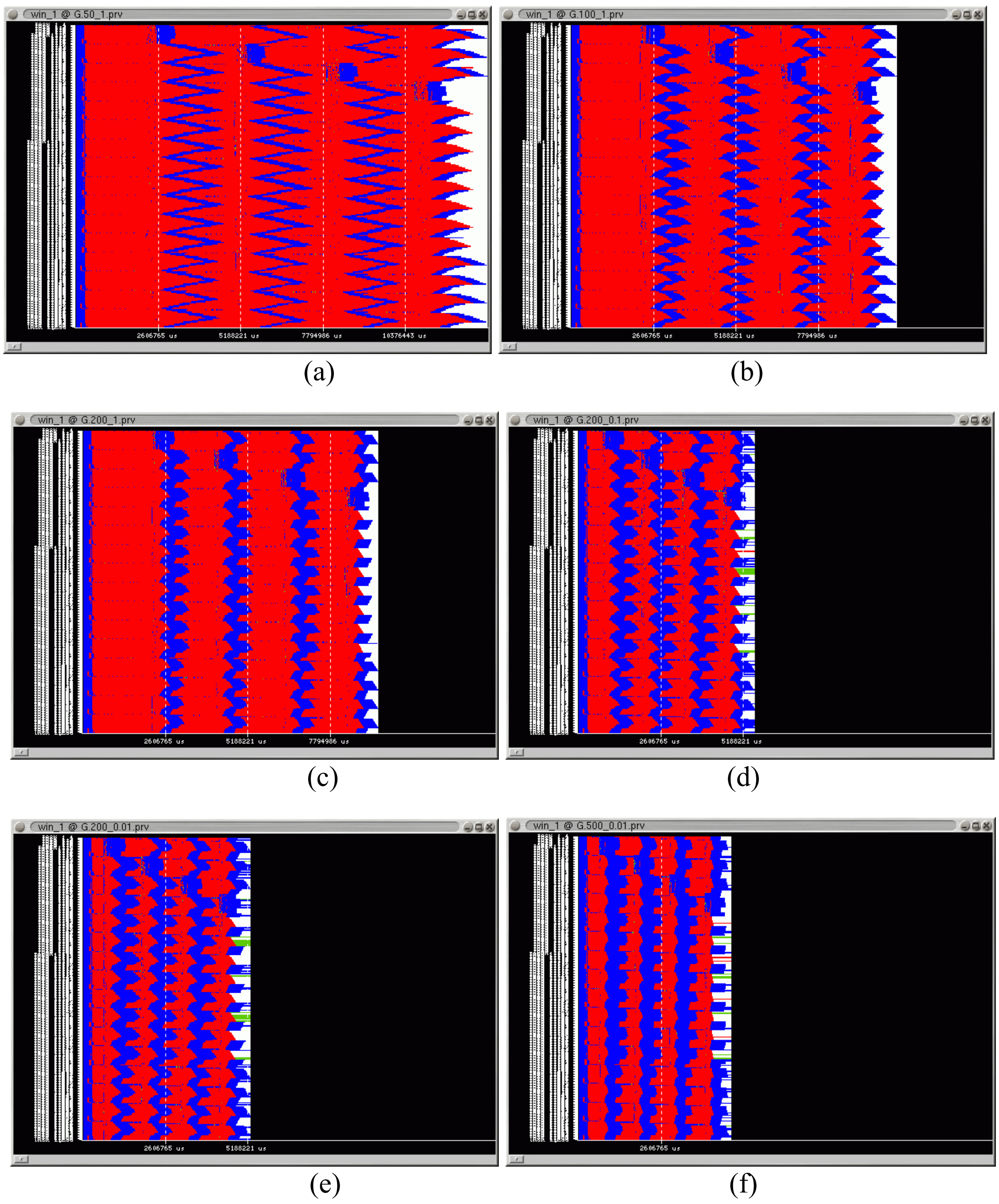


Figure 6 Effect of BW and Flight time (a) BW = 50 MB/s, flight time = 1ms; (b) 100 MB/s, 1 ms; (c) 200 MB/s, 1ms; (d) 200 MB/s, 0.1ms; (e) 200MB/S, 0.01ms; (f) 500 MB/s, 0.01ms

experiments. The simulated metacomputing environment was a set of 4 16-way SMPs with the flight time taking the following values: 0ms, 1ms, 10ms and 50 ms. Figure 5 presents a set of visualizations of different simulations from the visualization tool Paraver [5] . Paraver is a tool also developed at CEPBA. Paraver is a flexible performance visualization and analysis tool that can be used to analyze MPI, OpenMP, MPI+OpenMP, Java, hardware counters profiles, operating system activity and many other things you may think of. Dimemas is able to generate visualization traces of the simulations for Paraver and also for MetaVampir [6] .
Figure 5 shows the influence of the flight time in this architecture. In the x-axis we have the time line and the y-axis a line is seen for each process. Different colors are used to represent different states of the processes. The lines between different processes are point to point communications. In all cases, all visualizations are in the same scale. In Figure 5.a), we see the visualization of the prediction when no flight time is considered (flight time = 0). In Figure 5.b), c) and d) we see the visualization for increasing values of the flight time. We can see how the elapsed time increases, as expected. Also, we see that the communication pattern changes when this flight time increases. However, the behavior of the application is not significantly affected by the increasing values of the flight time. This kind of analysis helps developers on deciding if the application performance will be affected or not by the external network conditions.

Effect of bandwidth and flight time

For this example a tracefile of linpack application with 256 processes was used. The modeled target architecture was a 16 16-way SMPs. In this case, the influence of two parameters was studied: the external network bandwidth and the flight time. Figure 6 shows the visualizations of the results of this experiment. In this case, the difference between the different cases is much more notorious. In the cases a) and b) we can see that the behavior of the application is really degraded by the communications[2]. We can clearly see that the application spent most of this time doing the communications in those two cases.
With the same case and target architecture, a more detailed exploration of the effect of both parameters on the response time was performed, performing hundreds of simulation with different parameters. This experiment was performed using ST-ORM. ST-ORM is a metacomputing tool that allows performing intensive parametric studies in a Grid [3] Figure 7 shows two views of the 3D plots of the results of these simulations (the variables shown are the elapsed time, flight time and bandwidth). In the figure of the right we can see how the behavior of the application is degraded by the increase of the flight time parameter. In the left, we can see that this application in not affected by the bandwidth. Probably, this application will have small messages. A strategy that can be followed in this case, is group the small messages in larger ones. Then the influence of the flight time in the application will be reduced.
Other similar studies can be performed to study the influence of the different parameters of the architecture on the performance of the application. These studies can be very interesting on the early stages of the application in order to improve its performance and to select the best target architecture before the production phase.

[2] Communications are painted in grey if you print the paper in BW or in red if you print the paper in color.

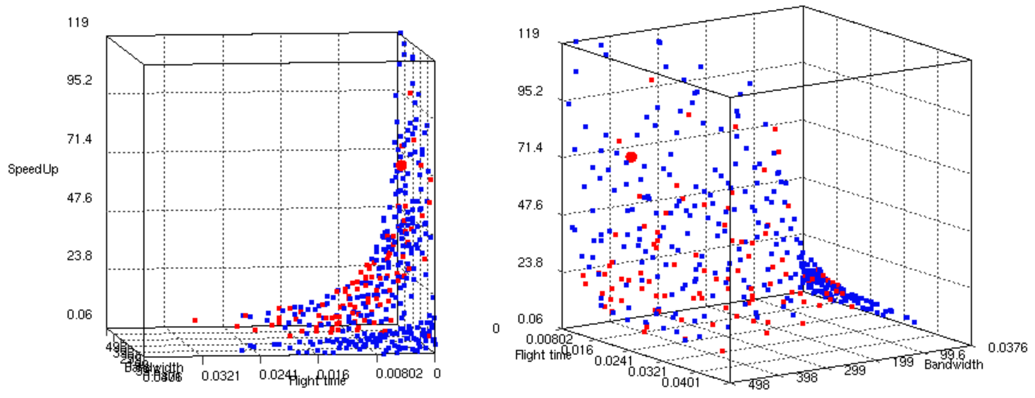


Figure 7 Effect of bandwidth and flight time in linpack

6 Conclusions

In this paper we have presented Dimemas, a performance prediction tool for message passing applications. In project DAMIEN this tool has been extended to take into account computational Grids as target architecture. This paper mainly describes the new features of Dimemas and results of some experiments.
Dimemas has been proven to be a valuable tool at two levels of the applications: during its development, for performance improvement and at production time, for selecting the most appropriate hardware resources as target architecture.

Acknowledgements

The authors acknowledge the High Performance Computing Center Stuttgart (HLRS) they collaboration in some of the experiments described in the paper. This work has been partially funded by the European Commission under contract number DAMIEN IST-2000-25406 and by the Ministry of Science and Technology of Spain under CICYT TIC2001-0995-CO2-01.

References

[1] L. Hofacker, W. Fontana, L. S. Bonhoe_er, M. Tacker, and P. Schuster. ViennaRNA Package. Internet, October 2002. http://www.tbi.univie.ac.at/_ivo/RNA.
[2] Covise. Internet, October 2002. http://www.hlrs.de/organization/vis/covise
[3] http://www.easi.de/storm/
[4] http://www.hlrs.de/people/boenisch/PROJECTS/URANUS/p_uranus.html
[5] http://www.cepba.upc.es/paraver/
[6] http://www.pallas.com/e/products/vampir/index.htm
[7] R. M. Badia, F. Escalé, E. Gabriel, J. Giménez, R. Keller1, J. Labarta, Matthias S. Müller, “Performance Prediction in a Grid Environment”, AcrossGrid conference, 2003.
[8] Referencia DAMIEN
[9] PMB - Pallas MPI Benchmarks, http://www.pallas.com/e/products/pmb/
[10] A. Sulistio, C.S. Yeo, and R. Buyya, Simulation of Parallel and Distributed Systems: A Taxonomy and Survey of Tools, International Journal of Software: Practice and Experience, Wiley Press, 2002.
[11] L. Cottrell, R Hughes-Jones, T. Kielmann, B. Lowekamp, M. Swany, and B. Tierney, A Hierarchy of Network Performance Characteristics for Grid Applications and Services, GGF, Network Measurements Working Group, http://www-didc.lbl.gov/NMWG/
[12] Sergi Girona, Jesus Labarta y Rosa M. Badia, Validation of Dimemas communication model for MPI collective operations, Lecture Notes in Computer Science, vol. 1908, EuroPVM/MPI 2000.

The Integration of Grid Technology with OGC Web Services (OWS) in NWGISS for NASA EOS Data

Liping Di, Aijun Chen, Wenli Yang and Peisheng Zhao
Laboratory for Advanced Information Technology and Standards (LATIS)
George Mason University (GMU)
9801 Greenbelt Road, Suite 316-317, Lanham, MD 20706
Email: ldi@mason.gmu.edu , achen@laits.gmu.edu

Abstract: Grid technology provides secure, fundamental methods for advanced distributed computing and data sharing over the Internet. The technology has been used in applications in many disciplines. However, applications in the geospatial disciplines are just beginning. This paper describes a project that applies Grid technology to the Earth observation environment through the integration of the Globus Toolkit with the NASA Web GIS Software Suite (NWGISS). As one of the implementations of Open GIS Consortium's (OGC) Web Services (OWS) technology, NWGISS is a web-based, multiple OGC-standard compliant geospatial data distribution and service system. It provides geospatial data access services to data users for Earth observing data archived in individual data centers (e.g., NASA Distributed Active Archive Centers, DAACs). But it does not provide secure on-demand distributed geospatial data processing and transfer among the data centers. In this project, Grid technology has been used successfully to solve these problems and a prototype Grid-enabled OGC-compliant NWGISS system was produced. This prototype is part of a large Committee on Earth Observation Satellites (CEOS) testbed for evaluating the usability of Grid technology in the geospatial disciplines, especially in Earth observations and remote sensing.

Keywords: Grid, Globus, OGC, OWS, Geospatial Data, NWGISS

1 Introduction

Grid is a promising technology for easily sharing distributed heterogeneous computing resources. It brings together geographically and organizationally dispersed computational resources, such as CPUs, storage systems, communication systems, data and software sources, instruments, and human collaborators to provide advanced distributed high-performance computing to users [1][2]. Globus, consisting of a set of services and software libraries, is the key middleware that provides core Grid capabilities. The Globus Toolkit facilitates the creation of usable Grids, enabling high-speed coupling of computers, databases, instruments, and human expertise. With Globus, scientists can run their gigabyte-per-time-step job on multiple high-performance machines at the same time, even though the machines might be located far apart and owned by different organizations. Grid technology helps scientists to deal with very large datasets and carry out complex remote collaborations. It can be used for large distributed computational jobs, remote instrumentation, remote data transfer, and shared immersive storage spaces [3].

Geospatial data are those that can be located on Earth. Earth observations through remote sensing are by far the largest source of geospatial data. The NASA's Earth Science

Enterprise (ESE) is generating a huge volume of remote sensing data daily for supporting Earth system science and application research through its Earth Observing System (EOS) project. Most of the EOS data are in HDF-EOS, the standard format for NASA’s Earth Observing System (EOS) Data and Information System (EOSDIS). These data are vitally important to studying global changes [4]. Therefore, making those data widely, easily, and freely accessible to users will greatly facilitate the global change research.

OWS is one of OGC’s many initiatives for addressing the lack of interoperability among systems that process georeferenced data. In the past several years, OGC has successfully executed several phases of the OWS initiative, including Web Mapping Testbed I (WMT-I), WMT II, OWS-1.1, and OWS 1.2. Those initiatives have produced a set of web-based data interoperability specifications, such as the OGC Web Mapping Specification (WMS), which allows interactively assembling maps from multiple servers, and the OGC Web Coverage Service (WCS) Specification, which defines interoperable interfaces for accessing geospatial data, especially those from remote sensing, from multiple coverage servers [5].

As a member of OGC and a participant of those OGC interoperability initiatives, LAITS at GMU has developed an OGC-specification compliant software package called the NASA Web GIS Software Suite (NWGISS). It is a web-based, multiple OGC-specification compliant, geospatial data distribution and service system for delivering NASA EOS data to broad user communities. Currently, NWGISS consists of a Web Map Server (WMS), a Web Coverage Server (WCS), a Web Catalog Server (WCAT), a Multiple-Protocol Geoinformation Client (MPGC), and a toolbox [4][9].

The Committee on Earth Observation Satellites (CEOS) is an international organization responsible for coordinating international civil spaceborne missions designed to observe and study the planet Earth. Its membership encompasses the world's government agencies responsible for civilian Earth Observation (EO) satellite programs, along with agencies that receive and process data acquired remotely from space. Inspired by the successful applications of Grid technology in other disciplines, the CEOS Working Group on Information Systems and Services (CEOS WGISS) started a CEOS Grid testbed in September 2002 to evaluate the feasibility and applicability of Grid technology to the EO community. The testbed, currently consisting of applications from NASA, USGS, NOAA and ESA, aims to address the use of grid technology for efficient support to diverse users worldwide for easy access and applications of EO data and to EO data providers for improving their efficiency of operation and maximizing the usefulness and benefit of the EO data which they gather [6].

As one of the NASA representatives to the CEOS Grid testbed, LAITS is contributing to the testbed by integrating OGC technology with Grid technology through the development of a Grid-enabled NWGISS. This paper describes our contributions.

2 Background

The Globus project team has just released Globus 3.0 alpha, the first major implementation of new Open Grid Service Architecture (OGSA). However, in this project, we use Globus 2.2.2, the last release before Globus 3.0 alpha. Globus 2.2 provides the following function modules: the Globus Resource Allocation Manager (GRAM) for providing a common user interface to submit a job to dispersed multiple machines, the Monitoring and Discovery Service (MDS) for providing information services through soft state registration, data modeling and a local registry, the Grid Security Infrastructure (GSI) for providing generic security services such as authentication, authorization and credential delegation for applications that will be run on the Grid, the GridFTP for providing a standard, reliable, high-speed, efficient and secure data transfer service, Metadata Catalog Service (MCS) for providing a mechanism for storing and accessing metadata of geospatial data, Replica Location Service (RLS) for maintaining and providing access to mapping information from logical names for data items to target names which may represent physical locations of data items, and other modules such as simple Certificate Authorization (CA) etc[3].

The Hierarchical Data Format (HDF) is a widely used scientific data format because of its portability and multiple data model support. NASA's EOS project extended HDF to HDF-EOS by adding three new EOS specific data models – point, swath, and grid. HDF-EOS is the standard format for EOS data and products. The EOS project is producing a tremendous amount of data in HDF-EOS format at the rate of more than 2 Tb/day. These data are highly demanded by a broad range of research and application communities. However, currently most geographic information systems (GIS) cannot easily ingest HDF-EOS data. Therefore, the development of the capability for making EOS data directly accessible by users' GIS systems through the Internet for analysis will greatly enhance the interoperability and public use of EOS data [8]. NWGISS aims to provide such capability, based on OGC Web Service specifications. Currently NWGISS works with all generic HDF-EOS files and is easy to use and easy to setup. It makes NASA EOS data immediately available and accessible to GIS developed by major GIS software vendors. Hence, it significantly increases the accessibility, interoperability, and inter-use of HDF-EOS data, improves data analysis and visualization, promotes the use of HDF-EOS data not only in global change research but also in the issues of public concern such as environment and resource management, education, and community planning [9].

The Open GIS Consortium, Inc. is a not-for-profit international membership-based organization founded in 1994 to address interoperability among systems that process georeferenced data. Since 1999 OGC has successfully implemented four web-based geospatial interoperability programs: Web Mapping Testbed I (WMT-I) in 1999, WMT II in 2000, OGC Web Service Initiative 1.1 (OWS 1.1) in 2001, and OWS 1.2 in 2002, and produced a set of web-based data interoperability specifications. WMT I produced the OGC Web Mapping Specification (WMS) version 1. WMS allows interactively assembling maps from multiple servers. WMT II produced a set of new interoperability specifications. One of the most important specifications for data access from WMT II is the draft Web Coverage Service (WCS) Specification. It allows a WCS client to access real multi-dimensional, multi-temporal data from coverage servers. WCS provides an interoperable way of accessing geospatial data, especially those from remote sensing. OWS-1 further developed WCS specification and produced three draft versions of WCS specification, v0.5, v0.6, and v0.7 [5]. Currently WCS 0.7 is going through OGC specification process to become a formal OGC interoperability specification.

In order to evaluate the applicability of Grid technology to the EO community, an interim CEOS Grid Task Team was established in May 2002 in Frascati, Italy by CEOS WGISS subgroup. In Jun 2002 the Team held a workshop at College Park, Maryland to focus on preparing a preliminary draft plan for the CEOS Grid testbed and to discuss some Grid-related projects. The draft plan was approved by WGISS in September 2002 and the testbed was officially started then. Currently, the testbed consists of five Grid demonstration projects, including the NOAA Operational Model Archive and Distribution System (NOMADS), USGS EDC's Data Delivery, ESA ESRIN Ozone, NASA GFSC's Advanced Data Grid, and NASA EOSDIS Data Pools [6][7]. As the executor of the Data Pools demonstration project, we have established a prototype of data pools based on Grid and OGC OWS technologies at GMU LAITS.

3 Integration of Grid Technology with OWS

OGC Web Service (OWS) Specifications provide standards for implementing interoperable, distributed geospatial information processing software systems (e.g. GIS), by which a user can easily share geospatial data, information, and services with others. However, OGC technology consists mainly of interface specifications and content standards. They do not provide a mechanism for securely sharing the distributed computational resources. Meanwhile, because of the large volumes of EO data and geographically scattered receiving and processing facilities, the EO data and associated computational resources are naturally distributed. The multi-discipline nature of global change research and remote sensing applications requires the integrated analysis of huge volume of multi-source data from multiple data centers. This requires sharing of both data and computing powers among data centers. Grid technology, because of its security and distributed resource sharing capabilities, is the ideal technology for filling the technology gap. Therefore, the integration of Grid technology with OWS will greatly benefit the EO community.

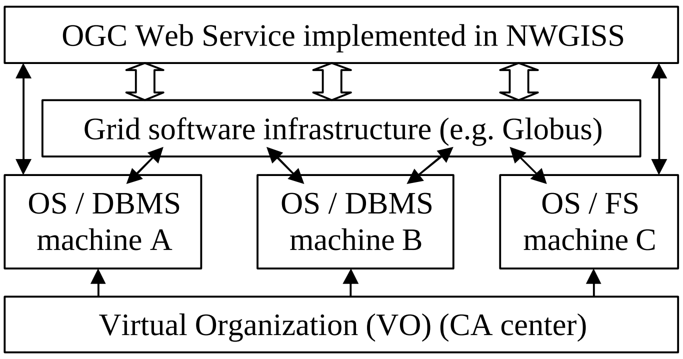

OGC Web Service implemented in NWGISS
Grid software infrastructure (e.g. Globus)
OS / DBMS machine A
OS / DBMS machine B
OS / FS machine C
Virtual Organization (VO) (CA center)

Fig. 1 Integrating Grid technology with OWS I

The information model used for the integration is shown in Figure 1. The Grid software works as the security and information infrastructures on top of the computer Operating System (OS) and its File System (FS) and Database Management System (DBMS), while

OWS software works as the application on top of the Grid security and information infrastructures to execute the secure sharing of geospatial data, information, and services. The Virtual Organization (VO) manages all sharable computers and associated resources. All machines and data users in the VO must be authorized with certifications from VO's CA (Certificate Authorization) center. OWS services will first be authenticated by the Grid security infrastructure as an authorized user and then use the Grid information infrastructure to complete the user requests. If all required resources for filling a user's request are available at the machine where the OWS service request is received, the services will be provided at the local machine without using the Grid infrastructure. Doing so will reduce the response time for filling user requests.

In addition to the integration of the OWS and Globus for secured data access and processing, we also integrated the geospatial metadata information model with Globus MCS/RLS information model [10][11] to facilitate the Grid-enabled geospatial data finding. Based on the metadata available at HDF-EOS data files [12], and referring to ISO 19115 Geographic Information- Metadata [13] and FGDC Content Standard for Digital Geospatial Metadata: Extensions for Remote Sensing Metadata [14], we proposed a general geospatial metadata information model to describe the HDF-EOS datasets. At the database level, we integrated this geospatial metadata information model with the MCS, making geospatial data retrieval Grid enabled. Users first search the geospatial metadata and MCS/RLS to get the accessible physical file name of data items. Then, users retrieve the data items by the physical file names. Details on the data retrieval process can be found on section 4.3 of this paper.

4 System Architecture and Data Flow

As a part of our initial development effort, we first set up our development environment by establishing our own Virtual Organization (VO) and Certificate Authorization (CA) center at LAITS (Figure 2), based on the Globus Toolkit. Then we integrated the Globus Toolkit with NWGISS on this VO environment to make NWGISS Grid enable. After development and proper testing, the Grid-enabled NWGISS will be installed at NASA EOS Data Pools for operational use. The following subsections will discuss the system architecture and the simplified data flow.

4.1 GMU LAITS Virtual Organization (VO) and Certificate Authorization (CA)

Currently, we have established our own VO and CA center, shown in Figure 2.

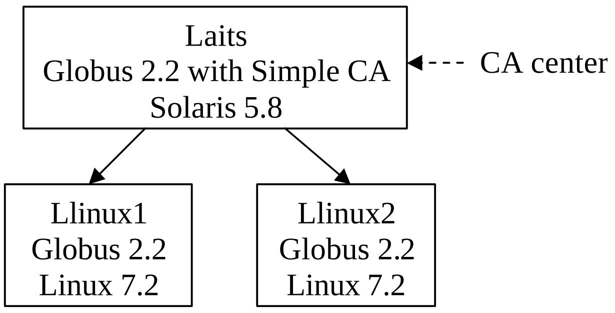


Fig. 2 Laits' VO and CA center

Globus Toolkit 2.2 was installed on a Unix server named Laits, with Solaris 5.8 and two DELL PC machines named Llinux1 and Llinux2, with Linux 7.2. We issued Globus certificates from the LAITS CA center to members of our own VO. Members in the VO can share their resources with their certificates. Our CA center’s ID is “O=Grid, OU=GlobusTest, OU=simpleCA-laits.gmu.edu, CN=Globus Simple CA”. Taking Laits machine’s Certificates as example: Host Certificate’s subject is “O=Grid, OU=GlobusTest, OU=simpleCA-laits.gmu.edu, CN=host/laits.gmu.edu”, User Certificate’s subject is “O=Grid, OU=GlobusTest, OU=simpleCA-laits.gmu.edu, OU=laits.gmu.edu, CN=Aijun Chen” and Service Certificate’s subject is “O=Grid, OU=GlobusTest, OU=simpleCA-laits.gmu.edu, CN=ldap/laits.gmu.edu”.

4.2 System Components

Figure 3 shows the architecture of the integration. The figure also shows the data flow between two machines within the VO. NWGISS consists of a map server, a coverage server, a catalog server, a geoinformation client, and a toolbox. All NWGISS components can work both independently and collaboratively, and the Globus Toolkit consists mainly of GSI, GRAM, MDS and GridFTP.

The OWS interface specifications implemented in NWGISS include the Web Map Service (WMS)[15], Web Coverage Service (WCS)[16], and Catalog Inter-operability Specification (CIS). The following paragraphs provide brief descriptions of each component. Detailed information can be found in [4].

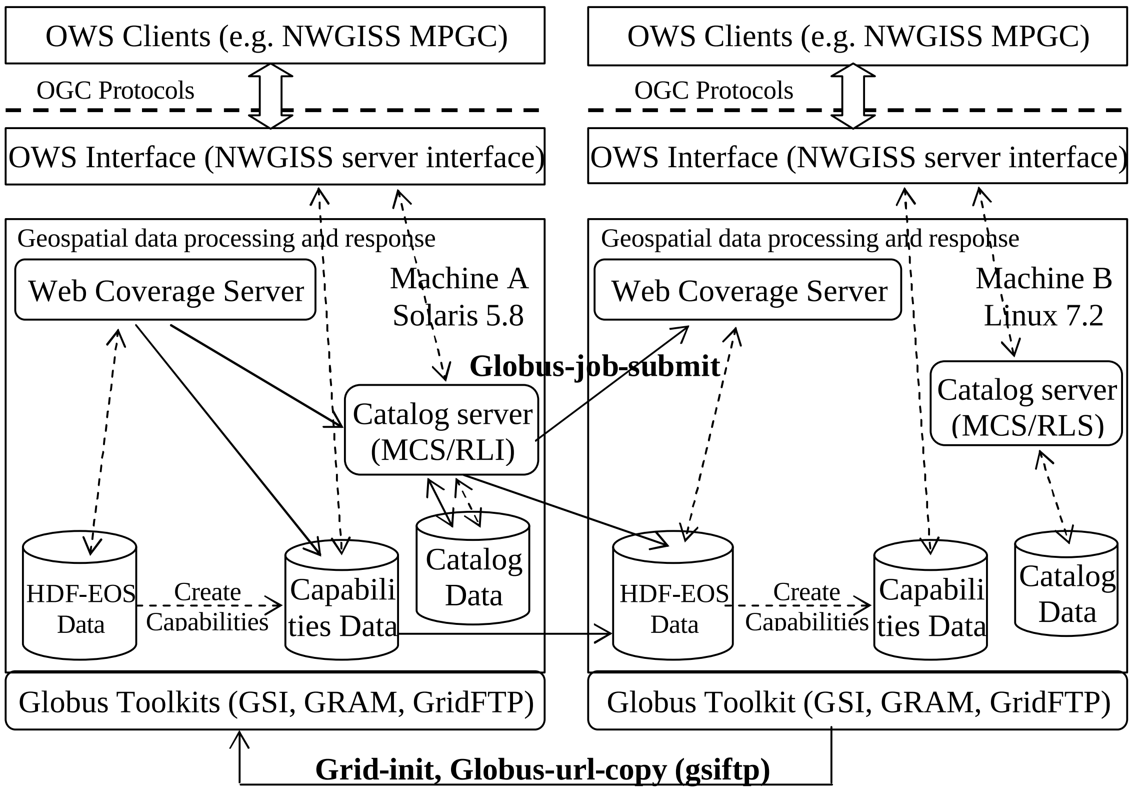

OWS Clients (e.g. NWGISS MPGC)
OGC Protocols
OWS Interface (NWGISS server interface)
Geospatial data processing and response
Web Coverage Server
Machine A
Solaris 5.8
Globus-job-submit
Catalog server (MCS/RLI)
HDF-EOS Data
Create Capabilities
Capabilities Data
Catalog Data
Globus Toolkits (GSI, GRAM, GridFTP)
OWS Clients (e.g. NWGISS MPGC)
OGC Protocols
OWS Interface (NWGISS server interface)
Geospatial data processing and response
Web Coverage Server
Machine B
Linux 7.2
Catalog server (MCS/RLS)
HDF-EOS Data
Create Capabilities
Capabilities Data
Catalog Data
Globus Toolkit (GSI, GRAM, GridFTP)
Grid-init, Globus-url-copy (gsiftp)

Fig. 3 System architecture and the simplified data flow demonstrating the integration (request from machine A to machine B).
- - - - - Broken lines show internal requests of NWGISS.
——— Solid lines show requests related to Globus.

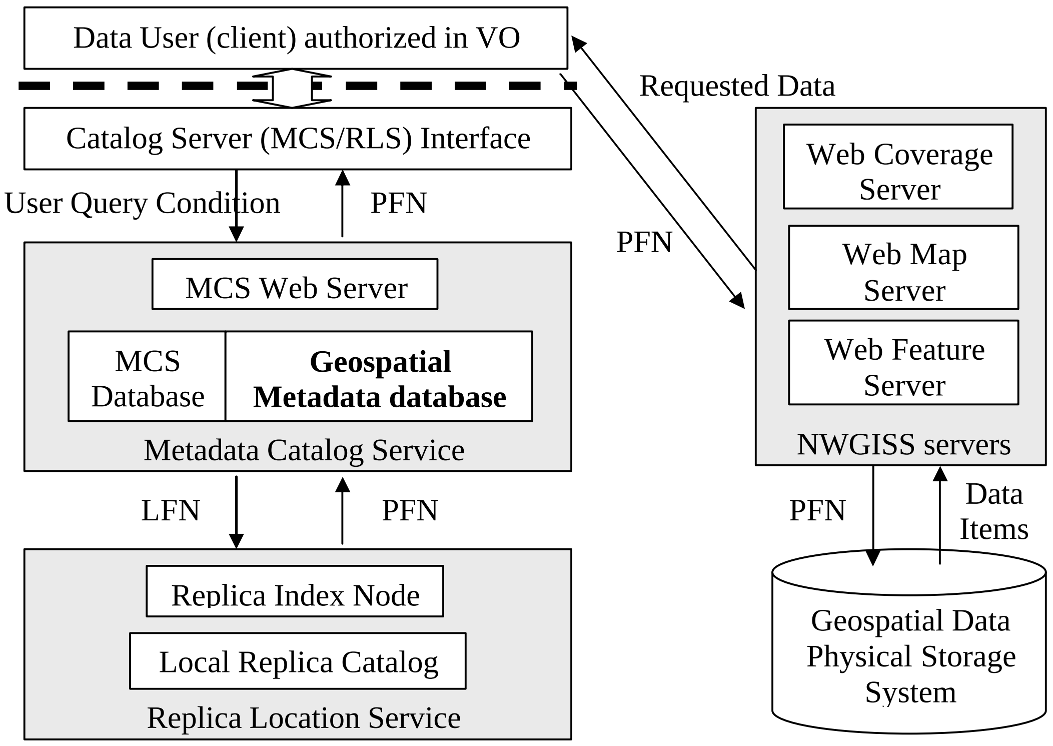
A. *The Map Server*: The map server enables GIS clients to access HDF-EOS data as maps. Currently the NWGISS map server complies with OGC WMS version 1.1.0. The OGC specification defines three interfaces, namely GetCapabilities, GetMap, and GetFeatureInfo. All three interfaces have been implemented and all three HDF-EOS data models (Grid, Point, and Swath) are supported.
B. *Coverage Server*: the OGC Web Coverage Service (WCS) specification is designed to enable GIS clients to access multi-dimensional, multi-temporal geospatial data. WCS defines three interface protocols: getCapabilities, getCoverage, and describeCoverageType. The NWGISS coverage server has implemented both versions 0.5 and 0.7 of the draft WCS specification. Three formats for encoding user-requested coverages are available in NWGISS, namely, HDF-EOS, GeoTIFF, and NITFF.
C. *Catalog Server*: Both WCS and WMS have the GetCapabilities protocol for clients to find geographic data/map and services available at servers. This protocol works nicely when a server has a small data archive. If the server has a lot of data, the capabilities description, which basically is a data catalog, is become very large. The catalog server allows GIS clients to search and find available geographic data and services in a NWGISS site based on the OGC catalog interoperability specification (CIS). The integration of this OGC catalog server with the MCS and RLS of Globus Toolkit makes the retrieval of geospatial data and services Grid enabled.
D. *Toolbox*: It contains tools for automated data ingestion and catalog creation. Currently, two types of tools are provided: the format conversion tools and XML capabilities creation tools. A third type of tools, the catalog creation tools, will be provided later.
E. *The Multi-protocol Geoinformation Client (MPGC)*: It enables the access of multi-dimensional and multi-temporal geospatial data from multiple coverage servers in the form that exactly matches users' requirements, regardless of the original forms of the data in the servers. The easy-to-use and easy-to-install web Java client enables GIS users to easily access the HDF-EOS data as well as other formatted data in a unified way, greatly enhancing the interoperability and public use of EOS data. Coupled with the NWGISS Server packages, MPGC makes HDF-EOS data available to GIS users based on the Open GIS Consortium's (OGC) interoperability protocols

The details about the Globus Toolkit can be found on the website: http://www.globus.org.

4.3 Data Requests and Data Flow Description

In our development and testing environment at LAITS, all three machines in the VO have been installed with Globus Toolkit and populated with geospatial data in HDF-EOS format. The Grid-enabled WCS, WMS and Catalog servers of NWGISS are installed at Laits and Llinux1. Users can send data and service requests from any OGC-compliant client (e.g., MPGC) through the Internet. NWGISS servers receive user requests and process the data on archives to produce data products on users' behavior. The most common data service functions include spatial, temporal, and parameter-based subsetting and subsampling, georectification, reprojection, reformatting. Those functions can be combined to produce data products that exactly match user requirements on data.

In our implementation, NWGISS servers at the machine that receives the user request play the major role in responding to user requests. Firstly, a user sends a query to the catalog server for searching and finding data through a client. User's requirements on the data are expressed in the query. The catalog server executes the query by searching the geospatial metadata database and MCS database to obtain matched logical file names (LFN). Then, the catalog server obtains the physical file names (PFN) through sending the logical file names to the RLS. The matched physical filenames and related metadata are sent back the client.



LFN: Logical File Name; PFN: Physical File Name

Fig. 4 Integration mechanism of Globus MCS with geospatial metadata

Secondly, after the user receives the search results, the user may choose to obtain the data by sending a data access request to one of NWGISS servers (e.g., WCS server). The WCS server firstly parses query to find the physical file name contained in the query. Then it uses the name to judge if the datasets reside in the local machine. If datasets exist locally, the data services of NWGISS at Laits will process the geospatial data sets locally, and return the results to user. If the datasets are not available locally, but available on any other Grid nodes in the VO, then the server on Laits will send a data request by this physical file name to the Grid nodes which host the data. Some of these Grid nodes may have only data but no services while others may have both. If the node (e.g. Llinux1) contains only the requested data, not the service, the data service on Laits will initialize Grid (grid-proxy-init) both at Laits and at Llinux1. Then it invokes the Grid secure copy function (globus-url-copy) at Laits to copy data from Llinux1 to Laits. After the data are copied, the NWGISS server at Laits will complete the user requests. If the site (e.g. Llinux2) has both data and requested services, the server on Laits will initialize Grid at both Laits and Llinux2, and then call the Grid job submitting function (globusrun) to submit user request to Grid services at Llinux2. Grid services will invoke the NWGISS data services at Llinux2 to complete the user request. The results will be copied back to Laits by using the Grid secure copy mechanism. The second case will significantly reduce the data traffic since the data reduction is done at the machine where the original dataset resides. All of the above processing is transparent to user. Figure 4 shows the data and the request/response flows.

5 Conclusions and Future Work

Our preliminary work and initial success have extended the applications of Grid technology to the EO community and also made OGC technology Grid enabled. Our experience with the project indicates that Grid technology has great potential for applications in the geospatial disciplines. By using Grid software as the foundation for geospatial data infrastructure, we achieve secure sharing of geospatial data and associated processing while providing common OGC interfaces and services to users. This integration takes advantages of both Grid and OGC technologies. It provides the user community a standard, disciplinary specific access to a huge volume of geospatial data available at space agencies and managed by Grid while shielding the details of Grid infrastructure underneath. As one of the technology prototypes for NASA ESE, the results from this project will be used by EOS Data Pools at NASA DAACs for securely sharing geospatial data and resources. Also, as one of the NASA application projects for CEOS Grid testbed, our experience and software will be shared with other participating space agencies. The NWGISS software mentioned in this paper can be downloaded at http://www.laits.gmu.edu.

The next step in our research will be the integration of OGC Web Registry Service (WRS) and Grid catalog systems for providing geospatial-specific OGC-compliable and Grid-enabled catalog services and enabling the virtual geospatial data services. Because of the nature of geospatial data and services, the current Grid metadata catalog system is not good enough for geospatial applications. Integration will provide a better, geospatial-specific catalog services for effective search and discovery of geospatial data and services managed by Grid.

Acknowledgement

This project was supported mainly by grants from the NASA Earth Science Data and Information System Project (ESDISP) and NASA Earth Science Technology Office (ESTO). Additional funding was provided by the Open GIS Consortium (OGC) for the development of NWGISS coverage server as a part of OGC WMT II and OWS-I.

References

[1] Ian Foster, Carl Kesselman and Steven Tuecke. The Anatomy of the Grid – Enabling Scalable Virtual Organizations. Intl. J. of High Performance Computing Applications, 15(3), 200-222, 2001.

[2] Ian Foster, Carl Kesselman, Jeffrey M. Nick and Steven Tuecke. The Physiology of the Grid: An open Grid services architecture for distributed systems integration. Feb. 17, 2002.

[3] Globus Toolkit. http://www.globus.org, Jun. 2003.

[4] Liping Di, Wenli Yang, Meixia Deng, D. Deng and K. McDonald. The prototype NASA HDF-EOS Web GIS Software Suite (NWGISS), Proceedings of the NASA Earth Science Technologies Conference. Greenbelt, Maryland. August 28-30, 2001 (CD-ROM, 4pp).

[5] OGC. OGC Interoperability Program. http://www.opengis.org/ogcInterop.htm, Mar. 2003.

[6] CEOS-Grid Task Team. V1.0 CEOS-Grid Interim Task Team Project Plan, Sep. 20 2002.

[7] CEOS Grid Tech Wiki. http://grid-tech.ceos.org/gridwiki, Feb. 2003.

[8] Liping Di and Ken McDonald. Next Generation Data and Information Systems for Earth Sciences Research, In Proceedings of the First International Symposium on Digital Earth, Volumn I. Science Press, Beijing, China, p92-101. Nov. 1999.

[9] Liping Di, Wenli Yang, Meixia Deng, D. Deng and K. McDonald. Interoperable Access of Remote Sensing Data through NWGISS, In Proceedings of IGARSS 2002. Toronto, Canada. Jun. 2002.

[10] The Metadata Catalog Service (MCS). http://gaul.isi.edu/mcs/, Jun. 2003.

[11] The Replica Location Service (RLS). http://www.globus.org/rls/, Jun. 2003.

[12] NASA. Proposed ECS Core Metadata Standard Release 2.0. http://edhs1.gsfc.nasa.gov/waisdata/docsw/pdf/tp4200105.pdf, Dec. 1994.

[13] OGC. The OpenGIS™ Abstract Specification, Topic 11: OpenGIS(tm) Metadata (ISO/TC 211 DIS 19115). http://www.opengis.org/techno/abstract/01-111.pdf, May 2001.

[14] FGDC. Content Standard for Digital Geospatial Metadata: Extensions for Remote Sensing Metadata. http://www.fgdc.gov/standards/status/ csdgm_rs_ex.html, Dec. 2002.

[15] OGC. Introduction to OGC Web Services. A. Doyle and C. Reed eds. 2001. http://ip.opengis.org/ows/index.html.

[16] OGC. Web Coverage Service (WCS), versions 0.5, 0.6, 0.7. J. Evans eds. 2000, 2001, 2002. http://www.opengis.org/.

Enhanced Product Generation at NASA Data Centers Through Grid Technology

Bruce R. Barkstrom+, Thomas H. Hinke*, Shradha Gavali* and William J. Seufzer!

*** NASA Ames Research Center, + NASA Langley Research Center, !Science Applications International Corporation, Hampton, VA**

Abstract

This paper describes how grid technology can support the ability of NASA data centers to provide customized data products. A combination of grid technology and commodity processors are proposed to provide the bandwidth necessary to perform customized processing of data, with customized data subsetting providing the initial example. This customized subsetting engine can be used to support a new type of subsetting, called phenomena-based subsetting, where data is subsetted based on its association with some phenomena, such as mesoscale convective systems or hurricanes. This concept is expanded to allow the phenomena to be detected in one type of data, with the subsetting requirements transmitted to the subsetting engine to subset a different type of data. The subsetting requirements are generated by a data mining system and transmitted to the subsetter in the form of an XML feature index that describes the spatial and temporal extent of the phenomena. For this work, a grid-based mining system called the Grid Miner is used to identify the phenomena and generate the feature index. This paper discusses the value of grid technology in facilitating the development of a high performance customized product processing and the coupling of a grid mining system to support phenomena-based subsetting.

1. Introduction

NASA has a number of archives that hold large amounts of data generated by various satellites. One group of archives is dedicated to the storage of Earth science data, with the nine archives associated with the Earth Observing System Data and Information System (EOSDIS) holding the bulk of the NASA's Earth science data. Currently users can access one of the EOSDIS archives through various data gateways spread around the world that are all listed on http://redhook.gsfc.nasa.gov/~imswww/pub/imswelcome/imswwwsites.html. Through these gateways, the user can perform a search for data across all of the EOSDIS archives, based on the name of the instrument that was used to capture the data (e.g., TRMM TMI -- Tropical Rainfall Measuring Mission Microwave Imager or CERES – Cloud and Earth's Radiant Energy System), various keywords such as parameters (e.g., Cloud Liquid Water/Ice), processing level, and numerous others. The result of a search is a list of data sets that satisfy the search criteria. From this list the user can select a particular data set and view the file that can be ordered for that data set. The smallest orderable quantity of data is called a granule and may cover a fraction of an orbit, a single orbit, a single day or some longer period. Under current practice, the user can place his order though the data gateway's web page or through web interfaces maintained by the individual data centers. If only a small amount of data is ordered (say, a few gigabytes or less) then the data will be pulled from the archive in a few hours and placed on an FTP server. If a larger amount of data is ordered, then the data will be shipped to the user on some media (such as digital tape).

This paper is investigating technology that will improve this process by allowing the user to specify some custom processing to be performed at the data center that will transform the data into a form that is more useable to him or her. Under this approach, rather than getting the standard files of data, and then having to perform custom processing at the user's site, the user can request that the data center produce some custom products. In the simplest case, this custom product can involve subsetting the data. For example, each granule might cover an entire orbit or an entire day (e.g., 14 orbits for some satellites), while the user is interested only in data that covers California. Alternatively, the user may be interested in time series data that covers California over the last three years. If the science team producing the data has not already done the selection, this interest could require extensive processing to go though three years of data and gather only the data that covers California. The user may also want only data that is associated with a mesoscale convective system (severe storm), a hurricane or a volcanic eruption. This involves phenomena-based subsetting which requires that a significant amount of processing be expended by the data center if these types of customized product requests are to be handled in the future.

This paper describes research and development work to improve the order production process by performing custom subsetting of the data prior to delivery to the requestor through the user of grid technology to support parallel subsetting of data. The data center portion of this work is being performed at the NASA Langley Research Center Atmospheric Sciences Data Center, which is one of the EOSDIS archives. The data mining portion of this work to support phenomena-based subsetting is being performed by the NASA Ames Research Center.

The next section of this paper will describe the work at the Atmospheric Sciences Data Center to use a combination of grid technologies and commodity processors to provide a high performance engine to perform customized product processing, with subsetting being the initial application. Section 3 will present the approach being used to tie a Grid Miner to the data center's subsetter to support phenomena-based subsetting. Section 4 will conclude with a discussion of how grid technologies are facilitating the development of this system.

2. Customized Product Processing Through Grid Technology

At present, EOSDIS data is stored in robotic tape archives, with data read rates for a single file of about 10 MBps. This read rate limits the amount of data a user can access in a month to about 26 TB – assuming that there are no glitches and that the user makes no mistakes. Unfortunately, users who want to work with long time series, say five years, need access to several times this amount of data and are likely to have to iterate with their subsetting algorithms three or four times. With this rate of access, it is nearly impossible for users to manipulate and discover interesting items because it may take several years to produce well-validated subsets. In order to maximize use of the data for this type of user, it will be necessary to massively increase the rate at which data can be run through user programs. As a useful goal, we would like to be able to obtain a throughput rate of about 400 MBps or about 1 PB per month. This would allow the user with 26 TB of data to iterate through the entire data set in about one day (or less), allowing much improved ability to remove processing artifacts and errors.

This throughput can be achieved by transferring the data to a large number of disks, with an appropriate number of CPU's to perform the data filtering and subsetting operations. To move toward this goal, the Atmospheric Sciences Data Center at the NASA Langley Research Center will start by transferring a modest number of files out of the archive for storage on disk. Then, they will provide a simple subsetting program that allows users to interactively build a script of instructions similar to what one might do with a relational database. The intent is to allow this program to create a data structure similar to the rows with fields in a database and then to perform four basic functions on the data:

- Use simple queries on the fields to select rows into the subset
- Calculate simple statistics on the fields in the selected rows
- Visualize the relationships between the fields – at high display rates that would allow millions of data points to be plotted on a user's browser in under thirty seconds
- Create transformed variables that are placed in new columns of the in-memory data structure

This approach is expected to deliver useful subsets for various NASA science teams. In addition, we intend to explore the use of commodity computing hardware and highly reliable software using grid computing technology to reduce the overall cost of ownership.

Figure 1 shows the schematic architecture of the Atmospheric Sciences Data Center's grid-based data access and production system. This system will initially support parallel subsetting, but could just as easily support parallel reformatting of the data to transform it from its archived format into a format desired by the data requestor, or perform some other processing to customize the data for the user. It is expected that both internal and external data users will interact with this system through a web-services style of interface. At the same time, it is important for the system to be sufficiently automated that intelligent agents could provide files for ingesting data and for obtaining data through the distribution interface – expected to be push and pull FTP. Internally, both the CPU's and the Storage Nodes should be designed as peer-to-peer daemons to increase system reliability and to improve security.

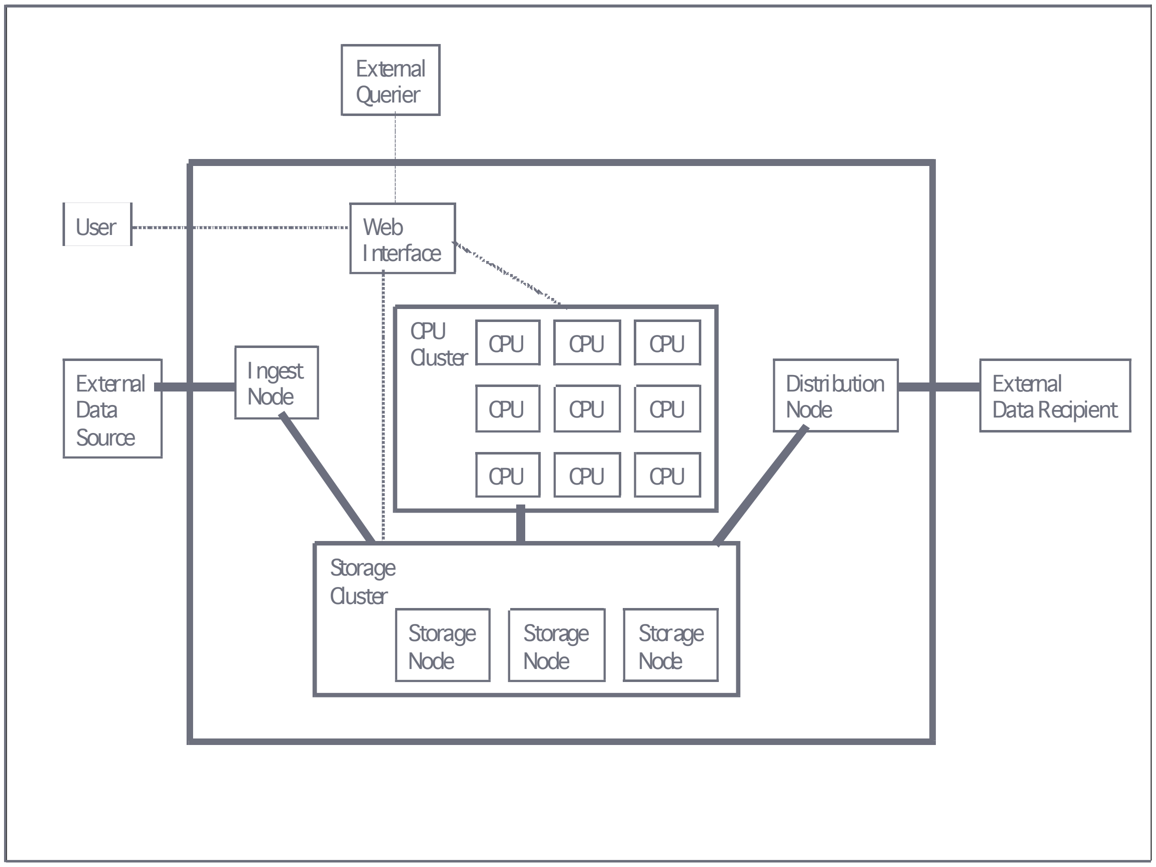


Figure 1. Schematic Diagram for Architecture of Grid-Enabled Data Access and Production System. Heavy lines show high-bandwidth data transfer paths; lighter (dotted) lines show command paths. Both CPU's and Storage Nodes are intended to operate autonomously to increase reliability and security of this architecture.

The NASA Langley Atmospheric Sciences Data Center, has about 500 TB of data currently stored in two systems - both having most of the data on AMASS tape storage systems. Data production uses SGI machines - mainly Origin 2000's or 3800's - with about 150 CPU's in each of the two systems. The original architecture and much of the hardware comes from the early to mid-1990's, when both the data volume and the I/O were perceived as very high in terms of the typical needs of the IT community. With recent developments in hardware and software as well as budgetary pressures, consideration is being given to moving to commodity hardware and open source software including the following:

- Linux for the OS and standard package support
- PostgreSQL for storing and searching our metadata
- Perl and Python for production scripting needs
- gcc for compiler support in Ada95 and Java, Intel ® for C, C++, and FORTRAN
- Hierarchical Data Format (an Open Source format from the HDF Group at NCSA - see http://hdf.ncsa.uiuc.edu) and
- NetCDF (another formatting library from Unidata - see http://www.unidata.ucar.edu/packages/netcdf/)

- Grid computing tools, notably Storage Resource Broker (SRB) from the San Diego Supercomputing Center (see http://www.npaci.edu/DICE/SRB/)
- Condor-G from the University of Wisconsin (see http://www.cs.wisc.edu/condor/condorg/), and
- Globus toolkit from Argonne National Labs (see http://www.globus.org/toolkit/)

The approach described in this follows the recommendations in the recent report of the National Research Council on Government Data Centers [NRC 2003], which recommends that data centers consider moving from tape storage to disk and incorporating more “bleeding edge” technologies.

While we do not present a detailed architecture here, recent advances in versioning theory for Earth science data products [Barkstrom 2003] will support database use for holding and querying metadata, as well as the ability to perform provenance tracking from “cradle to grave” on the data that will be stored in this system. These features also accord well with the recommendations of the NRC.

The initial application of most interest to the Atmospheric Sciences Data Center is finding ways of subsetting and finding phenomena (like mesoscale convective systems or hurricanes) in the data we have in the Center. From a variety of perspectives, this application suggests the need to be able to stream large files (say 200 MB each) through filtering programs at a rate of about 1 PB/month. This figure would support both migration of large data collections and a moderate number of climatological data users who need reasonable turnaround (say 1 week or less) in working with five or ten years of data records. Assuming that the major bottleneck lies in the I/O, it may mean that we need to do some additional architectural work on how we store the files, but this will await experience with the system being described in this paper.

Perhaps the simplest way of describing the systems currently used by the Atmospheric Sciences Data Center is that they were engineered to deal with data storage - under the assumption that data users wanted only a few files in each access and that the accesses were more or less randomly distributed across all of the files. This access pattern makes for a reasonable match between robotic tape storage, the data storage strategy, and the user access pattern.

In the data access scenario that is driving the new, grid-based subsetting work, users want to stream through data files that span a period of time, perhaps five years or more. Under this scenario, the data center has a need for content based searching with a throughput requirement of approximately 1 PB per month. In some early experiments, the primary bottleneck for this kind of user scenario is in the ability to get data out of the input side of a computational node and through a filtering program in the computer. There is currently no perceived need to worry about parallelizing the individual program code - computation isn't the bottleneck, data bandwidth is. That almost certainly means that the new architecture needs to spread the data out in multiple disks and run it through as many CPU's as we can make available, using coarse-grained parallel processing, with for example, each processor working on data for a different day or orbit.

Since data may be located in different data centers, we anticipate the need to subset related data at multiple data centers. To minimize the amount of data flowing between centers, we would be better off exchanging "pointers" or feature indexes for features discovered in one data center that need to be used to subset data in another data center. This approach should reduce the required network traffic by several orders of magnitude over an approach in which data from multiple centers is all moved to the same location to perform such multi-center, mult-data-set subsetting. It also allows data centers to retain autonomy with respect to their collections at the same time it increases services to users, since the feature indexes that serve as an additional source of metadata to help new user communities find objects of interest can be replicated at both data centers for a very small storage overhead. We expand on this approach in the section that follows.

3. Phenomena-Based Subsetting

Phenomena-based subsetting is a concept that supports the desire to perform research on data from a number of different datasets that are all associated with the same phenomena. In order to support phenomena-based subsetting, the spatial and temporal location of the phenomena of interest must be determined. Since this could involve sifting through a large amount of data to locate phenomena of interest, it represents a potentially good application for data mining, which has been defined as "… the process by which information and knowledge are extracted from a potentially large volume of data using techniques that go beyond a simple search through the data." [Data Mining Workshop 1999] Scientific data mining in general, and Earth Science data mining in particular is characterized by the need to mine possibly large amounts of data that has been captured by satellite-based remote sensors. An example is data from the TMI (TRMM Microwave Imager) instrument, which consists of approximately 230 megabytes (uncompressed) of data per day. Other satellite data can be even more voluminous due to its finer resolution.

This research uses a software system called the Grid Miner [Hinke 2000b] that was developed at the NASA Ames Research Center. The Grid Miner is a grid-enabled version of the stand-alone ADaM data mining system that was developed at the University of Alabama in Huntsville under a NASA research grant [Hinke 2000a, Hinke 1997a]. The Grid Miner is an agent-based mining system in which mining agents are sent to processors on the grid to mine remote data that is accessible from the grid and described in a mining database that has been pre-loaded with the URLs of data to be mined.

Figure 2 shows the architecture that is used to perform phenomena-based subsetting. A user invokes the miner by staging "thin" mining agents to the grid processors that are to support the mining, along with the mining plan (written by the user) that is to guide the mining for the desired phenomena. Based on the mining plan provided, these thin agents are able to grow in capability, through the acquisition of the necessary mining operations required to execute the plan. Each of the mining operations is configured as a shared library executable, with one operation per executable file. As the thin mining agent executes the mining plan, it identifies the operations that are to be used and then uses the grid to transfer the needed shared library executables from a mining operator repository to the grid processors where the mining is to be performed. The use of thin agents minimizes the size of the agent code that needs to be transferred. This approach of dynamically acquiring needed mining operations means that mining operations could be retrieved from multiple operator repositories, some public, some private and perhaps some for a fee, although this multi-site repository represents future work.

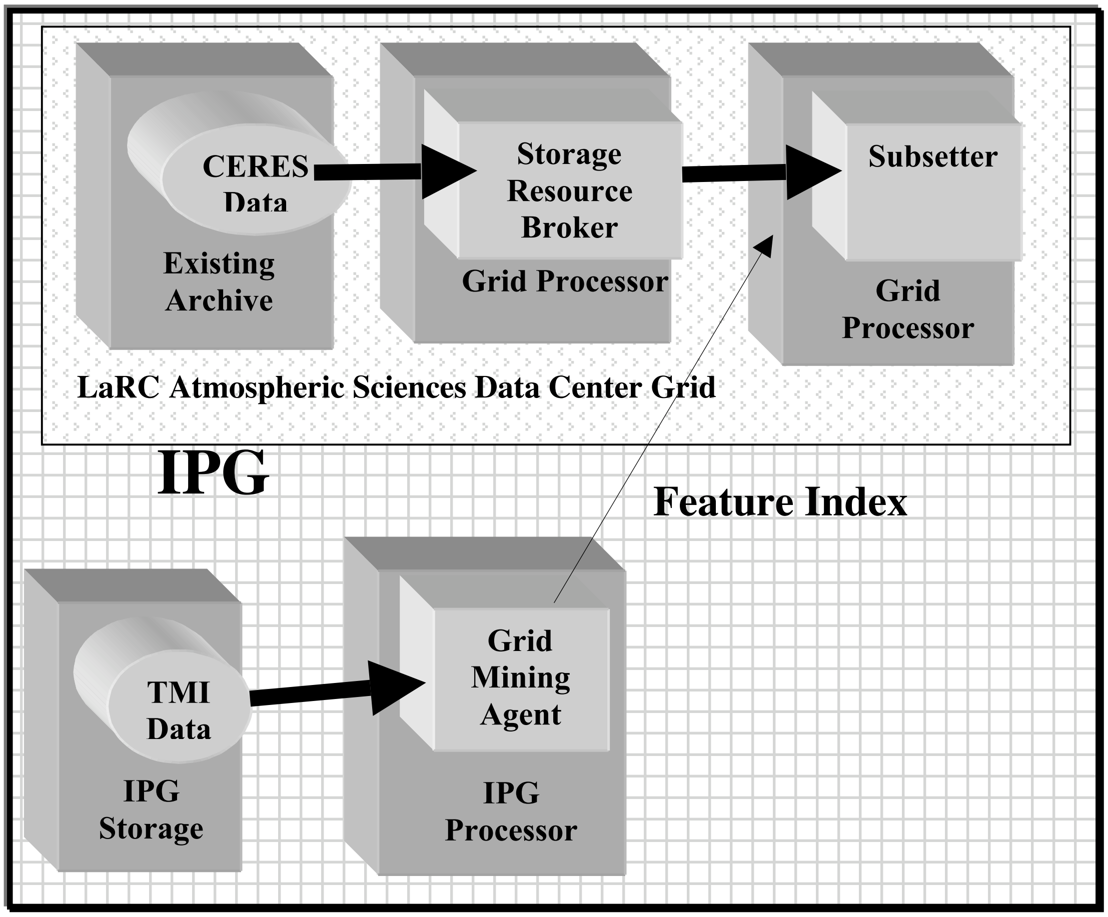


Figure 2. Schematic Diagram for Architecture of Phenomana-based Subsetting. Heavy lines show high-bandwidth data transfer paths; lighter show the path of the XML feature index. Note that this architecture uses both the IPG and an internal Atmospheric Scienes Data Center Grid.

Once the thin agent has grown to have the necessary mining operations to perform the mining plan, the mining agent contacts the mining database to acquire the URLs of the files to be mined. Using the grid, these files are then transferred to the mining site and the mining is performed as specified in the mining plan.

To support phenomena-based subsetting, the Grid Miner will mine the data for the desired phenomena and when found, will circumscribe the phenomena with a convex hull polygon and associated metadata to specify not only the spatial extent of the phenomena, but also its temporal location. These will be output as an XML document in the following form:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<polygon_list>
```

```
<polygon>
<date_time>
<dateYYYY-MM-DD> 1998-01-01 </dateYYYY-MM-DD>
<time_of_day_in_second> 13000</time_of_day_in_second>
</date_time>
<size_in_square_km> 2050.781250 </size_in_square_km>
<region_type> 2</region_type>
<vertices>
<number_of_vertices> 7</number_of_vertices>
<vertex> <latitude> LATw </latitude> <longitude> LONGx</longitude>
</vertex>
<vertex> <latitude> LATy</latitude> <longitude> LONGz</longitude>
.
.
.
</vertices>
</polygon>
<polygon>
.
.
</polygon>
</polygon_list>
```

For the initial phenomena for this work, we are searching for mesoscale convective systems within passive microwave data from the TMI instrument that was obtained from the Goddard Space Flight Center's Distributed Active Archive Center. The mining operation used to search for mesoscale convecive systems was developed at the University of Alabama in Huntsville and uses an algorithm originally suggested in [Devlin 1995]. For the purposes of this experimental work, the TMI data is being stored on storage that is accessible from the NASA Information Power Grid (IPG). The Grid Miner is staged to IPG computational resources, which could be located anywhere on the IPG, with the data to be mined pulled from Ames' storage.

When the mining is completed, the XML document describing the spatial and temporal location of the phenomena of interest will be sent to the subsetting engine. Again, the use of XML for transferring information accords well with the NRC recommendations [NRC 2003]. The XML document includes indexes to the original pixels in the data files that contribute to the object identifications. Such object indices provide the ability to efficiently retrieve the original data belonging to the phenomenon, as well as building metadata for each instance. This approach allows us to develop a database of phenomena instances [Hinke 1997b] that should markedly increase the scientific community's ability to extend the value of its data resources, as suggested several years ago in [Barkstrom 1998].

For this initial work, based on the date and time information provided in the XML document, the appropriate CERES data will be accessed and fed into the subsetter, along with the polygon that describes the spatial and temporal areas that correspond to the mesoscale convective systems from the TMI data. The subsetter will then extract the CERES data that corresponds to the TMI-discovered mesoscale convective system.

It should be noted that in this case (and by intent) the two instruments (TMI and CERES) were both located on the same satellite. Thus, the subset date extracted from the CERES data will have both temporal and spatial congruence with the phenomena discovered in the TMI data. It is not always the case that desired instruments are located on the same satellite, which means that the phenomena detected in data from one satellite may have moved to a slightly different location in the data that is subset. How one would address this problem is beyond the scope of this paper.

4. The Value of Grid Support

Grid technology provides valuable support for the user-centric approach we have described:

- The Storage Resource Broker (SRB) provides a useful way of separating the details of the local storage from the logical structure of the files and directories, reducing the operator overhead associated with storage management, thereby reducing the total cost of ownership.
- The external interfaces to grid architectures, such as that shown in figure 1, provide reliable FTP with the grid-provided single-sign-on environment for connecting the subsetting with access to data stored on a grid-enabled storage system, such as the SRB, increasing the possibilities for fully automated, secure system access, again reducing the total cost of ownership.
- Provides a single-sign-on environment for running the grid miner of available grid resources and handling the transfer of both mining operators and data.
- Provides a single-sign-on environment for injecting feature indexes into the data center's subsetting engine.

5. Acknowledgements

We would like to acknowledge the fact that the Grid Miner is a modification of the ADaM data mining system that was developed by the University of Alabama in Huntsville, under a NASA research grant. While Thomas Hinke, one of the authors of this paper designed a data mining system that was a precursor to the ADaM data mining system and was one of the investigators involved in the development of ADaM, Dr. John Rushing was the principle architect and implementer of the ADaM data mining system. Drs. Sara Graves and Hegge Ranganath also were significant contributors to the development of ADaM. We would also like to acknowledge the valuable comments that William Thigpen and Tony Lisotta at the NASA Ames Research Center made on this paper.

6. References

[Barkstrom 1998] Barkstrom, B. R., “Digital Archive Issues from the Perspective of an Earth Science Data Producer”, paper presented at the *Digital Archive Directions (DADS) Workshop*, June 22-26, 1998, available at http://ssdoo.gsfc.nasa.gov/nost/isoas/dads/dads21b.html.

[Barkstrom 2003] Barkstrom, B. R., "Data Product Configuration Management and Versioning in Large-Scale Production of Satellite Scientific Data," in B. Westfechtel and A. van der Hoek (eds.), SCM 2001/2003, *Lecture Notes in Computer Science* 2649, pp. 118-133, 2003.

[Devlin 1995] Devlin, K.I. Application of the 85 Ghz Ice Scattering Signature to a Global Study of Mesoscale Convective Systems. Master's thesis, Meteorology, Texas A&M University, August 1995.

[Hinke 2000a] Hinke, Thomas H., John Rushing, Heggere Ranganath and Sara J. Graves, "Techniques and Experience in Mining Remotely Sensed Satellite Data," *Artificial Intelligence Review: Issues on the application of data mining* , Vol. 14, No. 6, December 2000, pp. 503-531.

[Hinke 2000b] Hinke, Thomas H. and Jason Novotny, "Data Mining on NASA's Information Power Grid," *Proceedings Ninth IEEE International Symposium on High Performance Distributed Computing* , Pittsburgh, Pennsylvania, August, 2000.

[Hinke 1997a] Hinke, Thomas H., John Rushing, Shalini Kansal, Sara J. Graves, Heggere Ranganath and Evans Criswell, "Eureka Phenomena Discovery and Phenomena Mining System," *Proceedings: 13th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Ocenography and Hydrology* , Long Beach, California, February, 1997.

[Hinke 1997b] Hinke, Thomas H., John Rushing, Shalini Kansal, Sara J. Graves and Heggere Ranganath. "For Scientific Data Discovery: Why Can't the Arhive be More Like the Web," *Proceedings Ninth International Conference on Scientific Database Management*, Evergreen State College, Olympia, Washington, August, 1997.

[Mining Workshop 1999] *NASA Workshop on the Issues in the Application of Data Mining to Scientific Data*. 1999, NASA Goddard Space Flight Center: http://www.cs.uah.edu/NASA_Mining/DMFinalReport.pdf.

[NRC 2003] *Government Data Centers: Meeting Increasing Demands*. 2003, National Research Council of the National Academies, Washington, DC, http://www.nap.edu.

XDB-IPG: An Extensible Database Architecture for an Information Grid of Heterogeneous and Distributed Information Resources

David A. Maluf Ph.D.[1], David G. Bell Ph.D.[2], Chris Knight[1], Peter Tran[3], Tracy La[4], Jenessa Lin[1], Bill McDermott[1], Barney Pell Ph.D.[2]

[1]*NASA Ames Research Center (NASA ARC)*
Mail Stop 269-4
Moffett Field, CA 94035-1000
USA

[2]*Research Institute for Advanced Computer Science at NASA ARC*
[3]*QSS Group Inc. at NASA ARC*
[4]*Computer Sciences Corporation at NASA ARC*

Abstract

This paper describes XDB-IPG, an open and extensible database architecture that supports efficient and flexible integration of heterogeneous and distributed information resources. XDB-IPG provides a novel “schema-less” database approach using a document-centered object-relational XML database mapping. This enables structured, unstructured, and semi-structured information to be integrated without requiring document schemas or translation tables. XDB-IPG utilizes existing international protocol standards of the World Wide Web Consortium Architecture Domain and the Internet Engineering Task Force, primarily HTTP, XML and WebDAV . Through a combination of these international protocols, universal database record identifiers, and physical address data types, XDB-IPG enables an unlimited number of desktops and distributed information sources to be linked seamlessly and efficiently into an information grid. XDB-IPG has been used to create a powerful set of novel information management systems for a variety of scientific and engineering applications.

Introduction – *The Information Grid*

The Information Power Grid (IPG) is the National Aeronautics and Space Administration’s (NASA) project for providing seamless access to distributed information resources regardless of location [29]. The project addresses three major categories of distributed resources: 1) hardware resources, such as super computers and scientific instruments, 2) software resources, such as teamwork and CAD programs, and 3) data resources, such as data archives and document databases.

This paper is concerned with data resources. While much recent work in this area is focused on access to structured data archives, our focus is on integrating structured, semi-structured, and unstructured information. As with many enterprises, information and information processes services at NASA are highly distributed. NASA and its contractors have hundreds of databases with millions of records and hundreds of desktop computers with millions of files. The formats and structures of the information are diverse with hundreds of file-types and hundreds of thousands of explicit and implicit structures. The decision making applications that utilize this information are numerous with hundreds of procedures and guidelines and hundreds of thousands of diverse work practices. We seek to create an Information Grid that will provide seamless integration of these distributed heterogeneous information resources for distributed heterogeneous scientific and engineering applications.

XDB-IPG is a novel architecture that enables the creation of such an Information Grid. XDB-IPG is built upon three standards from the World Wide Web Consortium (W3C) Architecture Domain and the Internet Engineering Task Force: 1) HTTP: Hypertext Transfer Protocol – a successful and stable request/response protocol standard, 2) XML: Extensible Markup Language – A ubiquitous five-year old standard that defines a syntax for exchange of logically structured information on the web, and 3) WebDAV – A widely supported four-year old standard that defines HTTP extensions for distributed management of web resources. While the third of these standards was primarily designed for distributed authoring and versioning of web content, XDB-IPG leverages WebDAV for management of arbitrary information resources including information processing services.

Through a combination of these international protocols, universal database record identifiers, and physical address data types, XDB-IPG provides a number of capabilities for managing distributed and heterogeneous information resources such as the following:

- storing and retrieving information about resources using properties
- locking and unlocking resources to provide serialized access
- getting and putting information in heterogeneous formats
- copying, moving and organizing resources through hierarchy and network relations
- automatic decomposition of information into a query-able XML database
- context+content querying of information in the XML database
- sequencing workflows of information processing tasks
- seamless access to information in diverse formats and structures
- a common protocol for human and computer interface to grid services

XDB-IPG enables an unlimited number of desktops and distributed information sources to be linked seamlessly and efficiently into a highly scalable information grid as shown in Figure 1. XDB-IPG thus represents a flexible, high-throughput open architecture for managing, storing, and searching unstructured or semi-structured data. XDB-IPG provides automatic data management, storage, retrieval, and discovery [27] in transforming large quantities of highly complex and constantly changing heterogeneous data formats into a well-structured, common standard.

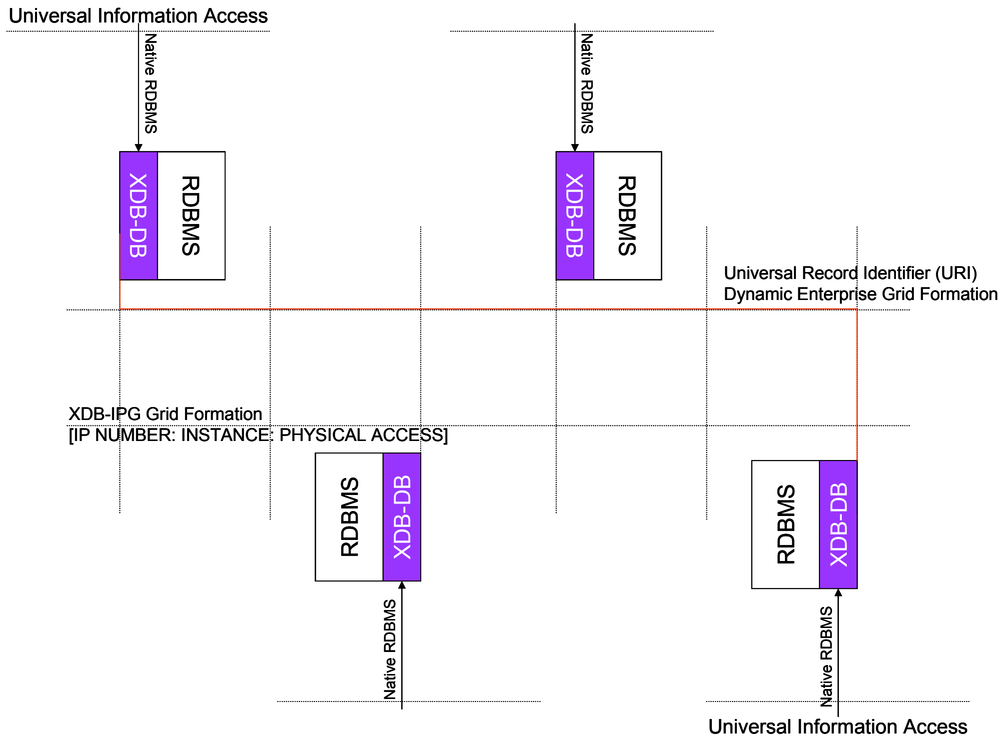

Universal Information Access
Native RDBMS
XDB-DB
RDBMS
Native RDBMS
XDB-DB
RDBMS
Universal Record Identifier (URI)
Dynamic Enterprise Grid Formation
XDB-IPG Grid Formation
[IP NUMBER: INSTANCE: PHYSICAL ACCESS]
RDBMS
XDB-DB
Native RDBMS
RDBMS
XDB-DB
Native RDBMS
Universal Information Access

Figure 1: Information Grid using the XDB-IPG architecture

In the next section, we provide background on the primary building blocks for XDB-IPG.

Object-Relational Database Technology

During the early years of database technology, there were two opposing research and development directions, namely the relational model originally formalized by Codd [1] in 1970 and the object-oriented, semantic database model [2][3]. The traditional relational model revolutionized the field by separating logical data representation from physical implementation. The relational model has been developed into a mature and proven database technology holding a majority stake of the commercial database market along with the official standardization of the Structured Query Language (SQL)[1] by ISO and ANSI committees.

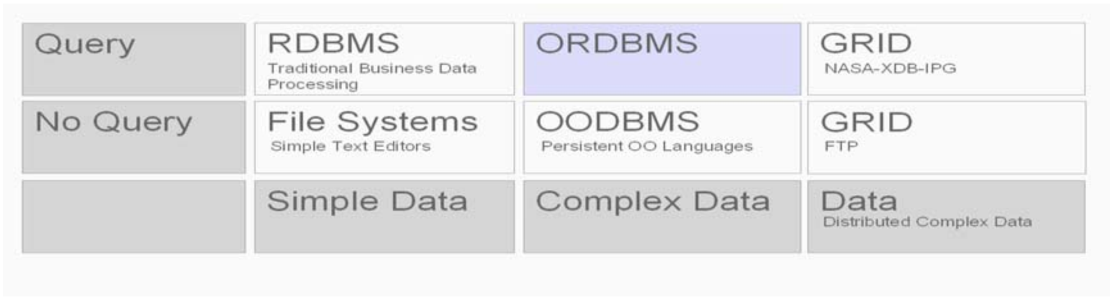
The semantic model leveraged from the object-oriented paradigm of programming languages, such as the availability of convenient data abstraction mechanisms, and the realization of the impedance mismatch [4] dilemma faced between the popular object-oriented programming languages and the underlining relational database management systems (RDBMS). Impedance mismatch here refers to the problem faced by both database programmers and application developers, in which the way the

[1] The Structured Query Language (SQL) is the relational standard defined by ANSI (the American National Standard Institute) in 1986 as SQL1 or SQL86 and revised and enhanced in 1992 as SQL2 or SQL92.

developers structure data is not the same as the way the database structures it. Therefore, the developers are required to write large and complex amounts of object-to-relational mapping code to convert data, which is being inserted into a tabular format the database can understand. Likewise, the developers must convert the relational information returned from the database into the object format developers require for their programs. Today, in order to solve the impedance mismatch problem and take advantage of these two popular database models, commercial enterprise database management systems (DBMS), have an integrated hybrid cooperative approach of an object-relational model [5].

The object-relational model takes the best practices of both relational and object-oriented, semantic views to decouple the complexity of handling massively rich data representations and their complex interrelationships. ORDBMS employs a data model that attempts to incorporate object-oriented features into traditional relational database systems. All database information is still stored within relations (tables), but some of the tabular attributes may have richer data structures. It was developed to solve some of the inadequacies associated with storing large and complex multimedia objects, such as audio, video, and image files, within traditional RDBMS. As an intermediate hybrid cooperative model, the ORDBMS combined the flexibility, scalability, and security of using existing relational systems along with extensible object-oriented features, such as data abstraction, encapsulation, inheritance, and polymorphism.

In order to understand the benefits of ORDBMS, a comparison of the other models need to be taken into account. The 2x3 database application classification matrix [6] shown in Figure 2 displays the six categories of DBMS applications—simple data without queries (file systems), simple data with queries (RDBMS), complex data without queries (OODBMS), complex data with queries (ORDBMS), distributed complex data without queries (Grid FTP), and distributed complex data with queries (Grid XDB-IPG). For the upper left-handed corner of the matrix, traditional business data processing, such as storing and managing employee information, with simple normalized attributes such as numbers (integers or floats) and character strings, usually needs to utilize SQL queries to retrieve relevant information. Thus, RDBMS is well suited for traditional business processing; but this model cannot query complex data such as word processing documents as if they were a structured database. The lower middle cell describes the use of persistent object-oriented languages to store complex data objects and their relationships. The lower middle cell represents OODBMS which either have very little SQL-like queries support or none at all. The upper middle with the light blue colored cell is well suited for complex and flexible database applications that need complex data creation, such as large objects to store word processing documents, and SQL queries to retrieve relevant information from within these documents. XDB-IPG is the upper right-handed corner with the light blue colored cell as indicated in Table below, and supports queries over complex data that is distributed.



Query	RDBMS Traditional Business Data Processing	ORDBMS	GRID NASA-XDB-IPG
No Query	File Systems Simple Text Editors	OODBMS Persistent OO Languages	GRID FTP
	Simple Data	Complex Data	Data Distributed Complex Data

Figure 2: Mapping the evolution of data complexity and database trends (adapted).

The main advantages of ORDBMS, are scalability, performance, and broad support by vendors. ORDBMS handle very large and complex applications. Most ORDBMS supports the SQL3 [10] specifications or its extended form. The two basic characteristics of SQL3 are crudely separated into its “relational features“ and its “object-oriented features”. The relational features for SQL3 consist of new data types [such as large objects or LOB and its variants]. The object-oriented features of SQL3 include structured user-defined types called abstract data types (ADT) [11][13] which can be hierarchical defined (inheritance feature), invocation routines called methods, and REF types that provides reference values for unique row objects defined by object identifier (OID) [13]. These new features are heavily exploited by XDB-IPG.

In order to take advantage of the object-relational (OR) model defined within an object-relational database system (ORDBMS) [5][6], a standard for common data representation and exchange is needed. Today, the emerging standard is the eXtensible Markup Language (XML) [7][8][9], commonly viewed to be the next generation of HTML for placing structure within documents.

Structuring with XML

XML is known as the next generation of HTML and a simplified subset of the Standard Generalized Markup Language (SGML)[1]. XML is both a semantic and structured markup language [7]. The basic principle behind XML is simple. A set of meaningful, user-defined tags surrounding the data elements describes a document’s structure as well as its meaning without describing how the document should be formatted [16]. This enables XML to be a well-suitable meta-markup language for handling loosely structured or *semi-structured data*, because the standard does not place any restrictions on the tags or the nesting relationships. Semi-structured data here refers to data that may be irregular or incomplete, and its structure can be rapidly changing and unpredictable [16]. Good examples of semi-

[1] The Standard Generalized Markup Language (SGML) is the official International Standard (ISO 8879) adopted by the world’s largest producers of documents, but is very complex. Both XML and HTML are subsets of SGML.

structured data are web pages and constantly changing word processing documents being modified on a weekly or monthly basis.

XML encoding, although more verbose than database tables or object definitions, provides the information in a more convenient and usable format from a data management perspective. In addition, the XML data can be transformed and rendered using simple eXtensible Stylesheet Language (XSL) specifications [8]. It can be validated against a set of grammar rules and logical definitions defined within the Document Type Definitions (DTDs) or XML Schema [19] much the same functionality as a traditional database schema.

Since XML is a document and not a data model, the ability to map XML-encoded information into a true data model is needed. XDB-IPG allows this to occur by employing a customizable data type definition structure defined by parsing dynamically the hierarchical model structure of XML data instead of any particular persistent schema representation. The customizable driver simulates the Document Object Model (DOM) Level 1 specifications [20] on parsing and decomposition of elements. The XDB-IPG driver is more effective on decomposition than most commercial DOM parsers, since it is less syntax sensitive and guarantees an output ("garbage in, garbage out"), when compared to most commercial parsers. The node data type format is based on a simplified variant of the Object Exchange Model (OEM) [28] researched at Stanford University, which is very similar to XML tags. The node data type contains an object identifier (node identifier) and the corresponding data type. Traditional object-relational mapping from XML to relational database schema models the data within the XML documents as a tree of objects that are specific to the data in the document [19]. In this model, element type with attributes, content, or complex element types are generally modeled as object classes. Element types with parsed character data (PCDATA) and attributes are modeled as scalar types. This model is then mapped to the relational database using traditional object-relational mapping techniques or via SQL3 object views. Therefore, classes are mapped to tables, scalar types are mapped to columns, and object-valued properties are mapped to key pairs (both primary and foreign). This traditional mapping model is limited since the object tree structure is different for each set of XML documents. On the other hand, the XDB-IPG SGML parser models the document itself (similar to the DOM), and its object tree structure is the same for all XML documents. Thus, XDB-IPG parser is designed to be independent of any particular XML document schemas and is termed to be schema-less.

Scalable and Efficient Linking of Distributed Resources

UNIVERSAL DATABASE RECORD IDENTIFIER (UDRI)

Universal Database Record Identifier (UDRI) is intended to be a subset to the Uniform Resource Locator (URL) and provide an extensible means for identifying universally database records. This specification of URI syntax and semantics is derived from concepts introduced by the World Wide Web global information initiative, and is described in "Universal Resource Identifiers in WWW" [RFC1630].

Universal access provides several benefits: it allows different types of databases to be used in the same context, even when the mechanisms used to access those resources may differ. It allows uniform semantic interpretation of common syntactic conventions across different types of records identifiers; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage on pre-existing, largely, and widely-used set of record identifiers.

The UDRI syntax is designed with a global transcribability and adaptability to URI standard as one of its main principles. A UDRI is a sequence of characters from a very limited set, i.e. the letters of the basic Latin alphabet, digits, and special characters. A UDRI may be represented in a variety of ways as a sequence of coded character set. The interpretation of a UDRI depends only on the characters used.

The UDRI syntax is a scheme derived from URI. In general, absolute URI are written as follows:

```
<scheme>:<scheme-specific-part>
```

An absolute URI contains the name of the scheme being used (<scheme>) followed by a colon (":") and then a string (the <scheme-specific part>) whose interpretation depends on the scheme. The XDB-IPG delineates the scheme to IPG where the scheme-specific-part delineates the ORDBMS static definitions.

RELATIONAL DATABASE ROWID SUPPORT

ROWID is a data type that stores either physical or logical addresses (row identifiers) to every row within the relational database [15]. Physical ROWIDs store the addresses of ordinary table records, clustered tables, indexes, table partitions and sub-partitions, index partitions and sub-partitions, while logical ROWIDs store the row addresses within indexed-organized tables for building secondary indexes. Each relational database table must have an implicit pseudo-column called ROWID, which can be retrieved by a simple SELECT query on the particular table and by bypass index search or table scan. Physical ROWIDs provide the fastest access to any record within an Oracle table with a single read block access, while logical ROWIDs provide fast access for highly volatile tables. A ROWID is guaranteed to not change unless the rows it references is deleted from the database.

XDB-IPG INTERFACES

The XDB-IPG API contains two major sets of interfaces: the first is the JDBC, ODBC, and C/C++ API for application writers, and the second is the lower-level SQL and corresponding server level procedure language API for driver writers. XDB-IPG drivers fit into one of four categories. Applications and servers (server-to-sever) can access XDB-IPG using standard compliant SQL technology-based drivers in particular for XDB-IPG core schema-less configuration.

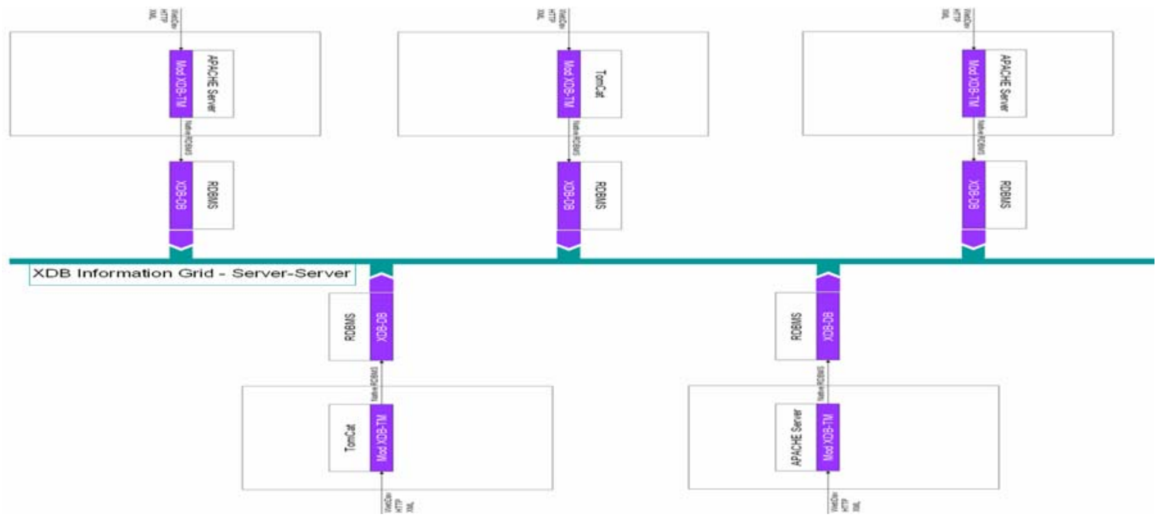


Figure 3: server to server mapping and providing a practical solution for intranet access.

Direct-to-Database: This style of driver converts JDBC, ODBC and C/C++ calls into the network protocol (or localized proprietary protocols) used directly by the data management system, allowing a direct call from the client machine to the server , server to server mapping and providing a practical solution for intranet access.

Transaction level Driver for Multipurpose Middleware: This style of driver translates JDBC, ODBC and C/C++ calls into the vendor's middleware protocol, which is then translated to a XDB-IPG by the middleware server. The middleware provides connectivity to many systems including file systems.

Benefits of XDB-IPG

With the XDB-IPG driver API, no configuration is required on the server side. With a driver written in the C/C++ programming language, all the information needed to map the information content is completely seamless as defined by the markup language or the Direct Access Virtual Information Directory object to be registered with XDB-IPG. The XDB-IPG driver library does not require special installation. It can be set to be automatically downloaded as part of the system that makes the XDB-IPG calls. This reduces the complexity of many data access tasks, and reducing both the upfront development costs for applications and the follow-on database administration maintenance costs.

The XDB-IPG API provides metadata access that enables the development of sophisticated applications that need to understand the underlying facilities and capabilities of specific information, a typical example is the metadata used in for Web-based Distributed Authoring and Versioning (WebDav). XDB-IPG technology also exploits the advantages of existing definition of Enterprise database management systems standard to manage information objects. The XDB-IPG API includes an even better way to identify and connect to a data source, using Direct Access Virtual Information Directory objects that make code even more portable and easier to maintain.

NASA is working with an array of companies in the industry to create and rapidly establish a NASA leadership API as the industry-standard, open interface for XDB-IPG applications to access databases. These leading middleware and tool vendors will provide support for XDB-IPG technology in many new products. This ensures that government, academic, and industrial entities can build portable applications while choosing from a wide range of competitive products for the solution best suited to their needs with little need to export and import the actual content and information.

Building Applications with XDB-IPG: Example 1 – Netmark

Netmark is an initial application that was built using the current XDB-IPG architecture. Netmark is comprised of a distributed, *information on demand* model for document management. Modules in the Netmark example are extensible and adaptable to different data sources. This example consists of (1) a set of interfaces to support various communication protocols (such as HTTP, WebDAV, FTP, RMI-IIOP and their secure variants), (2) an information bus to communicate between the client interfaces and the XDB-IPG core components, (3) the daemon process for automatic processing of inputs, (4) the XDB-IPG search on both document context plus content, (5) a set of extensible application programming interfaces (APIs), (6) and the XDB-IPG implementation for Oracle backend ORDBMS.

The three core components of XDB-IPG consist of the high-throughput information bus, the asynchronous daemon process, and the set of customizable and extensible APIs built on Java enterprise technology (J2EE) [22] and Oracle PL/SQL stored procedures and packages [13]. The XDB-IPG information bus allows virtually three major communication protocols heavily used today—namely HTTP/WebDAV and its secure HTTP web-based protocol, the WebDaV, and the new Remote Method Invocation (RMI) over Internet Inter-Orb Protocol [23][24] from the Object Management Group (OMG) Java-CORBA standards—to meet the information on demand model. The XDB-IPG daemon is a unidirectional asynchronous process to increase performance and scalability compared to traditional synchronous models, such as Remote Procedure Call (RPC) or Java RMI [23] mechanisms. The set of extensible C/C++, JNI and Java and PL/SQL APIs are used to enhance database access and data manipulation from most applications.

The information bus is comprised of a XDB-IPG module Apache HTTP web server and JNI interface integrated with Tomcat Java-based JSP/Servlet container engine [25]. It waits for incoming requests from the various clients, such as an uploaded word processing document from a web folder enabled browser. The bus performs a series of format conversions based on file-type. For instance, the XDB-IPG information bus will automatically convert a semi-structured Microsoft Word document into either a well structured HTML or XML format either on the client side or on the server side. A copy of the original word document, the converted HTML or XML file, and a series of dynamically generated configuration files containing additional pertinent but optional meta-data are added.

A novel aspect of Netmark is that the daemon process automatically converts heterogeneous documents such as word processing documents, presentations and spreadsheets into HTML or XML, the parser then stores the content in a connected node structure in the ‘schema-less’ database, and then a toolbar enables users to query the heterogeneous documents as if they were a database. By entering context+content keywords and phrases into the toolbar, users can retrieve the specific records of data from the heterogeneous documents (e.g., a section of a word processing document, or a cell of a spreadsheet).

CONCLUSION

XDB-IPG provides an extensible, schema-less, information on demand architecture that enables distributed and heterogeneous information resources to be integrated for a variety of scientific and engineering applications. XDB-IPG is a scalable, high-throughput open database framework for transforming unstructured or semi-structured documents into well-structured and standardized XML and/or HTML formats, and for managing, storing and retrieving unstructured and/or semi-structured data. Future plans for XDB-IPG include extending additional node data types, such as the SIMULATION node for handling other complex data sources.

ACKNOWLEDGEMENTS

XDB-IPG is the derivative of the NASA Information Technology-Base program, now part of the overall NASA Computing, Information, and Communication Technologies (CICT) program.

REFERENCES

[1] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks"; Communications of the ACM, Vol. 13, No. 6, pp. 377-387, June 1970

[2] R. Hull and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues"; ACM Computing Surveys, Vol. 19, No. 3, pp. 201-260, September 1987

[3] A. F. Cardenas and D. McLeod (Editors), "Research Foundations in Object-Oriented and Semantic Database Systems"; pp. 32-35, Prentice-Hall, 1990

[4] J. Chen and Q. Huang, "Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages"; Monash University, Australia, 1995

[5] R. S. Devarakonda, "Object-Relational Database Systems – The Road Ahead"; ACM Crossroads Student Magazine, February 2001,
http://www.acm.org/crossroads/xrds7-3/ordbms.html

[6] M. Stonebraker, "Object-Relational DBMS - The Next Wave", Informix Software (now part of the IBM Corp. family), Menlo Park, CA

[7] E. R. Harold, "XML: Extensible Markup Language"; pp. 23-55, IDG Books Worldwide, 1998

[8] Extensible Markup Language (XML) World Wide Web Consortium (W3C) Recommendation, October 2000,
http://www.w3c.org/TR/REC-xml

[9] "The XML Industry Portal"; XML Research Topics, 2001, http://www.xml.org/xml/resources_cover.shtml

[10] A. Eisenberg and J. Melton, "SQL:1999, formerly known as SQL3"; 1999,
http://www.incits.org/press/1996/pr96067.htm

[11] ISO/IEC 9075:1999, "Information Technology—Database Language—SQL—Part 1: Framework (SQL/Framework)", 1999

[12] American National Standards Institute (ANSI), http://web.ansi.org

[13] K. Loney and G. Koch, "Oracle 8i: The Complete Reference"; Oracle Press Osborne/McGraw-Hill, 10th Edition, pp. 69-85; pp. 574-580; pp. 616-644; pp. 646-663, 2000

[14] D. Megginson, "Structuring XML Documents"; pp. 43-70, Prentice-Hall, 1998

[15] Oracle Technology Network (OTN), "Oracle 8i Concepts Release 8.1.5"; Ch. 12 Built-In Data Types, pp. 9-

14, Oracle Corp. 1999,
http://technet.oracle.com/doc/server.815/a67781/c10datyp.htm

[16] J. Widom, "Data Management for XML Research Directions"; Stanford University, June 1999,
http://www-db.stanford.edu/lore/pubs/index.html

[17] Lore XML DBMS project, Stanford University, 1998, http://www-db.stanford.edu/lore/research/

[18] H. F. Korth and A. Silberschatz, "Database System Concepts"; pp. 173-200, McGraw-Hill, 1986

[19] R. Bourret, "Mapping DTD to Databases"; O'Reilly & Associates, 2000,
http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html

[20] L. Wood et al., "Document Object Model (DOM) Level 1 Specification", W3C Recommendation, October 1998, http://www.w3c.org/DOM/

[21] R. Bourret, "XML and Databases"; XML-DBMS, February 2002, http://www.rpbourret.com/xmldbms/

[22] Java 2 Enterprise Edition (J2EE) technology, Sun Microsystems, http://java.sun.com/j2ee/

[23] ava-CORBA RMI-IIOP Protocol, Sun Microsystems and IBM Corp., http://java.sun.com/products/rmi-iiop/

[24] Java Language to IDL Mapping, Object Management Group (OMG), July 2000,
http://cgi.omg.org/cgi-bin/doc?ptc/00-01-06

[25] Apache Software Foundation, Jakarta-Tomcat JSP/Servlet Project, 2000
http://jakarta.apache.org/tomcat/index.html

[26] M. B. Jones, C. Berkley, J. Bojilova, and M. Schildhauer, "Managing Scientific Metadata"; IEEE Internet Computing, pp. 59-68, October 2001

[27] D. A. Maluf and P. B. Tran, "Articulation Management for Intelligent Integration of Information";IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews, Vol. 31, No. 4, pp. 485-496, November 2001

[28] R. Goldman, S. Chawathe, A. Crespo, and J. McHugh, "A Standard Textual Interchange Format for the Object Exchange Model (OEM)"; Database Group, Stanford University, 1996,
http://www-db.stanford.edu/~mchughj/oemsyntax/oemsyntax.html

[29] NASA Information Power Grid (IPG);http://www.ipg.nasa.gov/

METACOMPUTING SUPPORT BY P-GRADE[1]-

P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas and N. Podhorszki

MTA SZTAKI
Lab. of Parallel and Distributed Systems
1111 Budapest Kende u. 13, Hungary
email: kacsuk@sztaki.hu

Abstract

The paper describes the latest novel features of the P-GRADE (Parallel Grid Run-time and Application Development Environment) system. P-GRADE enables the usage of the same high-level graphical environment to develop parallel programs for supercomputers, clusters and the Grid, as well as to run the developed parallel program on those systems. Integrating P-GRADE with Condor flocking was the first step to use P-GRADE for the Grid. The parallel checkpoint mechanism introduced in P-GRADE guarantees the same fault-tolerance for PVM jobs as the standard universe guarantees for sequential jobs in Condor. Parallel programs developed by P-GRADE can run both in PVM supported Condor-based Grids like the Hungarian ClusterGrid as well as in MPI supported Globus-based Grids like the Hungarian Supercomputing Grid.

1 Introduction

Grid computing has been originated from metacomputing where the main goal was to execute grand-challenge applications simultaneously at several supercomputers in order to reduce the actual execution time. In the recent years Grid computing was strongly shifted towards a much more general, service-oriented direction where metacomputing is just one possible aspect of the many others aspects of the Grid. Nevertheless, metacomputing is still an important branch of Grid computing and programming environments able to support this branch are extremely important.

Such a programming environment is P-GRADE (Parallel Grid Run-time and Application Development Environment) that has the following man features:

1. It supports the whole life cycle of parallel program development.
2. It provides a complete solution for efficient and easy parallel program development for non-specialist programmers (like chemists, biologists, etc.).

[1] The work described in the paper was partially supported by the following grants: Hungarian Scientific Research Fund (OTKA) no. T032226, Hungarian IKTA-00075/2001, EU GridLab IST-2001-32133 and EU DataGrid IST-2000-25182 projects.

3. It supports fast parallelization of sequential programs.
4. It provides a unified graphical support in program design, editing, debugging and performance analysis.
5. It supports the message-passing parallel programming paradigm by its hybrid (partially graphical, partially textual) language, called GRAPNEL.
6. It generates from a GRAPNEL program either PVM or MPI code according to the user's need.
7. Its run-time system is highly portable: a parallel program developed under P-GRADE can run nearly transparently on supercomputers, clusters and in the Grid.
8. Currently P-GRADE is integrated with Condor and hence it can be used to run PVM jobs in any Grid system, which is based on Condor.
9. The P-GRADE run-time system is extended with automatic parallel program checkpoint and hence either processes of a parallel application or complete applications can migrate in the Grid. (Condor can provide this feature only for sequential jobs).
10. The integration of MPICH-G2 code generator and the Globus enables P-GRADE to run MPI jobs in any Grid system, which is based on Globus.

One of the few real production Grids of the world is the Hungarian ClusterGrid (HCG) that has been constructed since September 2002 and now is operational [1]. The HCG has several unique features:

1. It will connect clusters of all the Hungarian higher educational institutions.
2. These clusters are used as teaching laboratories during the day and they are switched to Grid-mode only for the nights and for the weekends.

During the day these laboratories are used for teaching purposes typically under MS Windows. In the evening these laboratories are switched to Grid-mode, i.e. they are re-booted with Linux and are automatically connected together by the Hungarian Academic network. The basic Grid software is Condor. Currently eight clusters containing 450 PCs are connected from eight universities. The HCG rapidly grows and by the end of 2003 more than 2000 PCs are expected to be integrated in it.

HCG is an inexpensive way to build large production Grids from existing teaching resources by combining them to support high-performance and high-throughput computing. Such way the HCG approach looks attractive for other countries. Already two countries (Ireland and Israel) expressed their interest to adopt this technology for building their cluster-Grids.

Another Grid project is the Hungarian Supercomputing Grid (HSG) that integrates the supercomputers and large clusters of Hungary into a high-performance Grid system. HSG is based on Globus and MPICH-G2.

P-GRADE provides a powerful programming environment both for such cluster and high-performance Grid systems. The paper describes those features of P-GRADE that are used for the Condor-based HCG. Section 2 describes the interactive mode of P-GRADE. Section 3 defines the job mode of P-GRADE showing its integration with Condor. Section 4 explains the parallel check-pointing mechanism of P-GRADE and its connection to P-GRADE. Section 5 shows how P-GRADE is used in the Hungarian Cluster Grid.

2 P-GRADE

In order to cope with the extra complexity of parallel and distributed programs due to inter-process communication and synchronisation, we have designed a graphical programming environment called P-GRADE. Its major goal is to provide an easy-to-use, integrated set of programming tools for development of general message-passing applications to be run on both homogeneous and heterogeneous distributed computing systems like supercomputers, clusters and Grid systems. The central idea of P-GRADE is to support each stage of the parallel program development life-cycle by an integrated graphical environment where all the graphical views applied at the separate stages are related back to the original graph that defines the parallel program and which is designed and edited by the user. The parallel program development life cycle supported by P-GRADE in the interactive execution mode is shown in Fig.1.

The first stage of the life cycle is the program design, which is supported by the GRAPNEL (GRAphical Process NEt Language) language and the GRED graphical editor. In GRAPNEL, all process management and inter-process communication activities are defined graphically in the user's application. Low-level details of the underlying message-passing system are hidden. P-GRADE generates all message-passing library calls automatically on the basis of the graphical code of GRAPNEL. Since graphics hides all the low level details of message-passing, P-GRADE is an ideal programming environment for application programmers who are not experienced in parallel programming (e.g., for chemists, biologists, etc.). GRAPNEL is a hybrid language: while graphics is introduced to define parallel activities, textual language parts (C/C++ or FORTRAN) are used to describe sequential activities.

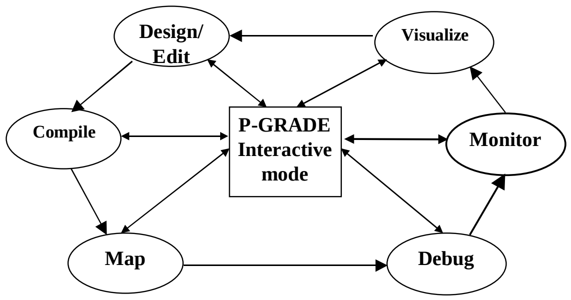


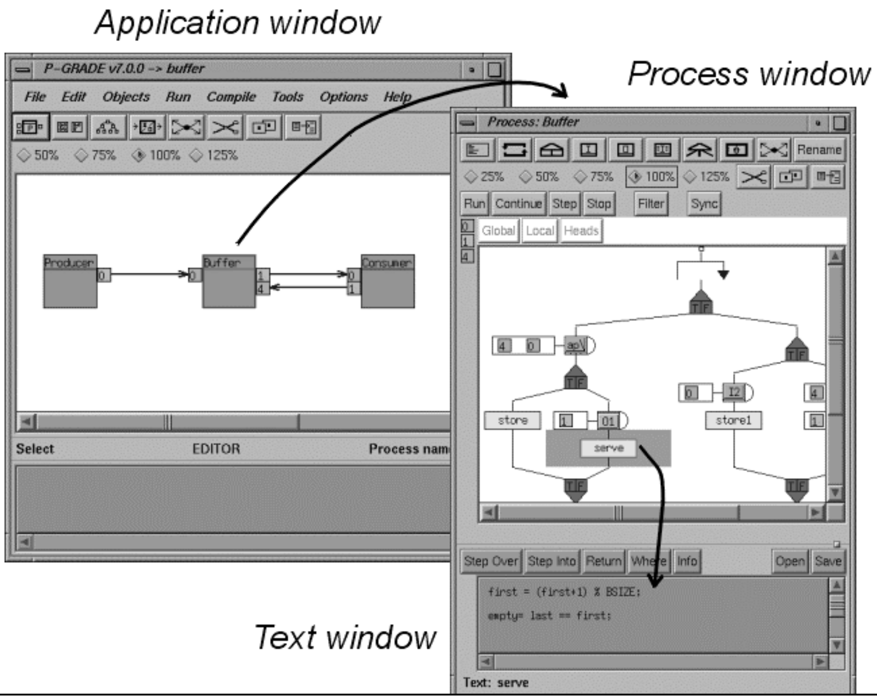
Fig. 1. Program development cycle supported by P-GRADE

GRAPNEL is based on a hierarchical design concept supporting both the bottom-up and top-down design approaches. A GRAPNEL program has three hierarchical layers, which are as follows from top to bottom:

- Application layer is a graphical layer, which is used to define the component processes, their communication ports as well as their connecting communication channels. Shortly, the Application layer serves for describing the interconnection topology of the component processes. An example is shown in Fig. 2.

- Process layer is also a graphical layer to define the internal structure of the component processes by a flow-chart like graph (see Fig. 2). The basic goal is to provide graphical representation for the message passing function calls. As a consequence, every structure that contains message-passing calls should be graphically represented. The following types of graphical blocks are applied: loop construct, conditional construct, sequential block, message passing activity block and graph block. Sequential blocks must not contain any message passing calls.
- Text layer is used to define those parts of the program that are inherently sequential and hence a textual language like C/C++ or FORTRAN can be applied at this level. These textual codes are defined inside the sequential blocks of the Process layer (see Fig. 2).

The top-down design method can be used to describe parallel activities of the application program. At the top level, the topology and protocols of the interprocess communication can be defined and then in the next layer the internal structure of individual processes can be specified. At this level and in the Text layer the bottom-up and top-down design methods can be used in a mixed way. In case of the top-down design method, the user can define the graphical structure of the process and then uses



```
first = (first+1) % BSIZE;
empty= last == first;
```

Fig. 2: Hierarchical layers of GRAPNEL programs and the supporting window types

the Text layer to define the C/C++ or FORTRAN code for the sequential blocks. In the bottom-up design approach, the user can inherit code from existing C/C++ or

FORTRAN libraries and then can build up the internal process structure based on these inherited functions. Moreover, GRAPNEL provides predefined scalable process communication templates that allow the user to generate large process farm, pipeline and mesh applications fast and safely.

The GRED editor helps the user to construct the graphical parts of GRAPNEL programs in an efficient and fast way. GRAPNEL programs edited by GRED are saved into an internal file called GRP file that contains both the graphical and textual information of GRAPNEL programs. The main concepts of GRAPNEL and GRED are described in detail in [2].

The second stage is the pre-compilation and compilation of GRAPNEL programs. The goal of pre-compilation is to translate the graphical language information of the GRP file into PVM or MPI function calls and to generate the C or FORTRAN source code of the GRAPNEL program. For the sake of flexibility, PVM and MPI functions are not called directly in the resulting C code; they are hidden in an internal library, called the GRAPNEL Library. It has two versions. In the first version (GRP-PVM Library) the GRAPNEL Library functions are realised by PVM calls, and in the second version (GRP-MPI Library) they are realised by MPI function calls. For compiling and linking any standard C compiler and linker can be used. The linker uses the following libraries:

- PVM, GRP-PVM (in case of PVM communication system), GRM
- MPI, GRP-MPI (in case of MPI communication system), GRM

The GRM monitoring library is optional; it is needed only if performance monitoring is applied at run time. Details of the code generator are described in [3].

The third stage is mapping in order to allocate processes to processors. A very simple mapping table generated by P-GRADE can do mapping. The mapping table can be easily modified by simple mouse clicks. The mapping information is also inserted into the GRP file.

Having the necessary executables for the parallel/distributed computing system, the next stage is validating and debugging the code. The DIWIDE distributed debugger has been developed for P-GRADE in which the debugging information is related directly back to the user's graphical code. DIWIDE applies a novel macrostep debugging approach [4] where both replay and systematic debugging is possible and it automatically detects deadlocks in the message-passing code. GRAPNEL programs can be executed step-by-step at the usual C instruction level, at the higher-level graphical icon level and at the macrostep level. These features significantly facilitate parallel debugging, which is the most time-consuming stage of parallel program development.

After debugging the code, the next step is performance analysis. First, it requires performance monitoring, which generates an event trace file at program execution time and then, performance visualisation, which displays performance oriented information by several graphical views. The P-GRADE Monitor (GRM) that can support both PVM and MPI [5] performs performance monitoring. PROVE, the performance visualisation tool of P-GRADE, can provide both on-line and off-line visualisation based on the output trace file of GRM [6].

3 Integrating P-GRADE and Condor

The interactive working mode of P-GRADE can be used in supercomputers and clusters when those resources are dedicated for a particular application. However, Grid resources are typically not dedicated for a single Grid application and hence those resources are exploited by jobs that can be located and controlled by Grid job managers. As a consequence since P-GRADE is aimed to support the development and control of parallel Grid applications, too it should support the Grid job execution mode, too. In order to do that we integrated P-GRADE and the flocking mechanism of Condor.

Condor is a resource manager to support high-throughput computing in clusters and in the Grid [7]. The most advanced feature of Condor is the classads mechanism by which it can match application programs with execution resources. When a user submits a job she has to describe the resource requirements and preferences of her job. Similarly, resource providers should advertise their resources by configuration files. The Condor Matchmaker process tries to match jobs and resources by matching the resource requirements and resources configuration files. When such a matching occurs the Matchmaker process notifies both the submitter machine and the selected resource. Then, the submitter machine can send the job to the selected machine that will act as an execution machine.

Integration of P-GRADE and Condor means that after developing a parallel program in the interactive mode the P-GRADE user can switch to batch-mode inside P-GRADE. The program execution will result in the automatic generation of a parallel Condor job. P-GRADE will automatically construct the necessary Condor job description file containing the resource requirements of the parallel job. The mapping function of P-GRADE was changed according to the Condor needs. In Condor the user can define machine classes from which Condor can reserve as many machines as the user in the job description file defines.

When the user generates Condor job under P-GRADE this is the only Condor-related task. P-GRADE supports this activity by offering a default set of machine classes for the user in the mapping phase. After submitting the Condor job under P-GRADE the user can detach P-GRADE from the job. It means that the job does not need the supervision of P-GRADE when it is executed by Condor in the Grid. Meanwhile the P-GRADE generated Condor job is running in the Grid, P-GRADE can be turned off or it can be used for developing other parallel applications. However, at any time the user can attach again the job to P-GRADE in order to watch the current status and results of the job inside the P-GRADE environment. The program development and execution mechanism of P-GRADE for the Grid is shown in Fig. 3.

The integration of P-GRADE and Condor was a joint work of the P-GRADE team at SZTAKI and the Condor team at the University of Wisconsin-Madison. The integrated system was presented at the CCGrid'2002 conference where a finite difference parallel application (FinDiff) shown in Figure 4 was launched from Berlin and it was simultaneously running on three different clusters (at Univ. of Wisconsin-Madison, SZTAKI and Univ. of Westminster) using the flocking techniques of Condor.

The Application Window in Figure 4 shows the inter-process topology of the parallel application. The Manual Mapping window was used to allocate the processes of the application to different clusters. The clusters are represented as Condor classes (see the lower part of the Manual Mapping window) and identified by 0, 1 and 2 shown in the upper part of the window. Finally, the middle part is used to allocate processes of the application to the available resource classes.

P-GRADE main window (P-GRADE 8.2 (2002 March) -> FinDiff) shows that the execution menu contains a choice between interactive and batch mode. If the batch mode was selected the Submit job, Remove job, Detach job, Attach job functions are enabled for the user to control the job execution mode.

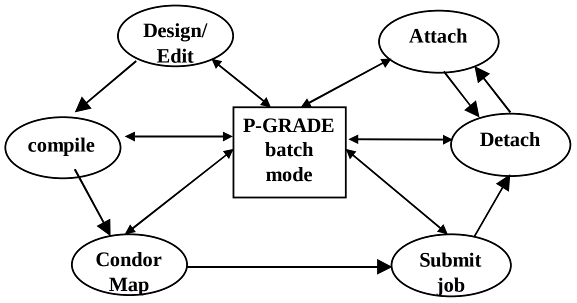


Fig. 3. P-GRADE services in job mode

An important advantage of the integrated Condor/P-GRADE system is that it enables the on-line application monitoring for parallel Condor jobs. Notice that in the original Condor system such application monitoring is not possible. Only the status of the job can be monitored under Condor but the internal behaviour of processes of the parallel job is invisible for the user. In case of the integrated Condor/P-GRADE system the GRM/PROVE tools support the same kind of on-line observation that is possible on supercomputers and clusters in the interactive mode.

If the user requests for execution monitoring and visualization of the parallel program, P-GRADE starts the main monitor of GRM (a Grid application monitoring system) as well as the PROVE visualization window at the submit machine. P-GRADE automatically generates an instrumented code and it links the code of the local monitor of GRM to the application code. By transferring the application, the local monitors are also transferred to the selected sites. After starting the parallel program, the application processes call the first instrumentation function that starts the local monitor and connects to it and then through the local monitor they connect to the main monitor. In this way, the user can observe the detailed behaviour of the parallel code on any selected remote Grid site.

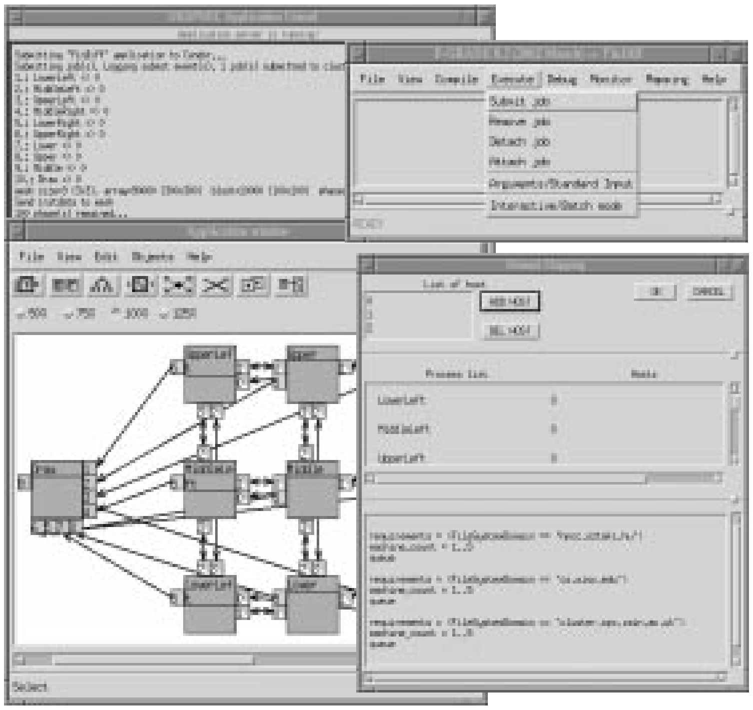


Fig. 4. Snapshot of the P-GRADE/Condor system

4 Check-pointing and process migration in P-GRADE

The P-GRADE run-time system is extended with an automatic check-pointing mechanism. The automatic feature means that check-pointing is completely transparent for the user. It is the task of the P-GRADE run-time system and not the application programmer to take care of check-pointing if it is necessary. Check-pointing can be initiated periodically by the P-GRADE server process for the sake of fault-tolerance or by the load-balancer if process migration is necessary to balance the load among the processors of a parallel system.

Unlike in the Dynamite parallel-check-point system [8], in P-GRADE the whole parallel application is check-pointed which enables to migrate either several processes of an application inside a cluster or to migrate the whole application from one cluster to another. The former option is the typical method to run parallel applications in a supercomputer or cluster while the second one might be necessary in the Grid when an executing Grid site like a supercomputer or cluster becomes overloaded or the pre-allocated time for the Grid application is expired.

The individual processes are check-pointed by the ckpt check-pointer [9] that was developed for check-pointing sequential programs. ckpt is not able to checkpoint parallel programs. Check-pointing parallel applications requires careful design in order to avoid message loss or duplication. An important requirement for the sake of portability was that the underlying communication layer should be untouched, i.e. no cooperation can be expected from the MP layer. However, this requirement raises several difficulties during check-pointing of the messages.

To interrupt the execution of a user process, signals are used. If we analyse the structure of a communication it can be seen that these methods cannot be interrupted at any point. It is because the identifier of a migrating partner process may change by the time the communication is completed. If the communication has already started we have no chance to change this id even if we are aware of the migration of the partner process.

To avoid this problem checkpoint signals must be disabled for the time the communication takes place. However, it raises another problem. If a process is suspended on a receive operation for a longer period of time, it may happen that the process will not be check-pointed.

Based on the GRAPNEL library concept we are able to modify the behaviour of the communication library by redefining the meaning of the MP functions inside the GRAPNEL library. Our solution is redefining the receive operation to act as wildcard receive and we insert the server identifier as a possible source. In this case if a process is stuck into a receive operation and no messages are in the queue, the P-GRADE server process is able to force the waiting process to continue its execution by sending a special checkpoint message. Of course, in this case, the receive operation should be repeated, since the process received an invalid message.

As a summary, there are two possible types of check-pointing interruption. In the first case when the user process is in computing phase (i.e. executing non-communicating code) the ckpt signal works perfectly for interruption. In the second case the user process is executing communication method when the check- pointing is initiated. In such case the server sends a checkpoint message in order to force the process to finish the operation. The latter technique is required only for the receiver processes.

In order to solve the above-mentioned problems we adapted the CoCheck check-pointing mechanism [10] developed for message passing parallel programs. With the modifications of the GRAPNEL library the execution of the whole application can be interrupted along a virtual line that does not cross the border of any communication operation. Before performing the real checkpoint it must be ensured that there are no in-flight messages in the underlying message passing layer. To achieve this state we are vacating the messages from the MP layer buffer by sending special checkpoint messages among the processes and collecting them sequentially. Assuming that the order of the messages sent between two processes does not change, receiving the final checkpoint message indicates the emptiness of the message buffer. During the collection phase user messages also arrive which are stored in the memory and later restored during the execution. Detailed description of the P-GRADE parallel checkpoint mechanism is published in [11].

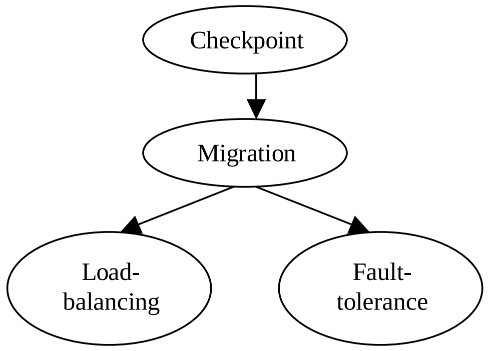


Fig. 5. P-GRADE run-time facilities based on the parallel checkpoint mechanism

Based on the new parallel checkpoint mechanism P-GRADE offers new possibilities at run-time. Processes can be migrated among processors of a supercomputer or cluster at any time if a load-balancer decides that the work of the processors is unbalanced. Creating periodically the checkpoint of an application makes it possible to create a fault-tolerant parallel system. It is particularly important in case of large parallel programs running days or weeks on a cluster where machines can become faulty or disconnected during the execution time. In such case the parallel application can continue from the last checkpoint reallocating the processes for the still available machines. Without check pointing the whole application should be restarted from the very beginning. The new facilities of the interactive P-GRADE mode based on the parallel checkpoint mechanism are summarised in Figure 5.

The techniques used in the interactive mode could be extended towards the integrated Condor/P-GRADE system, i.e. for the job mode of P-GRADE. In order to migrate the job, Condor provides transparent, automatic check-pointing for sequential programs. Based on this sequential check-pointing mechanism Condor guaranties that sooner or later a sequential job will be completed in the Grid. Unfortunately, Condor does not provide the same service for parallel programs.

By integrating P-GRADE and Condor our goal was to provide a Grid-enabled run-time system for P-GRADE and on the other hand to extend Condor with parallel check pointing inherited from P-GRADE. Notice that there is a strong equivalence of the Condor sequential job execution mechanism in the Grid and the P-GRADE parallel program execution mechanism in a cluster. The Condor run-time facilities for sequential programs are shown in Figure 6. If we compare Figure 5 and 6 it is clear that integrating Condor and P-GRADE leads to a Grid system where the same guaranties can be given for a parallel program as currently offered for sequential programs by Condor.

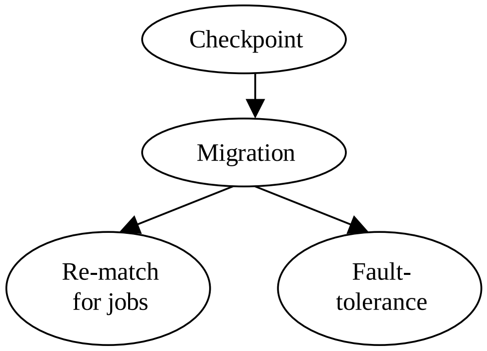


Fig. 6. Condor run-time facilities based on the sequential checkpoint mechanism

5 Using P-GRADE in the Hungarian ClusterGrid (HGG)

The HCG is a two level Condor system. Each cluster of the HCG is a Condor pool that is connected together as "friendly" Condor pools and hence the flocking technique is supported among them. Friendly Condor pools require the destruction of firewalls among them. In order to keep the HCG as secure as possible a Virtual Private Network (VPN) was created among the "friendly" Condor pools on top of the Hungarian Academic Network. It means that when a laboratory is switched from the day-shift laboratory mode into the night shift Grid mode not only the operating system is rebooted but also the routers are reconfigured to establish the VPN connection. As a result the whole HCG is protected by a firewall but internally among the clusters there are no firewalls. Once someone was allowed to pass the external firewall can access any resources internally based on Condor. In order to make the HCG as secure as possible there is only one entry machine that realizes security checks and it also plays the role of the main Condor machine representing the second level on the top of the first-level clusters.

The integrated Condor/P-GRADE environment (as described in Section 3 and 4) can ideally support such a Grid system. Parallel jobs can be easily developed, submitted, observed and supervised by end-users. In addition the parallel checkpoint system of P-GRADE can support even migration of parallel jobs under Condor. This feature is extremely important in the HCG where the day shift and night shift are used after each other. After the night shift (Grid mode) all the clusters are switched into laboratory mode in the morning. If a job is not finished by that time, it will be completely lost without checkpoint support and should be restarted in the next night shift. Condor supports check-pointing for sequential jobs but does not support parallel jobs. Using the parallel check-pointing mechanism developed for P-GRADE and described in Section 4, even parallel jobs can be saved in the morning and resumed in the evening. More than that Condor can allocate other resources for the parallel job in the next night shift to achieve better exploitation of resources of the HCG.

Under these conditions of application migration, the Grid Application Monitoring Infrastructure (GAMI) developed by SZTAKI in the EU GridLab project should support application monitoring. In this case GRM is used only for code instrumentation and the actual monitoring is realized by the GAMI. Details of the GAMI are described in [12]. However, all these changes are transparent for the user who still applies PROVE for performance and execution visualization.

The Condor job mode can be used only if the submit machine is part of a Condor pool. However, such a restriction is too strong in the Grid where there are many Condor pools as potential resources and these should be accessed from the submit machine. This is the case, for example, in the Hungarian ClusterGrid where the Grid itself is a collection of Condor pools (clusters) but the user's submit machine is typically not part of the ClusterGrid.

If the entry machine of the HCG is always used as the Condor submit machine, it could be a significant bottleneck from the point of view of Condor file management. It is because Condor executes every file operation on the submit machine. In order to avoid this problem some new functionalities should be provided for the user. In order to provide these missing functionalities we have created a thin layer, called PERL-GRID, between P-GRADE and Condor with the following tasks:

- The selection of the high-performance computing site (Condor pool) is the task of a Grid resource broker. Currently a random selection algorithm is applied by PERL-GRID. (A Grid resource broker will replace this in the future.)
- PERL-GRID takes care of contacting the selected site and executing the mutual authentication based on the SSH technology.
- PERL-GRID also takes care of file staging by transferring the input files and the code file into a temporary directory at the reserved Grid site. Finally, it passes the job to the local job manager. Currently it is Condor but soon others, like SGE will be supported.

The PERL-GRID job mode of P-GRADE was demonstrated by the Grid execution of the MEANDER nowcast program package of the Hungarian Meteorology Service at the 5th EU DataGrid conference in Piliscsaba (Hungary). The goal of the MEANDER package is to provide ultra-short range (up to 1 hour) weather forecast of dangerous situations like storm and fog on a high resolution regular mesh (10km -> 1km). To achieve this goal we have jointly parallelised six out of the ten component algorithms of MEANDER by P-GRADE [13].

At Piliscsaba, we used a simplified version of MEANDER containing only 4 algorithms for the demonstration. The P-GRADE view of this MEANDER version is shown in Fig. 7. It is clear on Fig. 7 that the four algorithms are realized by the processor farm concept. The numbers in the clouds represent the number of processors that were used in the demonstration. It can also be seen that these algorithms are

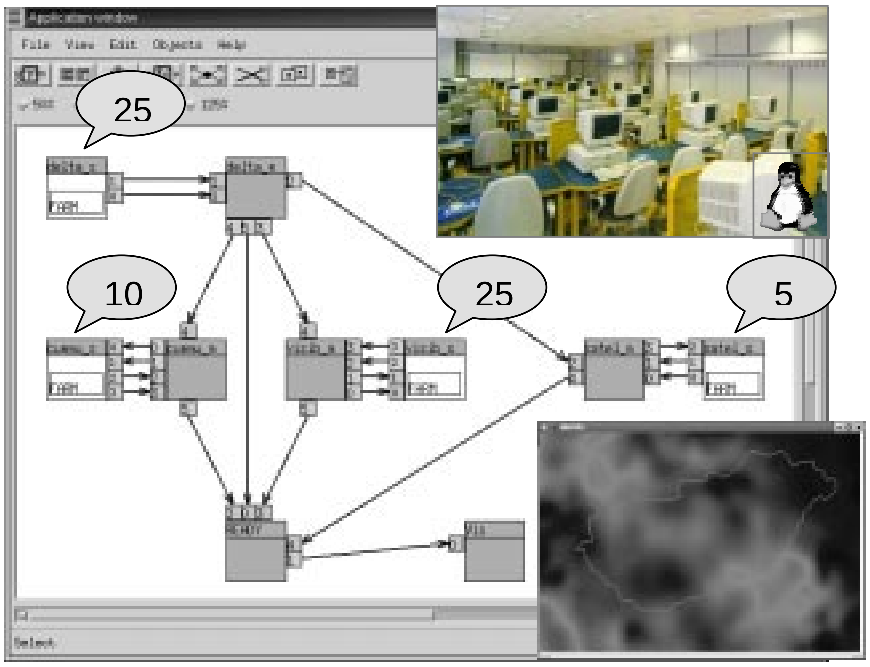


Fig. 7. The demonstration P-GRADE version of the MEANDER program

connected like a workflow. First the delta algorithm should be executed, and then in the second phase, the other three algorithms in parallel. At that time 40 processors were used in parallel. Finally, a visualization process is applied to draw the weather map shown in the right bottom of Fig. 7.

The parallel job was generated by P-GRADE and passed to PERL-GRID at Piliscsaba. PERL-GRID selected the 58 processor Linux cluster of SZTAKI (shown in the right top of the picture) to execute the job. Then PERL-GRID transferred the job to the SZTAKI cluster, collected the necessary meteorology database input file from the Hungarian Meteorology Service and passed the job with all the files to Condor at the SZTAKI cluster. Condor took care of the parallel execution of the job at SZTAKI [14]. When monitoring was requested, PERL-GRID delivered the local monitor code, too and the collected trace file was sent back and visualized by PROVE on the submit machine at Piliscsaba whenever the user requested a trace collection by PROVE. The final trace file visualization picture is shown in Fig. 8. The picture clearly shows that first the delta algorithm was running on 25 processors and when it was finished, it triggered the execution of the other three algorithms that were executed simultaneously. Finally, when the whole job was finished, PERL-GRID took care of transferring the

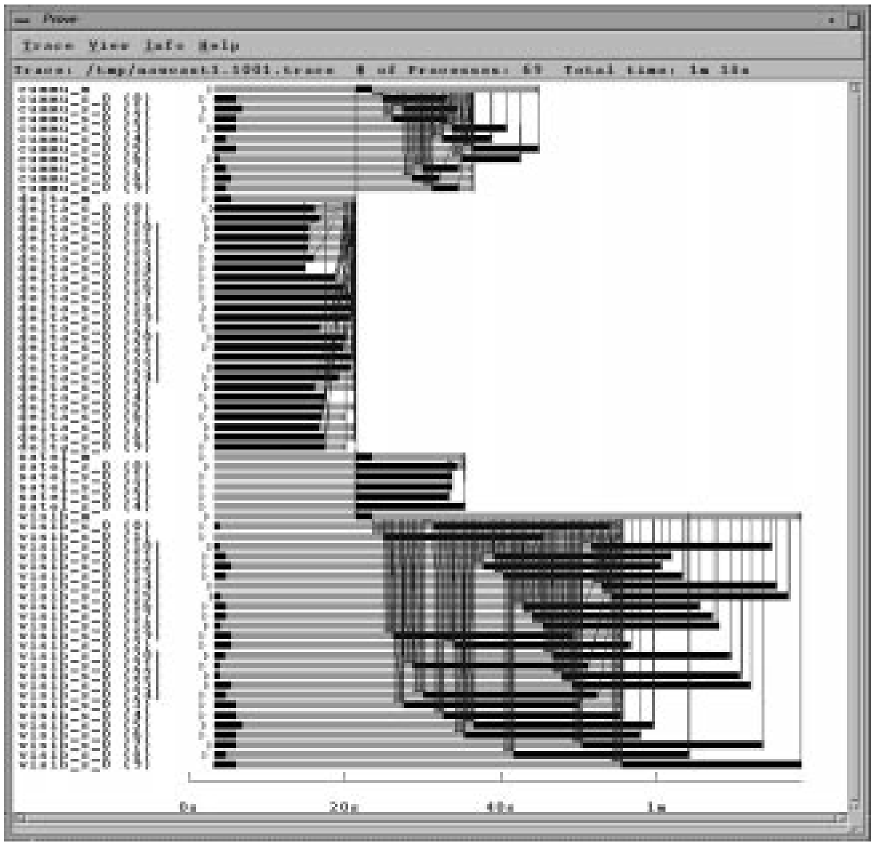


Fig. 8. Process space-time diagram of the MEANDER program

result file back to Piliscsaba and removing the temporary directory it created for the job at the SZTAKI cluster.

6 Related work

There is a tremendous Grid community effort in order to define the core Grid infrastructure. However, so far not too much attention was paid how to develop Grid applications and how to supervise the running application from the point of view of job status monitoring, trouble-shooting and performance monitoring. The basic user interface for Grid activities is the Grid portal, which assists the user to get knowledge on available Grid resources and their basic (static and/or dynamic) properties [15], [16]. It also helps the user to launch jobs in the Grid and query their status. Nevertheless, the typical Grid portals give little support to the user to construct Grid applications out of existing or newly written Grid services.

There are only very few projects that provide an overall Grid program development environment. One of those projects is Cactus that supports the creation of mesh topology based parallel applications [17]. Such applications are very frequently used for example in physics. An important Grid project that works on providing a generic Grid Application Toolkit (GAT) by which Grid enabled applications can be constructed is the EU GridLab project [17]. GAT will play a similar role in case of Grid as PVM [18] and MPI [19] played in case of parallel systems. However, like PVM and MPI requires the application developer to learn a lot of APIs for writing parallel programs, GAT will require the user to learn the GAT APIs for creating Grid applications.

Considering Grid application monitoring the EU CrossGrid project plans to develop an interactive and on-line Grid application monitoring system [20]. It will be based on OMIS [21] and the structure of the monitoring infrastructure is very similar to the GAMI developed in the EU GridLab project and used by P-GRADE.

7 Conclusions

P-GRADE provides a high-level graphical environment to develop parallel applications transparently both for parallel systems and the Grid. One of the main advantages of P-GRADE is that the user has not to learn the different APIs for parallel systems and the Grid, simply by using the same environment will result in a parallel application transparently applicable either for supercomputers, clusters or the Grid. The current version of P-GRADE supports the interactive execution of parallel programs as well as the creation of a Condor or PERL-GRID job to execute the parallel program in the Grid. The integrated P-GRADE/Condor Grid system guarantees reliable, fault-tolerant parallel program execution in the Grid, like the Condor system guarantees such features for sequential programs under the standard universe.

The GRM/PROVE performance monitoring and visualisation toolset has been extended towards the Grid and connected to a general Grid application-monitoring infrastructure (GAMI) developed in the EU GridLab project. Using the GAMI/GRM/PROVE system any parallel application launched by the integrated Condor/P-GRADE system can be remotely monitored and analysed at run time. On-line performance and execution visualization support is provided by PROVE.

P-GRADE is currently ported to the Hungarian ClusterGrid that connects the Condor pools of the Hungarian higher educational institutions into a high-performance, high-throughput Grid system. As such the Grid execution mode of P-GRADE is currently strongly connected to Condor. However, the use of PERL-GRID enables the easy connection of P-GRADE to other local job managers like SGE and PBS. As a result P-GRADE will provide a transparent parallel programming environment for cluster-Grids like the HCG no matter what kind of local job managers are applied in them. Currently we work on integrating P-GRADE with Globus in the framework of the Hungarian Supercomputing Grid (HSG) project. As a result MPICH-G2 program generated by P-GRADE will be able to run in the integrated P-GRADE/Globus system of HSG. When the GAT API developed in the GridLab project is available we will connect P-GRADE to the Grid through the GAT.

P-GRADE 8.2.2, the interactive version of P-GRADE can be freely downloaded (with a User's Manual and a set of demo programs) from the www.lpds.sztaki.hu web site. The new release (P-GRADE 8.3) with the single job Grid execution support (both with Condor and PERL-GRID job mode) is planned by the end of August. The prototype of the integrated P-GRADE/Globus system is also expected by the end of August.

8 References

[1] P. Stefán: The Hungarian ClusterGrid Project, Proc. of MIPRO'2003, Opatija, 2003
[2] P. Kacsuk: Visual Parallel Programming on SGI Machines, Invited paper, Proc. of the SGI Users' Conference, Krakow, Poland, pp. 37-56, 2000
[3] D. Drótos, G. Dózsa, and P. Kacsuk: GRAPNEL to C Translation in the GRADE Environment, In: Parallel Program Development for Cluster Computing, Methodology, Tools and Integrated Environments (eds: Cunha, C., Kacsuk, P. and Winter S.C.), Nova Science Publishers, Inc. pp. 249-263, 2001
[4] P.Kacsuk: Systematic Macrostep Debugging of Message Passing Parallel Programs, Journal of Future Generation Computer Systems, Vol. 16, No. 6, pp. 609-624, 2000
[5] Z. Balaton, P. Kacsuk, and N. Podhorszki: Application Monitoring in the Grid with GRM and PROVE, Proc. of the Int. Conf. on Computational Science – ICCS 2001, San Francisco, pp. 253-262, 2001
[6] P. Kacsuk: Performance Visualisation in the GRADE Parallel Programming Environment, Proc. of the Fourth Int. Conf. on High Performance Computing in Asia-Pacific Region (HPC'Asia 2000), Peking. pp. 446-450, 2000
[7] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003
[8] D. van Albada and P. Sloot: Surfing the Grid - Dynamic Task Migration in the Polder Metacomputer Project, Invited talk, EuroPVM/MPI'2002, Linz, 2002
[9] www.cs.wisc.edu/~zandy/ckpt
[10] http://wwwbode.cs.tum.edu/Par/tools/Projects/CoCheck.html
[11] J. Kovács and P. Kacsuk: Server Based Migration of Parallel Applications, Proc. of DAPSYS'2002, Linz, pp. 30-37, 2002
[12] Z. Balaton and G. Gombás: Resource and Job Monitoring in the Grid, submitted and accepted for EuroPar'2003, Klagenfurt, 2003
[13] R. Lovas, et al: Application of P-GRADE Development Environment in Meteorology, Proc. of DAPSYS'2002, Linz, pp. 30-37, 2002
[14] http://www.cs.wisc.edu/condor/manual[15] Grid Portal Development Kit (GPDK), http://www.doesciencegrid.org/Grid[16] Genius, http://www.infn.it/grid[17] G. Allen, et al.: GridLab - A Grid Application Toolkit and Tested, submitted and accepted for Special Issue on Grid Computing "Future Generation Computing Systems"
[18] Geist, A., et al.: PVM 3 User's guide and Reference Manual, ORNL/TM-12187, 1994
[19] Message Passing Interface Forum: MPI: A Message Passing Interface Standard, 1994
[20] M. Bubak, et al.: The G-PM Tool for Grid-oriented Performance Analysis, Proc. of the 1st European Across Grids Conference, Santiago de Compostela, 2003
[21] T. Ludwig and R. Wismüller. OMIS 2.0 -- A Universal Interface for Monitoring Systems. In M. Bubak, J. Dongarra, and J. Wasniewski, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 4th European PVM/MPI Users' Group Meeting, volume 1332 of Lecture Notes in Computer Science, Springer Verlag, pages 267-276, Crakow, 1997

Design and Implementation of the Web-based Grid-computing Framework *GridGate*

Kang kyung-woo, Kang yun-hee, Kim do-hyun, Cho kwang-moon, Kung sang-whan
Department of Information and Communication Engineering., Cheonan University,
115, Anseo-dong, Cheonan-city, Choongnam, Korea, 330-704
E-mail : {kwkang,yhkang,dhkim,ckmoon,kung}@cheonan.ac.kr

Abstract

Grid-computing on networked computers is increasingly applied to solve a variety of large-scale computation problems. Several software systems are developed to provide the application programmers with computers which are available in wide environment. However, these systems do not supplies web-based interface or the real-time visualization facility. Web technology is becoming the general technology on the development of network applications, in particular, because its interface can be made platform independent. In this paper, we propose a web-based framework for executing the parallel SPMD application written in MPI. Also, a web-based collaborative environment is developed with a real-time visualization technology.

***Key Words**: grid-computing, CFD, visualization*

I. Introduction

With the advance of network and software infrastructure, grid-computing technology on a cluster of heterogeneous computing resources becomes pervasive[1,3,4]. Grid-computing describes a coordinated use of an assembly of distributed computers, which are linked by networks and are deployed by many kinds of softwares. The potential benefit of the grid-computing is that users can exploit a powerful monolithic virtual machine. However, construction of a grid-computing system has been a challenging task which involves broad spectra of technical issues. The major hurdles of grid-computing environments are due to the lack of tools to facilitate the development of parallel and distributed applications[4]. Consider the basic operations that arise in the development cycle of such applications[8]:

1. Generation: source files
2. Transfer: source files and data files between computing resources.
3. Compile: each source file on each computing resource
4. Execution: execution file
5. Visualization: the result to be collected from computing resources after the finish of the execution

The steps including two, three and five can become quite time consuming as the number of resources increases because users must transfer many files between computers. Moreover, step five should wait for the completion of the previous step, the fourth step, which might require long time. In order to use the visualization service, the user needs to have an account on the machine on which the visualization system is installed.

In this research, a web-based framework(hereinafter referred to as *GridGate*) is developed for doing the five steps on one interface. For this purpose, this research will focus on more effective use of computing resources by making the systems more easily available and collaborative to the researchers.

II. Target Application

In this work, the target application area is confined to SPMD applications, a representative single-most-important displine in parallel computing applications. In this paper, CFD(Computational Fluid Dynamics) is introduced as an example of SPMD applications[8]. CFD describes a methodology to analyze fluid flows and associated heat transfer in a numerical way[8]. The main work is to acquire numerical solutions to the governing Navier-Stokes (possibly Euler or simplified) equations. Since these governing equations are a system of nonlinear partial differential equations, traditional analytical approach is not possible except for some specially-simple problem. From the earlier stage of computing, CFD plays a major role to drive parallel computing.

CFD transforms the original governing equation to a matrix equation by using an appropriate discretization method. The transformed matrix equation is then solved numerically, which usually requires massive arithmetic operations, i.e., computational power. Owing to nonlinearity and the huge size of the matrix equation, most of the solution algorithms are based on iterative method. In addition, analysis of the unsteady phenomena requires the time-marched solution. Due to these specific features, the computational procedure can be in general represented by a repeated computation. The repetition includes both iteration and time marching. Finite difference method (FDM), a representative method of CFD is conceptually depicted in figure 2.1. As shown in figure 2.1, the physical flow fields are discretized spatially into (I_{max}, J_{max}) grid points and discretized into Nstep_Max steps in the time axis. Imax and Jmax represent the maximum grid points in each spatial direction, and Nstepmax is the maximum time limits to be calculated.

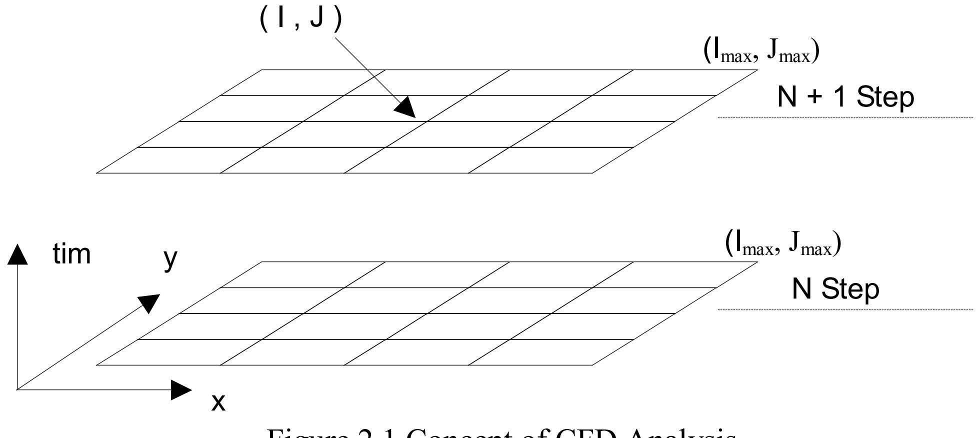


Figure 2.1 Concept of CFD Analysis

FDM adopts grid network in the solution domain. The solution is acquired at the given points rather than continuum. The discretization of the governing equation on each grid yields an algebraic equation at each grid point (I, J), which constitutes a matrix equation. Consequently, the CFD simulation is a process to get the simultaneous equation at every grid point (I, J) for each time step N. Each time step generates the intermediate files that could be merged into one file. Visualization system gets a series of merged files as its input and generates the animation that represents its simulation.

III. System architecture of *GridGate*

The simulation procedure in *GridGate* is schematically described in figure 3.1. The structure of *GridGate* consists of two major functional parts; grid portal and realtime visualization. The basic components for a computational run in *GridGate* are the user's source code for simulation (referred to as solver) and data files. It is assumed that the user prepares these files. The realtime visualization provides monitoring of the intermediate result of the assigned application by real-time graphical processing to a three-dimensional graph.

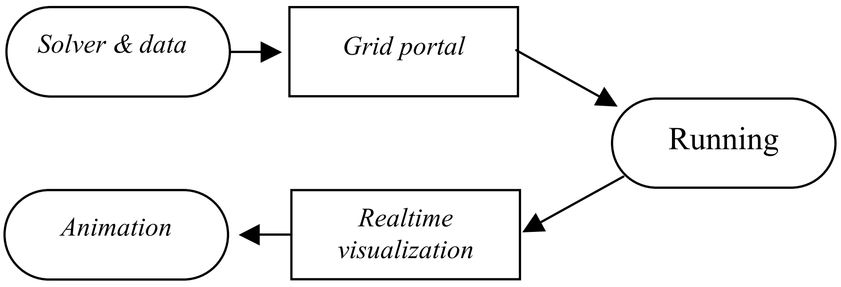


Figure 3.1. System Flow of *GridGate*

3.1 Grid portal

A grid portal is developed for easy and rapid development of SPMD applications on the Grid environment. Our grid portal supplies the web-based functions of authentication, visualization of resource's state, distribution&compilation of user's job and spawning the job. These functions are shown in the following figure 3.2.

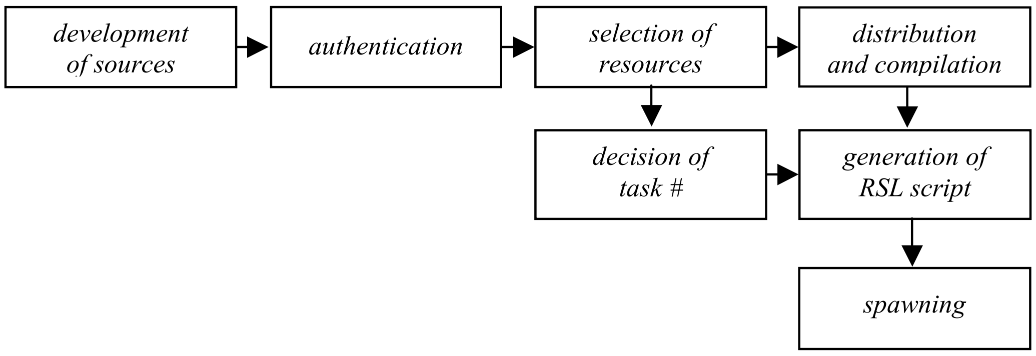


Figure 3.2. Structure of grid portal

The basic components for a computational run in *GridGate* are the user's source code and data file for simulation (referred to as solver). It is assumed that user prepares these files and the makefile to compile the source in each machine. The authentication step needs user's id and password one time. In this research, authentication step generates an user proxy using SWING[9]. In the next step, user could choose resources and the number of tasks according to the current information of the computing resources. The hardware information includes the name, architecture, CPU load and network traffic.

3.2 Realtime visualization

Visualization should show the time dependence of a data set, the result of simulations, using several techniques. Simulations of fluid dynamics are examples of the applications that generate data sets in time. Static visualizations of dynamic information are sometimes ineffective and incorrect because they do not convey motion. Animations can help the user understand, particularly if the user is interested in the process of simulation.

A traditional approach to the visualization is for the user to use the visualization system or a library to implement application specific visualizations. Many such visualization systems require that the user has an account on the machine on which the software is installed. For these reasons scientists often transfer the results of these visualizations by video-tape or by converting a series of animation frames to a movie file(MPEG, GIF etc). The process of translating a visualization to the movie file eliminates user interaction with the visualization.

Recent technological developments have created opportunities for new approaches for representing time-sensitive problems using web-based computer animation. More importantly, Java and the World Wide Web have provided a universal platform for building animations and visualizations.

This web-based execution model solves many dissemination problems. Users can interact with animations without having an account on a particular machine. Figure 3.3 shows the process of realtime visualization based on the web. Each task generates a series of intermediate files that could be merged into one file. Image generator gets the merged files as its input and generates the animation that represents its simulation. In this research, we use *gnuplot* as the image generator.

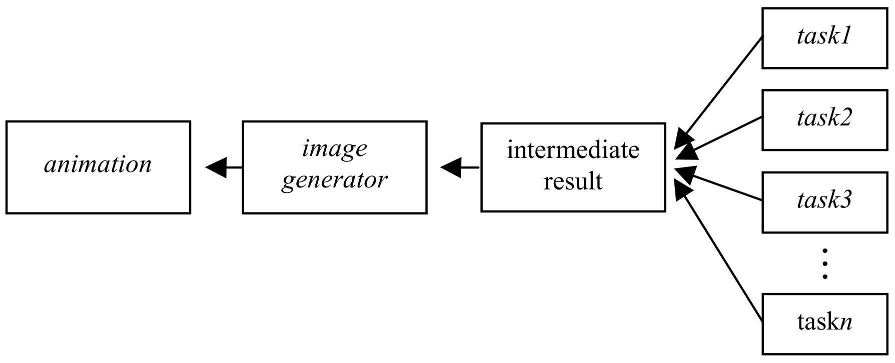
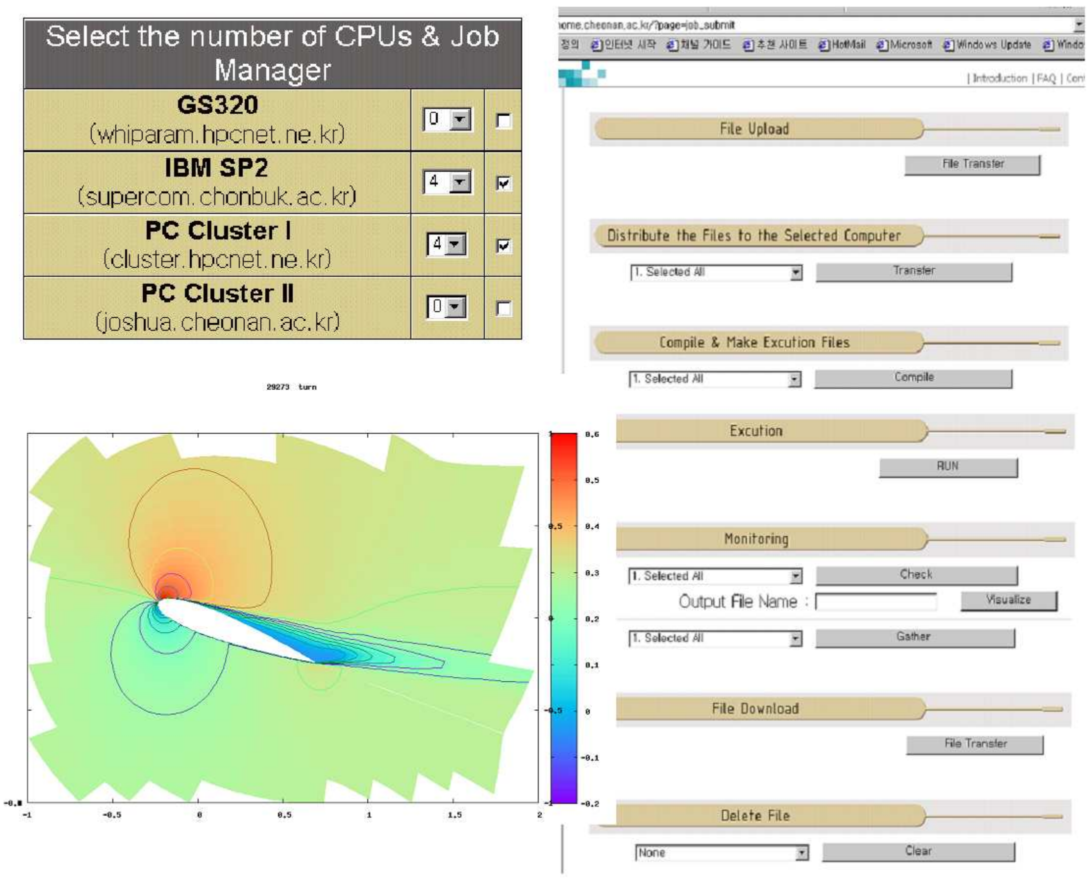


Figure 3.3. The process of realtime visualization

3.3 User Interface

A dedicated graphic user interface is designed for *GridGate*. This GUI is responsible for controlling the system and manipulation of a RSL(Resource Specification Language) file. The RSL file is the important input for Globus as a text file which contains the information both of the resources and the solver required for running a grid-computing application. The resource information includes the host name, local scheduler, and working directory. The solver information describes data I/O as well as its name.



Select the number of CPUs & Job Manager		
GS320 (whiparam.hpcnet.ne.kr)	0	☐
IBM SP2 (supercom.chonbuk.ac.kr)	4	✓
PC Cluster I (cluster.hpcnet.ne.kr)	4	✓
PC Cluster II (joshua.cheonan.ac.kr)	0	☐

Figure 3.4. User interfaces in *GridGate*

GridGate provides three major interfaces: list of available resources, panel for job-submit and visualization. The control panel supplies many buttons. A button "Run" as shown in figure 3.4 invokes the user's job according to the information prescribed in the RSL file that is automatically created by selecting the computing resources. A button "Visualize" opens the visualization interface, on which the status of execution of the solver is monitored by a real-time animation. The other buttons, "Transfer" and "Compile" transfers the user's job to each resource and compiles the solver, respectively.

IV. Experiment

4.1. Testbed for grid-computing

In this research, we established a testbed that consists of five parallel systems; Compaq HPC320, Compaq GS320, Linux-Cluster, two IBM SP2 machines. These systems achieve the different utilization from each other depending on the time. Eventually, our *GridGate* allows people to reach out and get the computational resources they need from their own desktop workstations and to monitor their intermediate results from anywhere. In this research, we developed *GridGate* System and used Globus Toolkits[3-6] and several job schedulers depending on the supercomputing resources as shown in figure 4.1. The *GridGate* System is the grid-computing tool that supplies GUI, job-distribution and remote compilation.

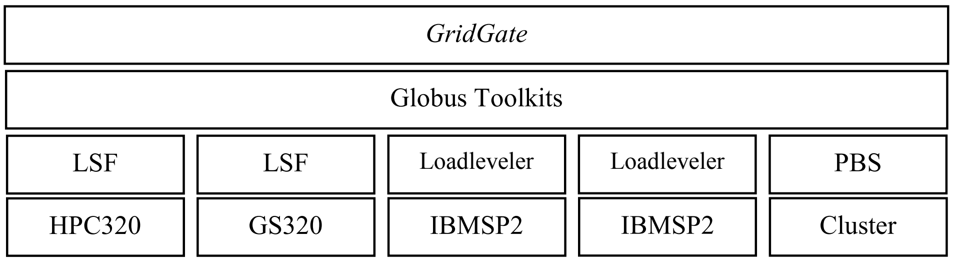


Figure 4.1 Hierarchy of middlewares in our testbed

Figure 4.2 shows the network configuration of our testbed. The supercomputers of KISTI are connected with 800 MBps HIPPI(High-Performance Parallel Interface) and are linked to the Korean R&D network; Kreonet and HPCnet. The HIPPI makes our grid faster because the latency time of the network is short. Two IBM SP2s are linked to 45 Mbps Kreonet/HPCnet

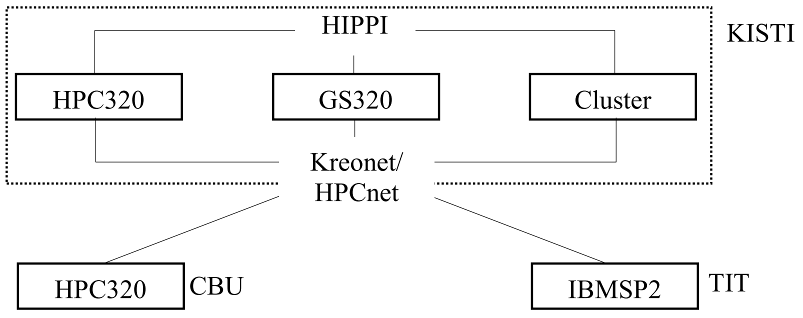


Figure 4.2 Network configuration in our testbed

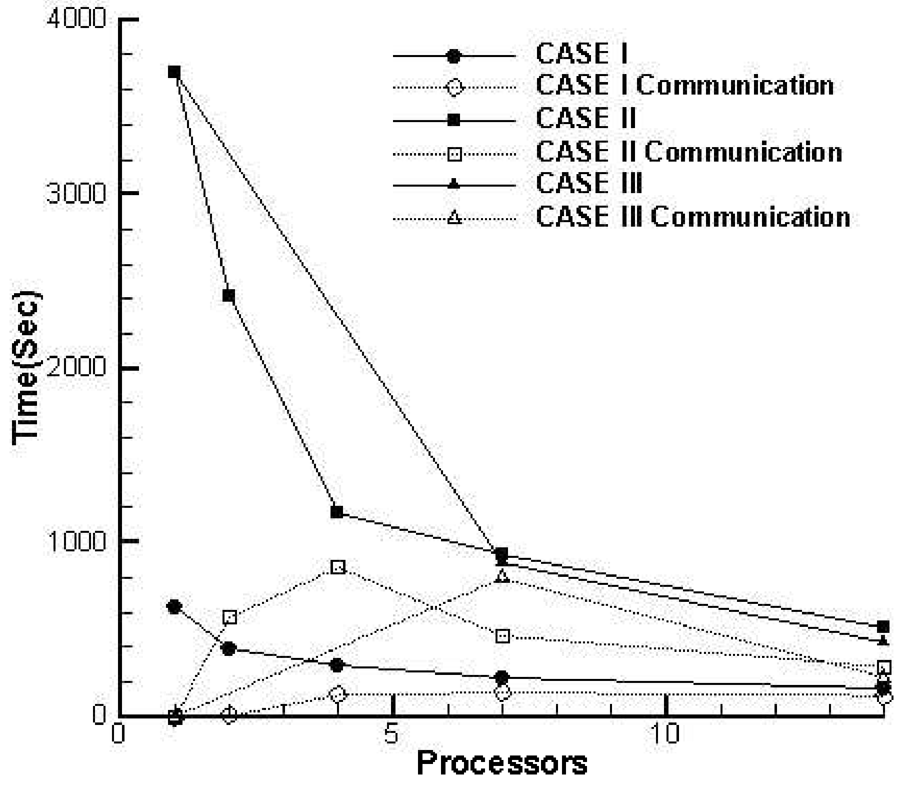
4.2 Experimental Results

Table 4.1 shows the number of processors on the experiments using a turbo machine fluid analysis program.

CPU #	CASE I		CASE II		CASE III			
	HPC320	GS320	IBMSP2	GS320	Cluster	IBMSP2	HPC320	GS320
1	1		1			1		
2	1	1	1	1				
3	2	2	2	2	1	1	1	1
7	3	3	2	5	1	1	2	3
14	4	10	2	12	1	2	4	7

Table 4.1 The number of processors on the experiment

These experiments are conducted according to the number of processors when the size of computation is 28 blocks. The elapsed time is measured when the iteration reaches 100. We are not fully able to use the processors of the supercomputers because of the restriction of the management policy. Although we could obtain the speed-up according to the increase of processors, the communication time commanded the absolute majority.



(1) CASE I (computation using HPC320, GS320)

This experiment is conducted using two supercomputers connected with LAN.

The result shows the speed-up in proportion to the number of processors in spite of the communication overhead.

(2) CASE II (computation using IBM SP2, GS320)
CASE II simulates the same code on the WAN environment between CNU and KISTI. In spite of WAN environment, the increase of processors makes the elapsed time reduced. In fact, we distribute the job on two supercomputers unequally in order to reduce communication data.

(3) CASE III (computation using cluster, GS320, IBM SP2, HPC320)
This case is similar to CASE II on the WAN environment and the unequal distribution of job.

V. Conclusion

In this research, we implemented a grid toolkit named *GridGate* on several supercomputers connected with LAN or WAN. The users from any systems could submit jobs and have them transparently run on the grid environment. This would provide many benefits to the supercomputing centers, including utilization and to the users a convenient simulation environment. A major challenge for this research was providing a uniform software environment across the geographically distributed and diverse computational resources. To meet this challenge, we developed *GridGate* and used Globus Toolkit, which provides a variety of services including co-allocation, security, and parallel programming support.

References

[1] I. Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit Intl. J. Supercomputer Applications, 11(2):115-128, 1997
[2] "Creating New Information Providers," MDS 2.1 GRIS Specification Document, USC/ISI, May, 2002
[3] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit" Intl. J. Supercomputer Applications, 11(2):115-128, 1997
[4] I. Foster and C. Kesselman (eds.) "The Grid: Blueprint for a new Computing Infrastructure" Morgan Kaufmann Publishers, 1998
[5] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing." Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001
[6] V. Sunderam. "PVM: A Framework for Parallel Distributed Computing," Concurrency: Practice and Experience, Vol 2 No 4, December 1990
[7] Bhatia, D., Burzevski, etc. "WebFlow-a visual programming paradigm for Web/Java based corse grain distributed computing," Concurrency: Practice and Experience, March 1997. Java Special Issue.
[8] Klaus A. Hoffmann, "Computational Fluid Dynamics for Engineers," 1993.

Grid Enabling Applications Using Triana

Ian Taylor[1], Matthew Shields[2], Ian Wang[2] and Roger Philp[3]
Contact: i.j.taylor@cs.cardiff.ac.uk

Abstract

In this paper, we describe the Triana problem-solving environment and outline how a scientist might use it to modularize their applications into a set of cooperating components. We show that using this approach, applications can be distributed onto the Grid in a flexible and intuitive way using any of the distribution policies and mechanisms available within Triana. This is illustrated through the use of example by describing the implementation of a monolithic galaxy formation visualization code into a set of Triana units that can be deployed onto the Grid. We outline the middleware independent nature of Triana through the use of the GAT application-driven API and describe how a user would take advantage of this functionality through the use of the Triana GUI.

1. Introduction

Recently, there has been significant interest in the field of Grid computing and the convergence of Grid Computing and Web Services in the form of the Open Grid Services Architecture (OGSA) [1]. This has given rise to an enormous drive in this direction by both industrial [2] and academic projects, such as Globus [3]. In parallel, peer to peer (P2P) technology has gained much interest through the popularity of services like Gnutella [4] and SETI@home [5]. One recent advance has been the emergence of architectures that support the programming of such networks e.g. project JXTA [6], which defines a set of protocols that can be used to create decentralized P2P networks. Grid Computing and P2P computing are both important emerging paradigms for seamless aggregation and utilization of the ever-increasing computing resources available today throughout the world. P2P is more focused on computing at the edges of the Internet i.e. transient devices that live behind NAT, firewalls etc whereas Gird computing is far more focused on connecting virtual organizations [7] that can cooperate in a collaborative fashion.

Application developers however are often left confused about exactly which middleware/infrastructure to use when grid enabling their application. They are faced with a collection of differing APIs that claim to do similar things, but which have a rather steep learning and implementation curve for deployment of their applications onto the Grid. To this end, an important advancement has been achieved by defining the Grid Application Toolkit (GAT) API [8]. The GAT provides an application-driven API and implements key bindings to the various underlying mechanisms for the implementation of this functionality. Further, the GAT can be dynamically switched at run time to utilize the functionality that exists on a particular platform or environment. Current GAT implementations include Web Services (OGSA to follow shortly), JXTA and local services for prototyping applications.

[1] Department Of Computer Science, Cardiff University
[2] Department of Computer Science and Physics and Astronomy, Cardiff University
[3] Cardiff Centre for Computational Science and Engineering, Cardiff University

In this paper, we describe Triana, an application that takes advantage of the GAT interface. We then give an overview of how Triana has recently been used to grid enable an existing galaxy formation visualization application and how this can be distributed across the Grid using the Triana distribution mechanism. The next section gives a brief overview of Triana, followed by a description of how one would use Triana to implement a distributed application. We then discuss the mechanism of how Triana distributes its units, the role of the GAT and how a user would specify this using the Triana GUI.

2. Triana

Triana [9] is a visual programming environment that allows users to compose applications from programming units (or components) by dragging and dropping them into a workspace, and connecting them together to build a workflow graph. Triana was initially developed by scientists in GEO600[4] to help in the flexible quick-look analysis of data sets. It therefore contains much of the core tools that are needed to support the analysis of one-dimensional data sets. For example, we have: flexible data importers/exporters; numerous signal processing algorithms; mathematical functions; and visualization tools based on the integration of commonly used packages, such as SGT [10] and XMGrace [11].

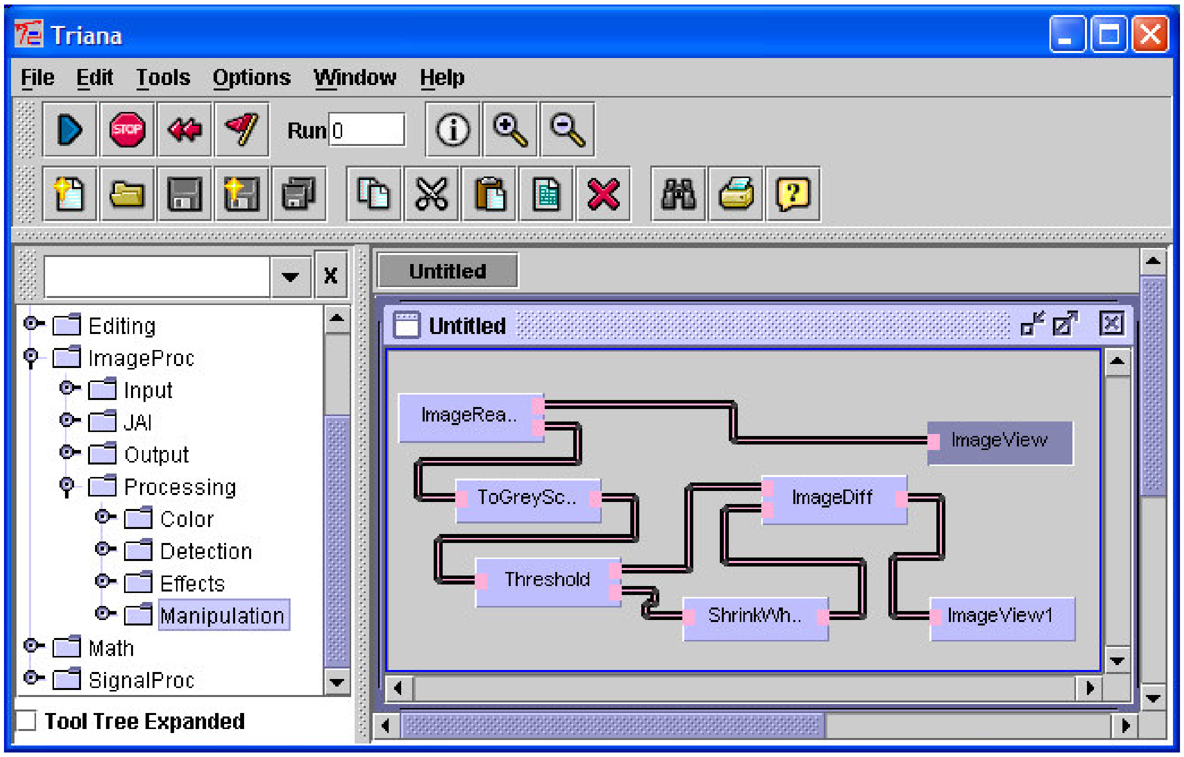


Figure 1: Image processing using Triana. This network extracts the edges from an image.

Triana can be used by applications in a variety of ways through the use of its 'pluggable software architecture'. For example, it can be used as a: graphical workflow composition system for grid applications; a data analysis environment for image, signal or text processing; and it can be used as

[4] GEO 600 Home Page: http://www.geo600.uni-hannover.de/

an application designer tool, creating stand-alone applications from a composition of components. The Triana user interface consists of a collection of toolboxes (see left panel on Figure 1) containing the set of Triana components and a work surface, for composition (see right panel). Within Triana you program graphically rather than writing source code to implement the behaviour you require. It has many of the key programming constructs e.g. looping e.g. do, while, repeat until, and logic units e.g. if, then etc that can be used to graphically control the data-flow, just as a programmer would control the flow within a conventional program using specific instructions. In this sense, Triana is a graphical programming environment. Programming units (i.e. tools) include information about which data-type objects they can receive and which ones they output and Triana performs dynamic run-time type checking on requested connections to ensure data compatibility between components serving the same purpose as the compilation of a program for compatibility of function calls.

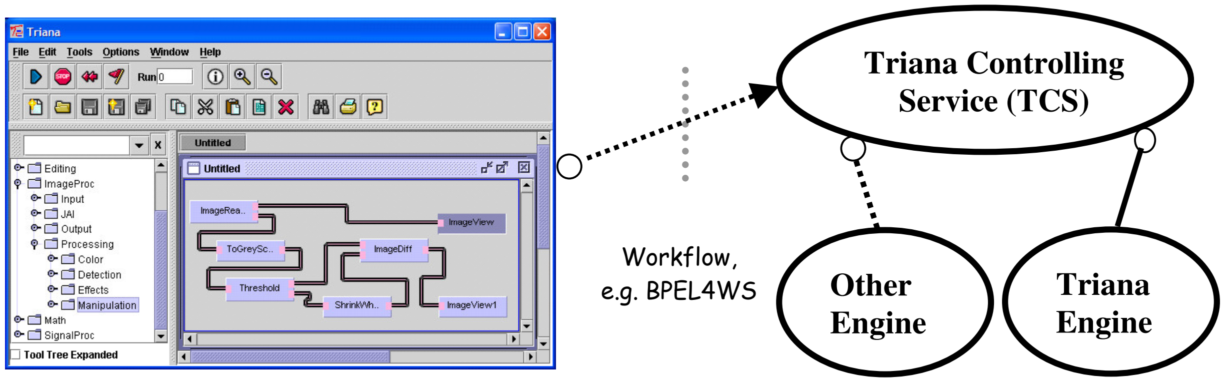


Figure 2: The Triana pluggable architecture. Applications can plug into any of the insert points (indicated by the white circles).

Triana is divided into a set of modularized pluggable interacting components. Briefly, the Triana GUI is a thin-client that connects to the Triana engine either locally or via the network (see Figure 2). Clients can log into a Triana Controlling Service (TCS) remotely build and run a Triana network and then visualize the result on their device even though the visualization unit itself is run remotely. Clients can log off without stopping the execution of the network and then log on at a later stage to view the progress (perhaps using a different device e.g. mobile phone, handheld). In this context, Triana can be used as a workflow monitoring system for Grid applications. Further, since any Triana network can be run with or without using the Triana GUI, Triana networks can be run as executables in a standalone mode. This architecture supports multiple usage scenarios. Programmers can use Triana at various levels by being able to plug in their own code at any of the insertion points within the system (indicated in figure 2 by the white circles). At these points, Triana reader and writer interfaces allow the integration of tools, task-graphs (e.g. BPEL4WS, WSFL) and GUI commands. There are 3 main ways of using Triana, by:

1. Using the Triana GUI directly on top of an existing application. The reader/writer interfaces can be used to connect Triana easily with existing applications. There are command writers (exporting the commands from the GUI) and workflow writers (exporting the taskgraph of the composed components).
2. Using the remote control facility to an existing application. Programmers can use Triana in the same way as above but take advantage of the remote control facility for logging on and

off to the TCS. Existing applications can then be inserted behind the TCS on the remote machine. Here, the scheduling will implemented by a third party system e.g. Condor

3. Modularizing their application into a collection of cooperating Triana units. This is by far the most advanced form of usage as the application becomes far easy to prototype and extend (by adding extra components) and can be seamlessly distributed using the Triana distribution mechanisms (see next section). In the next section, we describe how we used this mechanism to implement a distributed version of a galaxy formation visualization code.

3. Creating Distributed Applications Using Triana

3.1. Galaxy Formation Visualization Background

Galaxy and star formation using smooth particle hydrodynamics generates large data files containing snapshots of an evolving system stored in 16 dimensions. These dimensions are things like particle positions, velocities, and masses, type of particle, and a smooth particle hydrodynamic radius of influence. After calculation each snapshot is entirely independent of the others allowing distribution over the Grid for independent data processing and graphic generation. Processing may involve simple view port reorientation or more complex line of sight mass density calculations. A simple small graphic can then be returned to the client chronologically for animation.

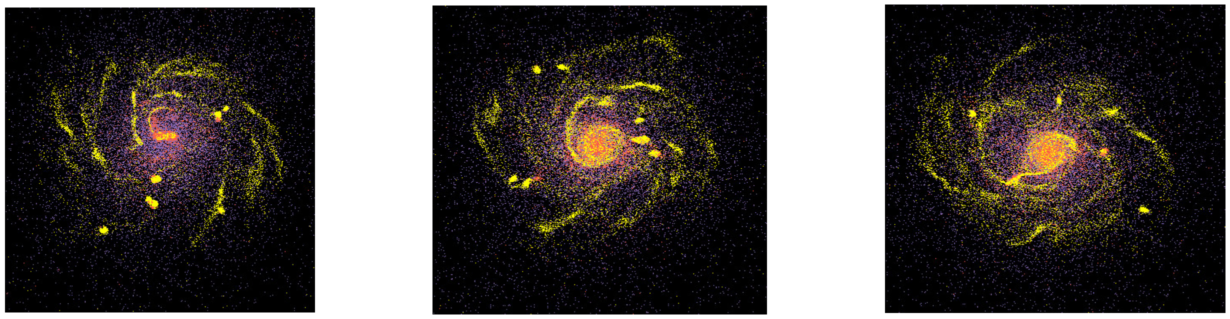


Figure 3: Typical Images as the result of a Galaxy Simulation. This particular calculation is the result of evolving 120000 sample points: 40000 gas, 40000 Baryonic matter, 40000 star forming particles, courtesy A. Nelson, P. Williams and R. Philp, Cardiff University.

The user of the code would like to visualize this data as an animation in two dimensions with the ability to vary the point of view, project that particular two dimensional slice and re-run the animation (see Figure 4). Due to the nature of the data, each frame or snap shot is a representation at a particular point in time of the total data set, it is possible to distribute each time slice or frame over a number of processes and recalculate the different frame view based on the projection point, in parallel. One of the main reasons for distributing the processing of these frames is their shear size. Currently a simplistic simulation with say a million particles may have a raw data frame size of 60 Mbytes, with an overall data set size of the order of 6GBytes, hence making the overall computation of the graphics themselves very intensive and difficult to do in real time on a sequential computer. However, the frames are independent once they are produced and can be processed individually, although at viewing they do have to be put into the correct chronological order.

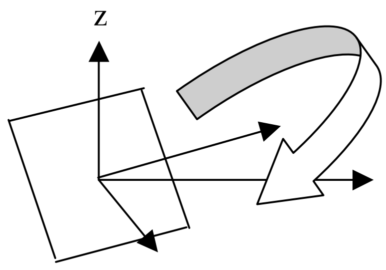


Figure 4: The user changes the view port and the 3-D simulation is projected onto a 2-D plain for viewing.

The original visualization application was written in Java as a standalone application. The Triana team worked to modularize this code into a set of Triana units. Triana then enables the composed application to be distributed across the Grid. The next section describes the modularization process in more detail.

3.2. Modularizing the Galaxy Formation Code

The process consists of three distinct phases, first the data files are parsed according to their format and the data loaded into data structures. Each data segment represents a distinct time frame or snap shot within the animation. The 3D data sets are then projected down onto a 2D plane from a viewpoint using a calculation. Finally the 2D frames are run together to form an animation. The inherently modular nature of the process together with the discrete nature of the data in the data sets make this a good candidate for splitting into components.

The first component in the implementation (see Figure 5) encapsulates the first phase of the process. The simulation data file loader (DataFrameReader) takes the file and generates a series of data structures representing the time frames in the data set and outputs them. The second phase is the calculation of the two dimensional projection and generation of an image this is the functionality of the ViewPointProjection unit. As Triana already contains an image viewer unit (ImageView) that will animate if it receives consecutive images the third phase of the process is simple as long as the calculation component can generate a pixel map image.

The whole original process can be modeled using the three components and connecting them together. We only have to create two components because of Triana's extensive library of components. The two new components were built using Triana's built in tool builder to generate code skeletons and then edited to take the appropriate code with very little modification from the original application. It is possible to extend the process and buffer the data for future calculations based on a new view point using the Sequence unit, this is a simple data player component that saves us having to run the DataFrameReader component multiple times, which depending on data size can be expensive.

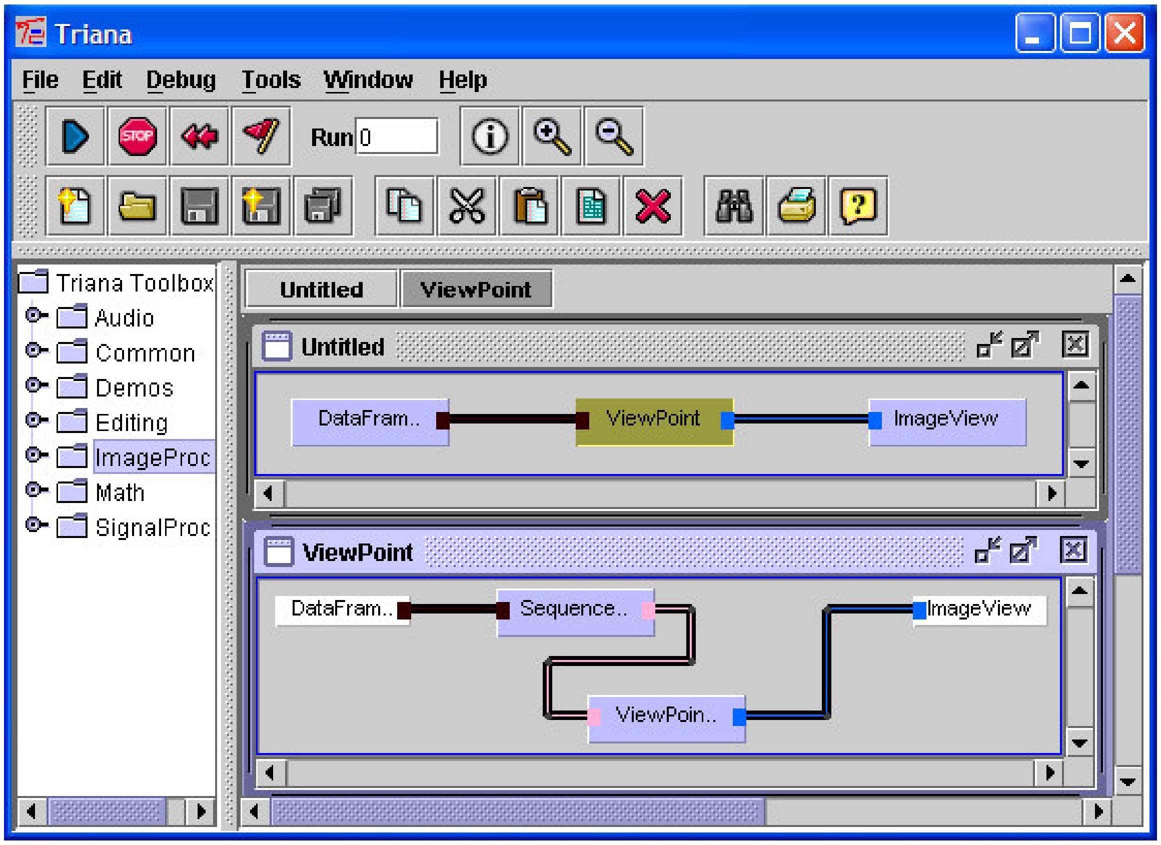


Figure 5: The Galaxy Formation Code implemented as a set of Triana units. The upper workspace containing three components is an unbuffered player: it reads data from the data file every time the player is activated. The lower workspaces stores the data frames in a sequence buffer for reuse.

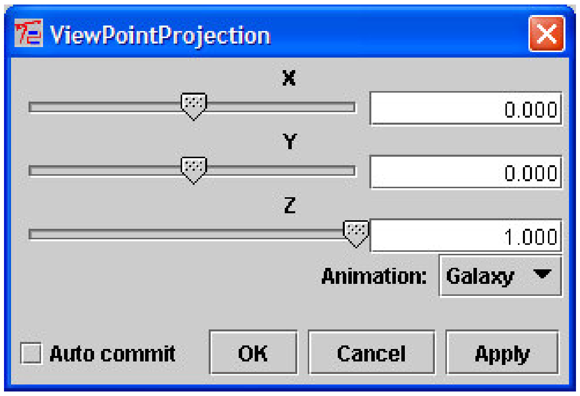
3.3. Distributing the Galaxy Formation Data

The loaded data is divided into frames, distributed amongst the various Triana servers on the available network and processed to calculate whichever analysis is required, for instance this might be a simple viewing of the morphology of the Galaxy or it might be a much more numerically intensive analysis of calculating the column density using smooth particle hydrodynamics: essentially how much mass there is along a line of sight assuming an optically thin Galaxy.

Currently the distribution of the data amongst the Triana servers is simple: the local thin client locates a data file, and then read the data from each frame and then distributes the data to a Triana server for processing. This is inefficient in two respects. First, there is a high startup cost in distributing the data: given the discussions above. Secondly, there is no inherent resiliency in the network: if a node dies what happens then? In the longer term we propose to overcome these problems as follows. First to take advantage of the nature of P2P computing we intend to distribute the data at the outset (at creation time) to a number of data repositories that will act as a data service provider. Then we intend to modify the paradigm for the data creation to include a metadata layer. The metadata will contain information relating not only to the internal structure of the data and the data frame offsets but also to the repository locations. The user at the thin client will then access the metadata file instead of the actual data. The thin client will then issue repository locations and data frame offset references to the remote Triana servers. The servers then locate the data repositories and only load those frames with the given frame offsets. The advantage of this is that not all servers will access the same repository thus reducing disk access and network bottlenecks. It is also intended to provide a robustness to the system by issuing instructions to the remote server to not only load their data frames but to load some of the data frames assigned to other servers. It is envisaged that the thin client will be able to determine the failure of any server and reassign its frames if need be: if a server is lost then there is automatic redundancy built into the network. The ability to distribute the data, particularly in the cases of Galaxy and Star Formation, is due to the fact that runtimes currently are measured in months of CPU time and so the time average load on the network is in fact kept to a minimum. The instructions as to what the remote Triana servers then do with the data are handled by the remote application steering in the next section.

3.4. Remote Application Steering

There are two separate user interfaces that allow the remote steering of the Galaxy Formation test case. One is the generic *SequenceBuffer* tool and the other is the user interface from the *ViewPointProjection* unit. Every Triana unit that implements a user interface can be viewed remotely. In this case by setting the new X, Y and Z parameters in *ViewPointProjection*, the user can simultaneously update these values on all nodes. Similarly, for the Sequence Buffer, the animation can be remotely started simultaneously by triggering each node i.e. by pressing the start button.

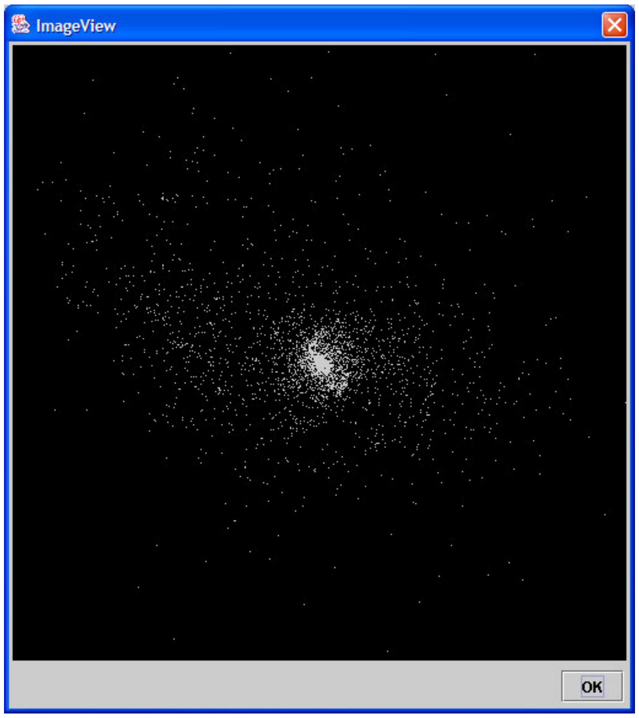

ViewPointProjection

X: 0.000

Y: 0.000

Z: 1.000

Animation: Galaxy



The *ViewPointProjection* unit's user interface on the user's local machine is used to steer the entire process. The user can select the precise view point using the given coordinates. If the user wants a different view of the data he changes these coordinates and presses the start button. Messages are then sent to all the distributed servers so that the new data slice through each time frame can be calculated and returned. Each distributed Triana service returns it's processed data in order and the frames are animated. A Triana visualization unit (ImageView) then displays the resultant animation as a sequence of GIF files. The result is that the user can visualize the galaxy formation in a fraction of the time it would if the simulation was performed in a single machine.

Opposite, a snapshot showing one time step of the projected data set is shown.

4. Triana Distributed Implementation and the Role of The GAT

4.1. Distribution Using Triana

Each Triana Controlling Service (TCS) has a corresponding Triana engine (or a 3rd party engine) that are implemented as a Triana service that are capable of executing complete or partial task-graphs. Triana engines can choose to execute the task-graph locally distribute it to other Triana servers according to the desired distribution policy and can communicate with each other to offer pipelined work-flow distributions.

The Triana distributed implementation is based around Triana *Group* units, that is, aggregate tools containing a number of interconnected units. They have the same properties as normal tools e.g. they have input/output nodes, properties etc, and therefore, they can be connected to other Triana units using the standard mechanism. Tools have to be *grouped* in order to be distributed and have an associated distribution policy. A distribution policy describes the particular way in which the units in this group are distributed. There are two distribution policies currently implemented in Triana:

- *Parallel: Parallel* is a 'farming out' mechanism (see figure 3) and generally involves no communication between hosts.
- *Pipeline*. *Pipeline* involves distributing the group *vertically* i.e. each unit in the group is distributed onto a separate resource and data is passed between them in a pipelined fashion.

Groups can contain groups and each group can have its own distribution policy. This means that complex hierarchical distribution mechanism can be specified. For example, figure 3 illustrates this: Here, one Triana Service distributes its group to three other Triana services using the task farming distribution policy then each of these Triana services act as a gateway and distribute their task graph (implemented by a subgroup) to two other services using the pipeline distribution policy.

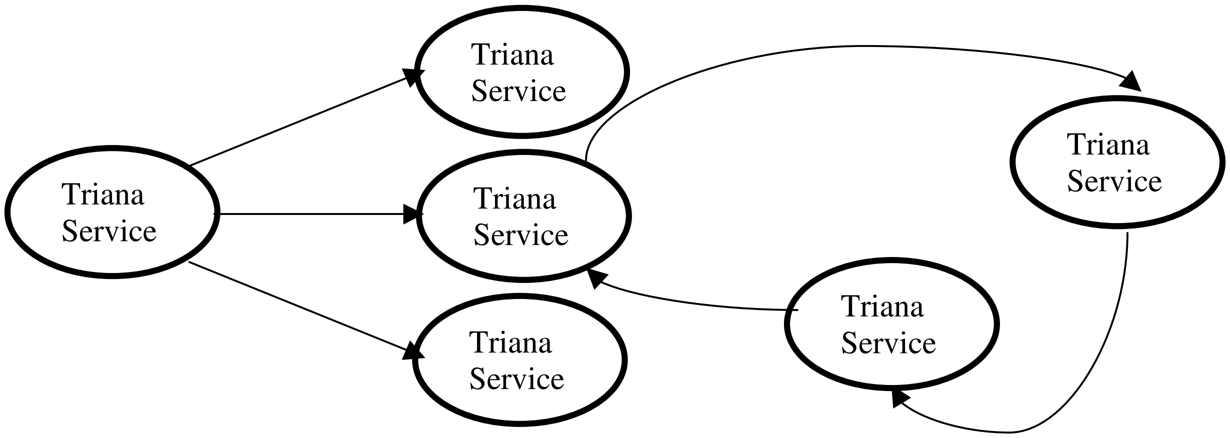


Figure 6: Example Triana distribution: a service distributes a task-graph to three other Triana services using task-farming then each of these distributes their task graphs to another two services using a pipelined approach.

4.2. The GAT

The Triana distribution mechanisms are based upon the concept of a GAT (Grid Application Toolkit). The purpose of a GAT is to shield applications from the implementation details and complexities of underlying Grid middleware through providing a standard application programmer's interface (GAT-API) and a set of common Grid services for tasks such as resource management and information management. From Triana's perspective a GAT should provide simple API calls for discovering and communicating with Triana services running on remote machines, and these calls should be independent of the current GAT middleware binding, enabling Triana to be distributed on different Grid middleware without modifying the core Triana code.

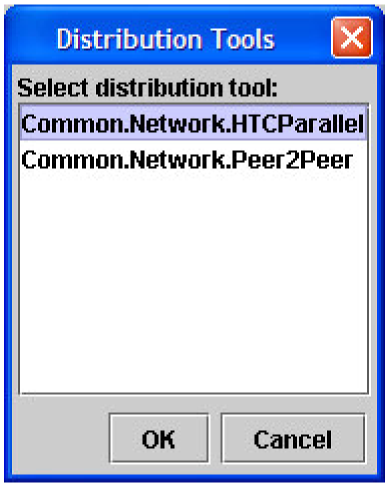
The Triana project is a partner in the E.U. funded GridLab project [8], a pan European project developing the GridLab GAT. The GridLab project is split into 14 work packages developing the GAT-API and Grid services in areas such as security, logging and data visualization. The role of Triana is as a test application and to specify and develop application requirements from the GridLab GAT.

To enable the distribution of Triana in preparation for the GridLab GAT, a prototype GAT API called the GAP Interface (Grid Application Prototype Interface) was created. The GAP interface provides a simple set of calls required for locating and communicating with remote services, a subset of the functionality expected in the GridLab GAT. There are currently GAP Interface bindings for JXTA and also for a lightweight socket based peer-to-peer implementation (P2PS). For the galaxy formation demo we chose to use JXTA binding as the distribution mechanism.

The JXTA GAP binding implements the basic GAT functionality for locating and communicating with JXTA services. A JXTAServe service has one or more input nodes (one is needed for control at least) and has zero, one or more output nodes. It advertises its input and output nodes as JXTA pipes and connects between pipes using the virtual communication channel that adapts to the particular communication protocol depending on the current operating environment.

4.3. Using Triana to Task Farm

Once the particular data flow has been created, sections of this data flow can be distributed to running Triana servers using any of the distribution policies described in section 4.1. For our galaxy formation example, we used the Parallel distribution policy to *task farm* the data sets across all available nodes in order to speed up the recalculation of the visualization of data set. To achieve this, first we selected the section of the task-graph that we wanted to distribute. As shown in Figure 5, we distributed the SequenceBuffer unit (a node buffering mechanism) and the ViewPointProjection unit that performs the recalculation of the data set for the new viewing angle. We then made a ***Group Unit*** out of the selected units and called this ViewPoint. Right clicking on the group then brings up a menu where you can select the enabling of the distribution for a group unit. Once this is selected, the window to the right appears prompting you to enter the distribution policy you wish to select for this group. We selected HTCParallel: the High Throughput computing Parallel distribution implementation within Triana. Triana dynamically

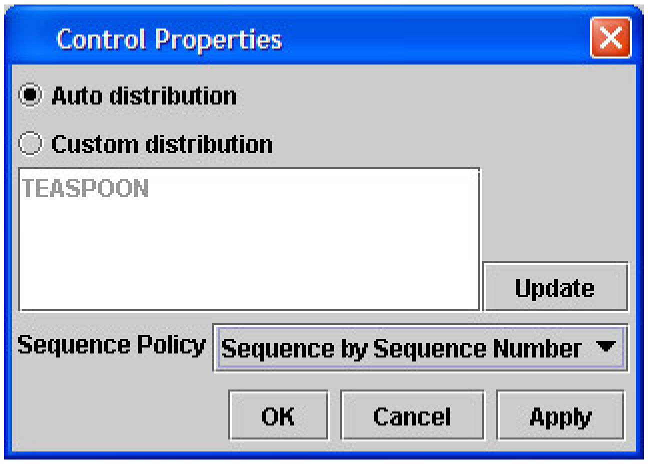

Distribution Tools

Select distribution tool:

Common.Network.HTCParallel

Common.Network.Peer2Peer

discovers distribution policies and new policies can be added easily by creating new Control units that specify the specific policy. We expect to implement more as we apply Triana to new problem domains.


Control Properties

[✓ Auto distribution] Custom distribution

TEASPOON

Sequence Policy: Sequence by Sequence Number

We then click OK and the Window to the left is displayed, which shows all available services (in this example we just ran one on a laptop for demonstration purposes) along with a number of options. Users can select *Auto Distribution* allowing Triana to automatically utilize the available services itself or *Custom Distribution* to apply a custom distribution on a specific subset of the servers listed. A *Sequence Policy* must then be selected. This specifies how Triana will reassemble the packets when they return to the client. For example, in our galaxy formation example, it is imperative that the GIF files for the animation are returned in the same order than they were distributed although in practice this will almost certainly not be the case. Specifying *Sequence by Sequence Number* ensures that the packets are output in the same order that they were received by buffering results that appear out of sequence. Alternatively, one can choose not to sequence the packets.

5. Open Source Triana Release

On May 30th 2003 Triana was released as an open source software package. This version includes the code described in this paper, both for the galaxy formation visualization and for the distributed Triana prototype. At the time of writing Triana includes two GAT bindings, one is implemented in JXTA and the other is a lightweight native Java socket based on P2P mechanism (P2PS). The user is prompted at start up to choose how to run Triana i.e. stand alone, as a JXTA peer or a P2PS peer.

6. Conclusion

In this paper, we presented an overview of how application scientists would use the Triana software environment to componentize their application and how they might use the Triana distribution mechanisms to distribute this onto the Grid. Further, since Triana uses the GAT interface to insulate it from the underlying middleware, the application developer does not have to commit to any underlying middleware when they are developing the Grid version of their application. At the time of writing, Triana is just about to be released as an open source project and we currently have a number of varied application groups that we are working with to use Triana for their Grid solutions. This integration will feed back directly into the Triana source tree for future releases (currently, we have around 500 tools implemented) and therefore enable wider applicability into new areas of research.

References

1. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. I. Foster, C. Kesselman, J. Nick, S. Tuecke, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
2. IBM and Globus Announce Open Grid Services for Commercial Computing, http://www.ibm.com/news/be/en/2002/02/211.html
3. The Globus Project: http://www.globus.org/
4. Gnutella : http://gnutella.wego.com/
5. SETI@Home : http://setiathome.ssl.berkeley.edu/
6. JXTA: http://www.jxta.org/
7. In The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C Kesselman, in *Computational grids*, Eds. Morgan Kaufmann Publishers, July 1998, ch. 2.
8. GridLab : http://www.gridlab.org
9. GridOneD project home page and the Triana Software Environment web site: http://www.gridoned.org/ and http://www.trianacode.org/
10. SGT Toolkit: http://www.epic.noaa.gov/java/sgt/index.html
11. XMGrace: http://plasma-gate.weizmann.ac.il/Grace/
12. Distributed P2P Computing within Triana: A Galaxy Visualization Test Case, Dr Ian Taylor, Matthew Shields, Dr Ian Wang, Dr Roger Philp, in proceedings IPDPS 2003, 22-26 April 2003, IEEE CD-ROM