

Performance Prediction Based Resource Selection in Grid Environments

Peggy Lindner¹, Edgar Gabriel², and Michael M. Resch¹

¹ High Performance Computing Center Stuttgart, University of Stuttgart,
Nobelstr. 19, 70569 Stuttgart, Germany
{lindner, resch}@hllrs.de

² Parallel Software Technologies Laboratory,
Department of Computer Science, University of Houston,
4800 Calhoun Road, Houston, TX 77204, USA
gabriel@cs.uh.edu

Abstract. Deploying Grid technologies by distributing an application over several machines has been widely used for scientific simulations, which have large requirements for computational resources. The Grid Configuration Manager (GCM) is a tool developed to ease the management of scientific applications in distributed environments and to hide some of the complexities of Grids from the end-user. In this paper we present an extension to the Grid Configuration Manager in order to incorporate a performance based resource brokering mechanism. Given a pool of machines and a trace file containing information about the runtime characteristics of the according application, GCM is able to select the combination of machines leading to the lowest execution time of the application, taking machine parameters as well as the network interconnect between the machines into account. The estimate of the execution time is based on the performance prediction tool Dimemas. The correctness of the decisions taken by GCM is evaluated in different scenarios.

1 Introduction

During the last couple of years, Grid computing has emerged as a promising technology for science and industry. Computational Grids give users the power to harness spare cycles on PCs or workstations or distribute data and compute intensive applications on a large number of geographically distributed machines. Among the key challenges of current Grid technologies is the problem of resource selection and brokering. The set of resources chosen for a particular job can vary strongly depending on the goals of the researchers, and might involve minimizing the costs, optimizing the location of the computational resources in order to access a large data set or minimizing the overall execution time of the job.

In this paper we present a novel approach for resource selection based on the estimation of the execution time of an application using a simulator. Given a pool of machines, the Grid Configuration Manager (GCM) tests several possible combination of machines fulfilling the requirements of the end user regarding

the required computational resources. The estimate for the execution time of the application is based on the performance prediction tool Dimemas [1]. After GCM determined the combination of machines with the lowest predicted execution time, it can automatically generate the required configuration files for PACX-MPI [6] jobs and launch the distributed simulation.

The distribution of an application onto heterogeneous, dynamically changing resources requires the complex coordination of the resources. Currently, there is no general strategy or solution to the scheduling problem in a Grid environment available which meet all the demands [12]. The currently available brokering and scheduling systems are usually very specific for the target systems [15] or tailored for special application scenarios [10]. Work on interoperability includes Grid projects targeting resources running different Grid middlewares [3,14] and projects using (proposed) standard formats, e.g., to describe computational jobs [3,9]. The UniGrids [11] project specifically targets inter-operation between Globus and UNICORE. The difference between the GCM and these projects is that we also target the use of traditional non-Grid resources.

The remainder of the paper is organized as follows: section 2 presents the main ideas behind GCM, section 3 introduces the performance prediction tool Dimemas. In section 4 we detail the integration of Dimemas and GCM. Section 5 evaluates the correctness of the decision taken by GCM for various configurations and application settings. Finally, section 6 summarizes the paper.

2 The Grid Configuration Manager GCM

The Grid Configuration Manager (GCM)[7] is a tool developed to hide some of the complexity of Grid environments from the end-user. The central objective is to ease the handling of scientific, computational jobs in heterogeneous Grid environments, by abstracting the necessary tasks and implementing them for the most widely used Grid software toolkits (Globus[4], UNICORE[13]) and for the most widespread traditional access mechanism, SSH. To hide the complexity of different Grid protocols, the Grid Configuration Manager provides methods to support users in dealing with three main problems: handling of different authentication/authorization methods, a common interface for the job description and steering, and support of file transfer mechanisms for post- and preprocessing. The GCM idea is based on individual models for the resource description of the participating resources. Two different abstract classes were defined: the *host class* and the *link class*. The *host class* describes the properties of a resource such as name of the machine, user authentication information, number of processes to be started on this machine, or the start command and path of the application on this machine. The *link class* contains information about the network connection between two hosts, e.g. number of connections between the hosts and network parameters for each connection (i.e. network protocol and port numbers to be used). The current implementation of the GCM does not include an automatic resource discovery mechanism. This means, that the user has to specify how many nodes are available on the hosts to run his job. In the

future this information will be gathered by requesting this information from the batch system for ssh based machines or using information services offered by Unicore and Globus.

A key component to create advanced applications in Grid environments is the file transfer component. File transfer operations are required for staging data files and executables, but also to distribute configuration files required by various tools and libraries. The file transfer component developed for GCM is based on the graphical Network Interface to File Transfer in the Internet (NIFTI). Given its modular architecture, NIFTI is conceptually independent of the implementation of the services it represents, and currently interfaces alternatively to FTP, SCP or an UNICORE file services.

One of the original goals of GCM from the very beginning was to support parallel multi-site jobs, since distributing jobs onto several machines simultaneously is an important part of the Grid concept. The need for doing so mainly comes from applications, which have a high demand on computational resources or for increasing throughput on the machines and reducing turnaround times. The GCM supports the handling of a multi-site job for a communication library called PACX-MPI [6]. While GCM is not restricted to PACX-MPI, the defined abstract interface has been up to now implemented to support the PACX-MPI configuration files and startup procedures.

3 Dimemas

Dimemas [1] is a performance prediction tool for message passing applications develop at the Barcelona Supercomputing Center (BSC). Within the frame of the European DAMIEN project [5], Dimemas has been extended to support performance prediction of distributed applications in Grid environments. Based on trace-files of the application, the tool is capable of estimating the execution time of the very same application for varying networking parameters or processor speeds. In order to generate a trace-file, the user has to re-link its MPI application with a tracing library provided by Dimemas, and run a small number of iterations of the application on the required number of processes. The tracing libraries intercepts each call to a communication routine using the MPI profiling interface [8], and stores relevant information such as the sender and receiver processes of the message, message length and time stamps for the beginning and the end of the data transfer operation. Thus, Dimemas offers a scalable and fully automatic approach for generating the trace-files, which has been used for highly complex, parallel applications.

The Grid architecture supported by Dimemas consists of individual nodes which can be grouped to a machine. Each node can have one or multiple, identical processors. Multiple processors are connected by a network interconnect. Based on a GUI, the user can modify the performance of the network as well as specify the no. and the relative performance of the nodes. The network connection between different machines can be based either on a shared network resources, i.e. the internet, or on dedicated network connections. The first approach is based on a congestion

function, a latency respectively flight time and a maximal bandwidth. Values for latency and bandwidth of a message are adapted at runtime based on the congestion function provided by the user. The second approach allows the precise definition of a latency and the bandwidth between the machines. However, the current version of the tool is restricted to the same latency/bandwidth values between all machines within a given Grid configuration.

4 Integration of GCM with Dimemas

Among the most challenging issues the user has to deal with in current Grid environments is the resource selection step. In order to ease the resource selection procedure, GCM has been integrated with the performance prediction tool Dimemas. The goal of this approach is to enable the automatic selection of the most efficient Grid configuration for a particular application within a given set of resources. To encapsulate the required data for Dimemas, GCM has been extended by a new ‘Dimemas-model’. This model contains all the configuration details and necessary information to run the Dimemas simulator program. The *host* and *link classes* of GCM have been extended to collect the additional information such as the specification of the available SMP-nodes on each machine, a detailed description of the network connection between the machines, location of the trace file etc. A special ‘Mapping-Dialog’ allows the user to specify how the application processes should be distributed across the machines. The class *Dimemas* contains all methods to run a Dimemas simulation and store the input data and results for future usage in a special metadata format. Special GUI interfaces help the user to specify all the information. The results of the Dimemas simulation and (error) messages can be retrieved on a special output window in order to allow the usage of GCM as a GUI interface for Dimemas.

To propose an optimal Grid configuration, GCM takes into account the given resources, the currently available nodes on each machine and the communication pattern of the machine respectively between the machines. Therefore, the user has to provide information about the application in form of a trace file which will be used by Dimemas. Based on this information, GCM creates different ‘test configurations’ for Dimemas, starts the Dimemas simulator, stores and compares the results achieved. The Grid configuration with the lowest estimated execution time will be offered to the user in form of a PACX-MPI configuration. 1 shows the individual steps of the resource selection within GCM.

Since the number of potential process distributions on the available machines can be fairly large, GCM creates test configurations based on a simple heuristic. The distribution of processes onto the hosts follows a simple round robin algorithm. For each available machine, GCM generates a test configuration using that machine as the starting point and allocates all available processors on that machine for the simulation. In case more processors are required in order to run the simulation, GCM takes the required number of processors from subsequent machines in sequential order, etc. Using this approach, n test configurations are

generated for a machine pool containing n machines. An advantage of this heuristic is, that since we assign the maximum number of processes on each machine to the simulation, the number of machines used for each test configuration are being minimized. For parallel, distributed applications this kind of configuration usually produces the lowest execution times, as the communication between the machines is fairly time consuming. As an example, lets assume that 16 processes should be distributed onto 4 machines, each of which offer 6 processors. According to the described algorithm, the following test configurations will be created:

- 6 processes each on machines 1 and 2; 4 processes on machine 3
- 6 processes each on machines 2 and 3; 4 processes on machine 4
- 6 processes each on machines 3 and 4; 4 processes on machine 1
- 6 processes each on machines 4 and 1; 4 processes on machine 2

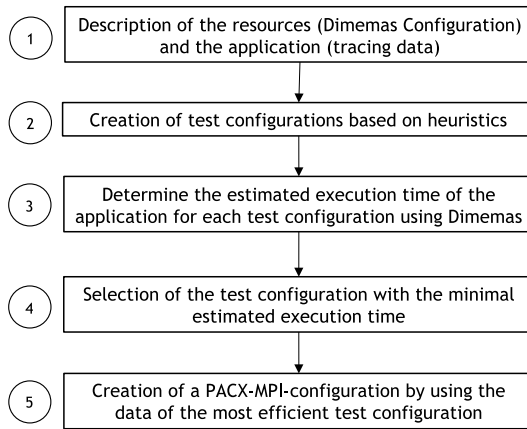


Fig. 1. Resource selection based on the estimated execution time of the application within GCM

The simulations based on Dimemas estimating the execution time of the application for different network and machine parameters are executed on the very same a local PC where GCM is being executed. Depending on the size of the trace file, a single simulation takes at most a couple of seconds on a state-of-the-art PC, and does not require any Grid resources itself.

5 Evaluation

In the following, we would like to verify the correctness and the accuracy of the choices made by the Grid Configuration Manager. The application used throughout this section is a finite-difference code which solves a partial differential equation (PDE), respectively the set of linear equations obtained by

discretization of the PDE using center differences. To partition the data among the processors, the parallel implementation subdivides the computational domain into rectangular subdomains of equal size. Thus, each processors holds the data of the corresponding subdomain. The processes are mapped onto a regular three-dimensional mesh. Due to the local structure of the discretization scheme, a processor has to communicate with at most six processors to perform a matrix-vector product.

The Grid configurations used in this subsection consists of a collection of homogeneous resources based on the technical data of local InfiniBand clusters of the participating institutions. This approach allows us to investigate the effect of different parameters on the decision procedure of GCM individually. As an example, a configuration such as shown in table 1 consists of four machines, each of them being from the technical perspective identical. Unless explicitly mentioned otherwise, the communication bandwidth between the machines is set to 12MB/s and the latency is 4 ms for the Dimemas simulations, which are typical values for Fast Ethernet connections.

Table 1. Estimated execution times by Dimemas of the Finite Difference Code for various test configurations. Times are given in seconds.

Test 1: 16 processes					Test 3: 16 processes				
No. of processors on machine				estimated execution time (s)	No. of processors on machine				estimated execution time (s)
1	2	3	4		1	2	3	4	
16	0	0	0	1.17 *	10	6	0	0	10.08 *
0	6	6	4	7.41	0	6	6	4	12.19
4	0	6	6	7.41	6	0	6	4	12.22
10	0	0	6	5.13	10	2	0	4	12.23
Test 2: 32 processes					Test 4: 32 processes				
6	26	0	0	8.71	10	12	10	0	10.69
0	32	0	0	4.44 *	0	12	20	0	7.99 *
6	14	6	6	10.52	6	0	22	4	11.22
6	20	0	6	10.52	10	12	6	4	10.47

The first set of tests have been set up in order to verify, that the configuration chosen by GCM/Dimemas is using the minimal number of machines to run an application. Since the communication between machines is at least an order of magnitude slower than the communication between processes on the same, minimizing the number of machines used by an application will in the vast majority of scenarios also minimize the overall execution time of the application. Of special importance are scenarios where the application could run on a single machine and thus would completely avoid expensive inter-machine communication. The results of the simulations are shown in table 1. GCM was configured such that its pool of machines consisted of four identical clusters. In the tests 1 and 2 (left) one machine advertises sufficient available resources to run the simulation

entirely on that platform, while using any of the other machines would require a combination of multiple machines in order to provide the requested number of processors. Similarly, in the tests 3 and 4 only a single combination of available resources leads to a two-host configuration capable of providing the number of requested processors. Table 1 shows for each test case the combination of resources respectively the number of processes on each platform used by GCM to find the minimal execution time and the estimated execution time by Dimemas. The combination of resources selected by GCM is marked by a star. In all scenarios, GCM/Dimemas was able to spot the combination requiring the minimal number of machines, by either using a single machine (tests 1 and 2), or two machines (tests 3 and 4).

Table 2. Estimated execution times by Dimemas of the Finite Difference Code for various test configurations with varying bandwidth between the machines. Times are given in seconds.

Bandwidth	No. of processors on machine					estimated exec. time (s)	No. of processors on machine					estimated exec. time (s)
	1	2	3	4	1		2	3	4			
	Test 5: 24 processes						Test 6: 32 processes					
100 MB/s	12	12	0	0	6.601620 *	16	16	0	0	6.937863 *		
25 MB/s	0	12	12	0	7.361563	0	16	16	0	6.958964		
30 MB/s	0	0	12	12	7.052505	0	0	16	16	6.939202		
95 MB/s	12	0	0	12	6.601623	16	0	0	16	6.937885		

In the second test we would like to evaluate, whether our approach is correctly determining the set of resources offering the best network connectivity between the machines. Two separate set of tests have been conducted. In the first test, we keep the latency between all machines constant, vary however the available bandwidth between each pair. As shown in table 2, all four machines advertise the same number of processors, namely either 12 in test 5 or 16 in test 6. Since the user requested however 24 respectively 32 processors, GCM has to find the combination of two machines promising the lowest execution time. The bandwidth between the according pair of machines is shown in in the first column of table 2.

As can be seen on the results shown in table 2, this particular application shows only limited sensitivity to the bandwidth between the machines. All execution times estimated by Dimemas end up in the very same range. Nevertheless, in both scenarios GCM did chose the combination of resources with the highest bandwidth, namely machines 1 and 2.

Similarly to the previous scenario, we kept this time the bandwidth between all four machines constant at 12 MB/s. However, we varied the latency between the different machines between 0.6 ms and 5 ms. The first column in table 3 shows the latency value used between the according pair of machines. The results indicate, that this finite difference code is significantly more sensitive to the network

Table 3. Estimated execution times by Dimemas of the Finite Difference Code for various test configurations with varying latency between the machines. Times are given in seconds.

Latency	No. of processors on machine				estimated exec. time (s)	No. of processors on machine				estimated exec. time (s)
	1	2	3	4		1	2	3	4	
	Test 2: 24 processes					Test 4: 32 processes				
4 ms	12	12	0	0	9.59947	16	16	0	0	7.414597
5 ms	0	12	12	0	10.480678	0	16	16	0	7.948321
0.6 ms	0	0	12	12	6.612363 *	0	0	16	16	5.660996 *
0.8 ms	12	0	0	12	6.787963	16	0	0	16	5.759201

latency between the machines than to the inter-machine bandwidth. GCM could correctly identify the combination of machines with the lowest network latency.

5.1 Using GCM and Dimemas to Evaluate Implementation Options for Grid Environments

An alternative usage scenario for GCM together with Dimemas is to evaluate the performance of different implementation options given a particular execution environment or Grid configuration. The motivation behind this approach is, that many scientific applications have various options given a particular functionality. As an example, there are many different ways to implement the occurring neighborhood communication in scientific code, each of which might lead to the optimal performance for different problem sizes and machine configurations [6]. Similarly, a application often has the choice to use different linear solvers, or multiple algorithms with slightly different characteristics for a particular solver. Using GCM together with Dimemas gives the code developers and end-users a powerful tool to get very realistic estimates about the performance of each option on various machines respectively set of machines, without having to run the actual code on all possible combination of machines.

This approach is being demonstrated in this subsection using different algorithms for the solver used in the finite difference code. Four different solvers namely QMR , QMR_1 , $TFQMR$, and $TFQMR_1$ are available within the framework. The main difference between these four solvers are the number of collective operations per iteration and the number of synchronization points per iteration [2]. As a first step, the user has to generate tracefiles for each solver and for the number of processors it would like to utilize. This can be done on a single machine, and does not require any distributed resources per se. In a second step, the user generates in GCM the Grid environment of interest, by specifying the number of machines, the number of nodes on each machine and the network characteristic between the machines. GCM will then call Dimemas for each of the according scenarios/traces and suggest the version of the code delivering the best performance. In case the application chooses the solver at runtime based on a setting in an input file, it is straight forward to add a plug-in to GCM which

could alter the input file by choosing the solver which has been determined to deliver the best performance for the given Grid environment.

The main limitation of this approach is due to GCM not having any 'extended' knowledge about the different version being compared. As an example, comparing the execution time of two iterative solvers for a given number of iterations does not include any information about the convergence behavior of the solver. Thus, a decision purely based on the execution time might not necessarily lead to the best solver. However, there are sufficient important scenario, where a useful decision can be made based on the execution time. In the tests shown here, the *QMR* and the *TQMR* solvers expose significantly different convergence characteristics from the numerical perspective. It is however safe to compare the execution times between *QMR* and *QMR*₁ and between *TFQMR* and *TQMR*₁ in order to determine, which implementation performs better in a given environment. The results of such a comparison is shown in table 4. Two different scenarios are shown here requiring 16 processes and 32 processes. The configurations provided to GCM have been chosen such that the processes are evenly distributed on two machines.

Table 4. Results scenario 3 - Comparison of different solvers (* indicates the configuration chosen by GCM)

Solver	No. of processors measured simulated				No. of processors measured simulated			
	on machine	exec time	exec time		on machine	exec time	exec time	
	1	2	(s)	(s)	1	2	(s)	(s)
	TEST 1: 16 PROCESSES				TEST 3: 32 PROCESSES			
<i>QMR</i>	8	8	1.78	1.85 *	16	16	1.42	5.66 *
<i>QMR</i> ₁	8	8	2.78	5.62	16	16	2.35	7.90
	TEST 2: 16 PROCESSES				TEST 4: 32 PROCESSES			
<i>TFQMR</i>	8	8	3.63	4.25	16	16	2.79	8.67
<i>TFQMR</i> ₁	8	8	3.50	3.50 *	16	16	2.66	5.04 *

Table 4 shows the number of processes used on each machine, the avg. execution time of each solver measured on the given environment and the estimated execution time by Dimemas. In all four scenarios, the estimated execution time and the average measured execution times are pointing to the same solver as being optimal/faster. Please note, that within the context of these tests, the accuracy of the estimated execution time compared to the measured execution time is only of limited interest. More relevant is the factor, that the relative performance between the solvers is determined correctly by Dimemas. Since Dimemas is a powerful tool with many options, a user can optimize the settings of Dimemas in order to close the absolute gap between the estimated and the measured execution time. Clearly, the smaller the gap the wider the range of useful scenarios for GCM. In the tests shown in table 4 most of the parameters of Dimemas were the default values delivered by the tool.

6 Summary

We described in this paper the integration of the performance prediction tool Dimemas with the Grid Configuration Manager GCM. Based on a trace-file of the application, this combination of tools can be used in order to optimize the resource selection procedure for a given pool of resources. Using various test-cases we have shown, that GCM together with Dimemas (i) minimizes the number of machines used for a configuration and (ii) does take network parameters such as network and latency between different machines into account when choosing the set of resources for a particular run. Furthermore, we have demonstrated, that this combination of tools can also be used in order to evaluate various implementation/functional options of an application for a given Grid environment without having to actually run the application on a distributed set of resources.

The limitations of this approach are two-fold: first, in order to use the approach outlined in this paper, the user has to generate a trace file of his application. This can be a restriction for some applications. However, since the trace file can be generated on a single machine, this step should not pose any major challenges. Second, the accuracy of the predictions does inherently have an influence on scheduling decisions. Using Dimemas as basis for the performance prediction step gives GCM the most powerful performance prediction tool currently available - in fact, the only tool known to the authors which can be used out-of-the-box for arbitrary complex MPI applications.

References

1. Badia, R.M., Labarta, J., Giménez, J., Escalé, F.: DIMEMAS: predicting mpi applications behavior in grid environments. In: Workshop on Grid Applications and Programming Tools Held in conjunction with GGF8, Seattle, USA Proceedings, June 25, 2003, pp. 52–62. Seattle (2003)
2. Bücker, H.M., Sauren, M.: A Parallel Version of the Quasi-Minimal Residual Method Based on Coupled Two-Term Recurrences. In: Madsen, K., Olesen, D., Waśniewski, J., Dongarra, J.J. (eds.) PARA 1996. LNCS, vol. 1184, pp. 157–165. Springer, Heidelberg (1996)
3. Open Middleware Infrastructure for Europe. OMII Europe
4. Foster, I., Kesselmann, C.: The globus project: A status report. In: Proc. IPPS/SPDP 1998 Heterogeneous Computing Workshop, pp. 4–18 (1998)
5. Gabriel, E., Keller, R., Lindner, P., Müller, M.S., Resch, M.M.: Software Development in the Grid: The DAMIEN tool-set. In: Sloot, P.M.A., Abramson, D., Bogdanov, A.V., Gorbachev, Y.E., Dongarra, J.J., Zomaya, A.Y. (eds.) ICCS 2003. LNCS, vol. 2659, pp. 235–244. Springer, Heidelberg (2003)
6. Keller, R., Gabriel, E., Krammer, B., Müller, M.S., Resch, M.M.: Efficient execution of MPI applications on the Grid: porting and optimization issues. *Journal of Grid Computing* 1(2), 133–149 (2003)
7. Lindner, P., Gabriel, E., Resch, M.M.: GCM: a Grid Configuration Manager for heterogeneous Grid environments. *International Journal of Grid and Utility Computing (IJGUC)* 1(1), 4–12 (2005)
8. Message Passing Interface Forum. MPI: A Message Passing Interface Standard (June 1995), <http://www.mpi-forum.org>

9. Miura, K.: Overview of japanese science grid project NAREGI. *Progress in Informatics* 1(3), 67–75 (2006)
10. Natrajan, A., Humphrey, M.A., Grimshaw, A.S.: Grid resource management in Legion. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) *Grid Resource Management: State of the Art and Future Trends*, pp. 145–160. Kluwer Academic Publishers, Dordrecht (2004)
11. Riedel, M., Sander, V., Wieder, P., Shan, J.: Web Services Agreement based Resource Negotiation in UNICORE. In: Arabnia, H.R. (ed.) *2005 International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 31–37 (2005)
12. Tonello, N., Yahyapour, R., Wieder, Ph.: A proposal for a generic grid scheduling architecture. Technical Report TR-0015, CoreGRID Technical Report, Institute on Resource Management and Scheduling (2006)
13. Joint Project Report for the BMBF Project UNICORE Plus, Grant Number: 01 IR 001 A-D, Duration: January 2000 to December 2002 (2003)
14. Venugopal, S., Buyya, R., Winton, L.: A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience* 18, 685–699 (2006)
15. Young, L., McGough, S., Newhouse, S., Darlington, J.: Scheduling Architecture and Algorithms within the ICENI Grid middleware. In: Cox, S. (ed.) *UK e-Science Program All Hands Meeting*, pp. 5–12 (2003)