A Metascheduler For The Grid

Sathish S. Vadhiyar and Jack J. Dongarra

Computer Science Department
University of Tennessee, Knoxville
{vss, dongarra}@cs.utk.edu

Abstract. With the advent of Grid Computing, scheduling strategies for distributed heterogeneous systems have either become irrelevant or have to be extended significantly to support Grid dynamics. In this paper, we describe a metascheduling architecture for a Grid system that takes into account both the application and system level considerations. Results are presented to prove the efficiency of the metascheduler.

1 Introduction

There have been number of efforts in devising and/or implementing scheduling strategies for heterogeneous distributed computing systems since the advent of Network of Workstations (NOWs) [1], [2], [3], [4], [5], [6], [7]. The Grid [8] is an abstraction of distributed heterogeneous systems and investigating the relevance of scheduling strategies for heterogeneous systems to Grid environment is a worthwhile effort. The work by Khaled Al-Saqabi et. al [2] considers a 2D array of processors and time slices and assigns the Virtual Processes (VPs) of the jobs to the array. Scheduling based on time slices will lead to huge overhead for the scheduling system when the scheduling strategies have to be invoked frequently in response to frequent Grid dynamics. The Load Sharing Facility [3] lays emphasis on distributing the jobs among the available machines based on the workload on the machines. The assumption that load sharing leads to good response times is not valid in a Grid scenario where the network heterogeneity can significantly affect the execution time of the application.

MARS [4] and more recently AppLeS [7] provide good approaches for application level scheduling in meta computing environments. AppLeS is more suitable for Grid environment with its sophisticated NWS [9] mechanism for collecting system information. However, both MARS and AppLeS do not have powerful resource managers that can negotiate with applications to balance the interests of different applications. The absence of these negotiating mechanisms in a Grid can lead to various problems like the bushel of AppLeS problem [7].

In this paper, we describe a metascheduling architecture that we have been building in the context of the GrADS project [10]. The metascheduler receives candidate schedules of different application level schedulers and implements scheduling policies for balancing the interests of different applications. The goals of the metascheduler include

1. Verifying that the applications made their scheduling decisions based on conditions of the system when competing applications are executing.
2. Accommodating short running jobs by temporarily stopping long running and resource consuming jobs.
3. Facilitating new applications to execute faster by stopping certain competing applications.
4. Minimizing the impact that new applications can create on already running applications.
5. Migrating running applications to new machines in response to system load changes to improve the performance or to prevent performance degradation.

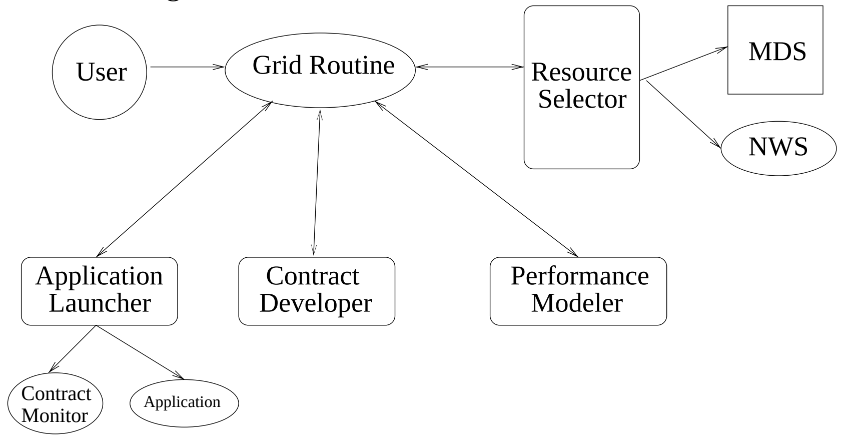
In Section 2, we give a brief overview of the existing GrADS project that utilizes application level scheduling. In Section 3, we describe the metascheduler that we have been building for GrADS environment. We explain in detail the different components of the metascheduler and the mechanisms in the components to achieve the goals mentioned above. In Section 4, we present experiments and results to validate the usefulness of the metascheduler. In Section 5, we compare our metascheduler with related efforts. In Section 6, we present some conclusions. In Section 7, we mention some of the future plans for our metascheduler.

2 The GrADS system

GrADS [10] is an ongoing research involving number of institutions and its goal is to simplify distributed heterogeneous computing in the same way that the World Wide Web simplified information sharing over the Internet. The University of Tennessee investigates issues regarding integration of numerical libraries in the GrADS system. In our previous work [11], we demonstrated the ease in which numerical libraries like ScaLAPACK can be integrated into the Grid system and the ease in which the libraries can be used over the Grid. We also showed some results to prove the usefulness of a Grid in solving large numerical problems. The architecture that was used in the work is illustrated by Fig. 1.

As a first step, the user invokes a Grid routine with the problem he wants to solve along with the problem parameters. The Grid routine invokes a component called Resource Selector. The Resource Selector accesses the Globus MetaDirectory Service(MDS) to get a list of machines that are alive and then contacts the Network Weather Service(NWS) to get system information for the machines. The Grid routine then invokes a component called Performance Modeler with problem parameters, machines and machine information. The Performance Modeler through an execution model built specifically for the application, determines the final list of machines for application execution. By employing the application specific execution model, GrADS follows the AppLeS approach to scheduling. The problem parameters and the final list of machines are passed as a contract to a component called Contract Developer. The Contract Developer is primitive in that it approves all the contracts that are passed to it. The Grid routine then passes the problem, its parameters and the final list of machines to Application Launcher. The Application Launcher spawns the job on the given machines

Fig. 1. GrADS Architecture for Numerical Libraries



using Globus job management mechanism and also spawns a component called Contract Monitor. The Contract Monitor through an Autopilot mechanism [12] monitors the times taken for different parts of applications and displays the actual and predicted times. Eventually the Contract Monitor will be used for sending information about contract violations to a rescheduler which can in turn take corrective measures on the application execution.

3 Metascheduling in the GrADS architecture

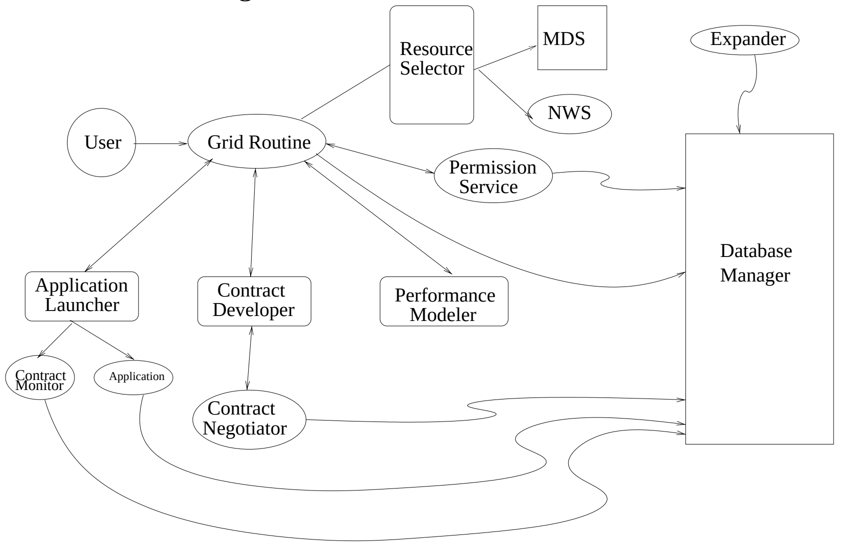
There are a number of potential problems for scheduling with the architecture shown in Fig. 1. First, when two applications are submitted to the Grid at the same time, scheduling decisions will be made for each application assuming the absence of the other application. Second, in the above architecture, if, through the performance model, a new job submitted to the Grid system detects that the Grid resources are not sufficient for it to execute, it cannot make further progress. Similarly, a long running job that was submitted to the system can severely impact the performance of new jobs that enter the system. The root cause of the above and other problems is the absence of a metascheduler that obtains the candidate schedules from different applications and try to balance the needs of different applications. The metascheduler is implemented by the addition of four new components, namely, database manager, permission service, contract negotiator and expander, to the above architecture as illustrated by Fig. 2

The following subsections describe each of these components.

3.1 Database Manager

The database manager maintains a record for each application submitted to the Grid system. The record contains the state of the application, the resource

Fig. 2. Modified GrADS Architecture



information of the Grid resources when the application entered the system, the final list of machines on which the application executes, the predicted time for the application etc. These information are queried by the different components of the metascheduler to make scheduling decisions.

3.2 Permission Service

Permission Service is a daemon that receives requests from the applications to grant them permission to proceed with the usage of the Grid system. The Permission service checks if the Grid resources have adequate capacity to execute the application. If the capacity of the Grid resources is less than the capacity required by the application, the Permission Service either denies permission for the request or tries to accommodate the application by stopping an already executing resource and time consuming application.

3.3 Contract Negotiator

An application contract consists of it problem parameters and the final list of machines on which it will execute. In the GrADS architecture shown in Fig 2, the contract developer, instead of approving the contracts of the applications under all conditions, contacts the Contract Negotiator for obtaining approval of the application contract. The Contract Negotiator acts as a queue manager controlling different applications of the Grid system. The Contract Negotiator either approves the contract in which case the application can proceed to the application launching phase, or rejects the contract in which the case the application restarts from the resource selection phase. The Contract Negotiator rejects the contract under the following conditions:

1. When the application has got its resource information from NWS before an executing application started executing.
2. If the performance of the new application can be improved significantly in the absence of an executing application and the contract monitor either waits for the executing application to complete or pro actively stops the executing application to accommodate the new application.
3. If the already executing applications can be severely impacted by the new application.

1 and 2 have already been implemented while 3 is a work in progress.

3.4 Expander

Expander is a daemon that queries the database manager at regular intervals for completed applications. When an application completes, the expander determines if performance benefits can be obtained for an already executing application by expanding the application to utilize the resources freed by the completed application. If the expander detects such an executing application, it stops the application and continues the application on the new set of resources.

4 Experiments and Results

ScaLAPACK LU factorization code was instrumented such that the time taken for each iteration corresponding to a block of the matrix is measured and monitored. Mechanisms have been implemented in the ScaLAPACK code that will enable the ScaLAPACK application to be stopped and restarted on possibly different number of processors. We use the Internet Backplane Protocol (IBP) [13] for storage of the checkpoint states. IBP depots, where storage can be allocated, are started on the processors of the Grid System.

The GrADS experimental testbed consists of about 40 machines that reside in institutions across the country including University of Tennessee, University of Illinois, University of California at San Diego, Rice University etc. For the easy demonstration of our experimental results, our experimental testbed consists of 4 machines in UIUC called *opus*, 1 machine from University of Tennessee called *torc* and 2 machines from University of Tennessee called *cypher*. Machines from different domains are connected to each other by Internet. Table 1 gives the specification of the machines.

The total execution times reported in the following subsections include the time for Grid overhead and not just the time taken by the actual application. The time for the Grid overhead is reported in our previous work [11].

Table 1. Machine specifications

Machine name	*Processor type*	*speed (MHz)*	*Memory (MByte)*	*Network*
torc	Pentium III	550	512	blabla
cypher	Pentium III	500	512	blabla
opus	Pentium II	450	256	1 Gbit switched Ethernet

4.1 Experiment 1

In this experiment, we demonstrate the functionality of the Permission Service. A large application, app_1, was introduced into the system consisting of the 7 machines mentioned above. Ten minutes after app_1 started, a relatively small application, app_2, that intended to use only the 4 *opus* machines was introduced into the system. app_2 was chosen such that its memory requirements were greater than the memory available in the *opus* system when app_1 was executing. In the following experiment, a problem with matrix size 13000 was chosen for app_1. The Permission Service evaluated the performance benefits of stopping app_1, accommodating app_2, and restarting app_1 after the completion of app_2. The functionality of the Permission Service, when the matrix size of app_2 is 5000, is illustrated on a single *opus* machine in Fig 3. The graph was generated in the NWS web site.

In Fig 4, we observe the percentage performance loss incurred by app_1 due to the accommodation of app_2 in the system. The x-axis represents different matrix sizes for app_2 and the y-axis represents the percentage performance loss incurred by app_1. Two points can be observed from Fig 4. First, for less than 20% of performance loss for app_1, the system was able to accommodate app_2. Without the Permission Service mechanism, app_2 would not have been able to use the system. Second, the performance loss increases with the increase in problem size of app_2. When the problem size of app_2 is comparable with the problem size of app_1, the Permission Service determines that performance benefits cannot be achieved for the system by accommodating app_1. In order to prevent continuous preemption of app_1 by small applications, the scheduling strategy is implemented such that app_1 is ensured to make at least 20% of progress between preemptions.

4.2 Experiment 2

In this experiment, we demonstrate the utility of the contract negotiator in accommodating a new application by stopping an already running application, if significant performance benefits can be obtained for the new application. The stopped application is restarted after the new application completes its execution. For this experiment, only *cypher* machines were used. In this experiment, an application, app_1 is executed on N processors. 3 minutes after app_1 started its execution, an application, app_2 is introduced in the Grid system. app_2 is intended

Fig. 3. Free memory available on a opus machine during the execution of app_1 and app_2

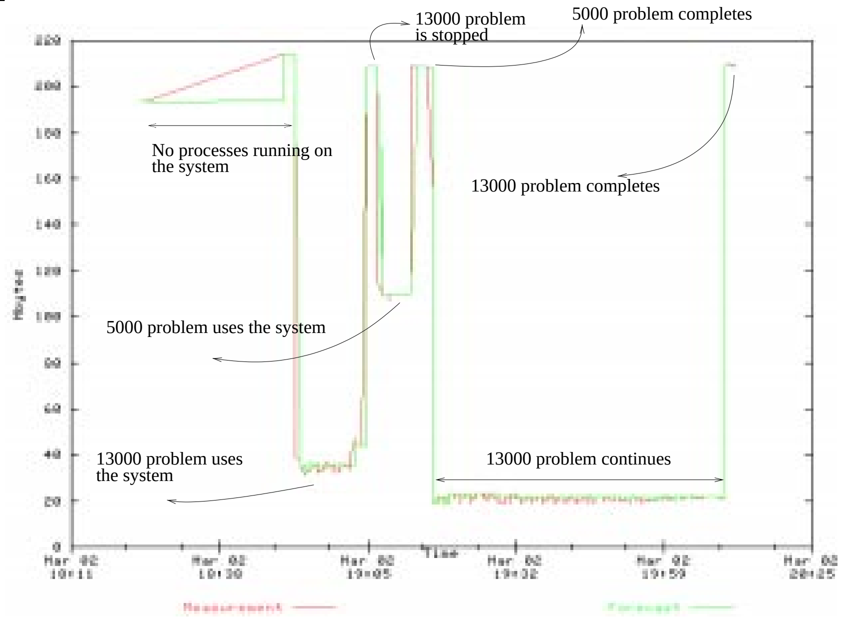
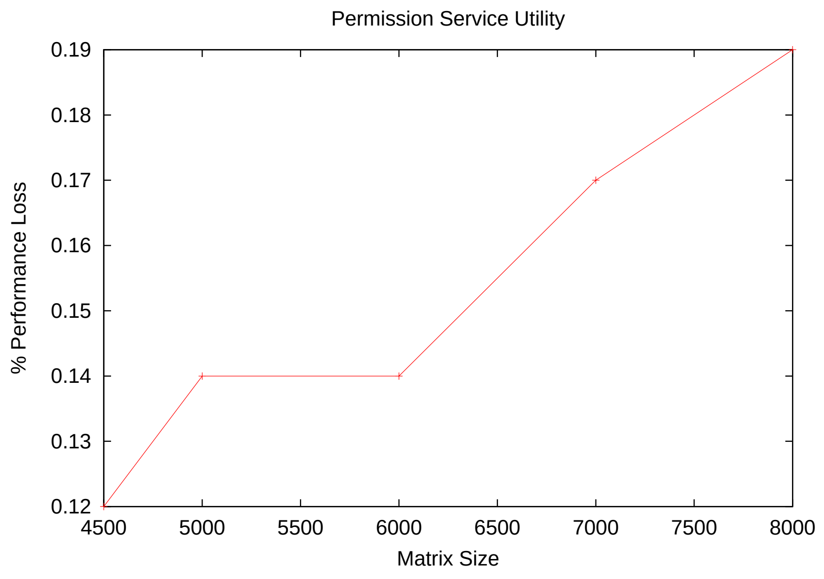


Fig. 4. Performance loss for app_1



to use (N+1) processors. Since N of the processors were occupied by app_1, only a single processor is available for app_2. The Contract Negotiator analyzes the performance benefits that can be obtained by stopping app_1 and making (N+1) processors available for app_2. In the experiments, matrix size 7500 was used for app_2. The total execution time of a 7500 matrix size ScaLAPACK problem when executed on a single processor is 818.11 seconds.

We define

1. Execution time of app_1 without rescheduling, $exec1_{without\_re}$
2. Execution time of app_1 with rescheduling, $exec1_{with\_re}$
3. Execution time of app_2 without rescheduling, $exec2_{without\_re}$
4. Execution time of app_2 with rescheduling, $exec2_{with\_re}$
5. Performance loss for app_1, perf_loss

$$perf\_loss = \frac{exec1_{with\_re} - exec1_{without\_re}}{exec1_{without\_re}}$$

6. Performance gain for app_2, perf_gain

$$perf\_gain = \frac{exec2_{without\_re} - exec2_{with\_re}}{exec2_{without\_re}}$$

7. Utility value, use_val

$$util\_val = \frac{perf\_gain}{perf\_loss}$$

util_val is >1 indicates that the rescheduling strategy is useful for the entire system. util_val <1 indicates that the rescheduling strategy can cause an overall loss in performance for the entire system. Greater the value of util_val, more the usefulness of the rescheduling strategy.

Table 2 shows the matrix sizes of app_1, the number of processors N, the number of processors eventually used by app_2 and the util_val. Note that the eventual number of processors used by app_2 depends on system conditions and execution time model and is not always the (N+1) processors available to app_2.

We observe from Table 2, that the values of util_val are consistently high for the above experiments. This proves that the scheduling strategy of compromising long running jobs for short running jobs is beneficial to the entire system. The value of util_val depends on a number of factors including the times for the long and short jobs and the times for checkpointing the states of the long job. As in the Permission Service, mechanisms have been implemented to avoid continuous preemptions of app_2.

4.3 Experiment 3

In this set of experiments, we illustrate the utility of Expander. An application, app_1, was introduced into the system consisting of 4 *torc* machines. Few minutes after app_1 started, an equally sized application, app_2, that intended to use 8

Table 2. Utility of Contract Negotiator

Matrix Size of app_1	*Processors N*	*Number of Processors used by app_2*	*util_val*
15000	4	5	2.13
17000	5	6	5.11
18500	6	7	2.27
20000	7	8	2.04
21000	8	9	2.05
22500	9	9	2.36
24000	10	9	1.72

machines, 4 *torcs* and 4 *cyphers* was introduced into the system. Since the 4 *torc* machines were occupied by app_1, app_2 was able to utilize only the 4 *cypher* machines. When app_1 completed, the 4 *torc* machines were freed and app_2 was able to utilize the extra resources to reduce its remaining execution time. The Expander evaluated the performance benefits of allowing app_2 to utilize the extra 4 processors.

The experiments for this subsection is in progress and initial results are encouraging.

5 Related Work

There are number of ongoing research efforts in Grid Computing [14], [15], [16], [17], [18], [19]. Although the task allocation policies employed by Condor [16] meet the objectives of our metascheduler, task reallocation takes place when the workstation owner wants to reclaim his resources and not to improve individual application's performance. The objectives of Nimrod-G's [18] scheduling policies are similar to those of our metascheduler where different users' requirements are balanced. Nimrod-G uses grid economies to implement its scheduling policies while our metascheduler uses predicted application time for our scheduling policies. Though the Ninf [19] team had evaluated their scheduler when multiple clients run their jobs, no substantial mechanism has been implemented to guarantee performance for each client.

6 Conclusion

In this paper we have explained the implementation of a metascheduler for the Grid that takes into account both the application level and system level considerations. We have explained in detail, the different components of our metascheduler, viz., the Permission Service, Contract Negotiator and the expander. These components provide valuable scheduling services that play important roles in providing scalability of the Grid system. We have demonstrated the utility of these scheduling decisions with encouraging results.

7 Future Work

Our immediate plans are to demonstrate the usefulness of our metascheduler in a large Grid system when large number of applications execute on the system. Capabilities like evaluating the impact of new applications on existing applications and migration under performance degradation will be added to the metascheduler. After the complete implementation of the metascheduler, the issue of reproducibility of numerical results in the Grid will be investigated.

References

1. T.L. Casavant and J.G. Kuhl, A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, IEEE Transactions on Software Engineering, Vol. SE-14, No. 2, February 1988, pp. 141-154.
2. K.A. Saqabi, S.W. Otto and J. Walpole, Gang Scheduling in Heterogeneous Distributed Systems, Technical Report, OGI, 1994.
3. S. Zhou, X. Zheng, J. Wang, and P. Delisle, Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Software – Practice and Experience, 23(12):1305-1336, December 1993.
4. J. Gehring, A. Reinefeld, MARS - A Framework for Minimizing the Job Execution Time in a Metacomputing Environment, Future Generation Computer Systems, FGCS-12,1 (1996), Elsevier, pp. 87-99.
5. C.A. Waldspurger and W.E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Management, In First Symposium on Operating Systems Design and Implementation (OSDI), pages 1-11. USENIX Association, 1995.
6. J. Weissman, The Interference Paradigm for Network Job Scheduling, Proceedings of the Heterogeneous Computing Workshop, pp. 38-45, April 1996.
7. F. Berman and R. Wolski, The AppLeS Project: A Status Report, Proceedings of the 8th NEC Research Symposium, Berlin, Germany, May 1997.
8. I. Foster and C. Kesselman, eds., The Grid: Blueprint for a New Computing Infrastructure, (San Francisco: Morgan Kaufmann, 1999), ISBN 1-55860-475-8
9. R. Wolski, N. Spring, and J. Hayes, The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Journal of Future Generation Computing Systems, Volume 15, Numbers 5-6, pp. 757-768, October, 1999.
10. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, The GrADS Project: Software Support for High-Level Grid Application Development, International Journal of High Performance Applications and Supercomputing, Vol. 15, number 4 (Winter 2001): 327-344.
11. A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, S. Vadhiyar, Numerical Libraries and The Grid: The Grads Experiments with ScaLAPACK, Journal of High Performance Applications and Supercomputing, Vol. 15, number 4 (Winter 2001): 359-374.
12. R.L. Ribler, J.S. Vetter, H. Simitci, and D.A. Reed, Autopilot: Adaptive Control of Distributed Applications, Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.

13. J. S. Plank, M. Beck, W. R. Elwasif, T. Moore, M. Swany, R. Wolski, The Internet Backplane Protocol: Storage in the Network, NetStore99: The Network Storage Symposium, (Seattle, WA, 1999).
14. I. Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, Intl J. Supercomputer Applications, 11(2):115-128, 1997.
15. A. Grimshaw, W. Wulf, J. French, A. Weaver and P. Reynolds, Jr., Legion: The Next Logical Step Toward a Nationwide Virtual Computer, Tech. Rep. CS-94-21, Department of Computer Science, University of Virginia, 1994.
16. M. Litzkow, M. Livney and M. Mutka, Condor - a Hunter for Idle Workstations, in Proc. 8th Intl. Conf. on Distributed Computing Systems, 1988, pp. 104-111.
17. H. Casanova and J. Dongarra, NetSolve: A Network Server for Solving Computational Science Problems, The International Journal of Supercomputer Applications and High Performance Computing, Volume 11, Number 3, pp 212-223, Fall 1997.
18. R. Buyya, D. Abramson, and J. Giddy, Nimrod-G Resource Broker for Service-Oriented Grid Computing, IEEE Distributed Systems Online, in Volume 2 Number 7, November 2001.
19. S. Sekiguchi, M. Sato, H. Nakada and U. Nagashima, Ninf: Network base information library for globally high performance computing. Parallel Object-Oriented Methods and Applications (POOMA), February 1996.