A Grid-Aware MIP solver: implementation and case studies

Emilio P. Mancini
Università del Sannio,
Dipartimento di Ingegneria,
RCOST, Benevento, Italy
epmancini@unisannio.it

Sonya Marcarelli
Università del Sannio,
Dipartimento di Ingegneria,
RCOST, Benevento, Italy
sonya.marcarelli@unisannio.it

Igor Vasilyev
Institute of System Dynamics
and Control Theory,
SB RAS, Irkutsk, Russia
vil@icc.ru

Umberto Villano
Università del Sannio,
Dipartimento di Ingegneria,
RCOST, Benevento, Italy
villano@unisannio.it

March 18, 2007

Abstract

This paper presents a grid-enabled system for solving large-scale mixed integer programming (MIP) problems. The system has been developed using Globus and MPICH-G2, and consists of two solvers and of an interface portal. After a brief introduction to Branch, Cut and Price optimization algorithms, the paper focuses on the system architecture, solvers and portal user interface. The performance of system is measured and analyzed on a small-scale grid environment consisting of three clusters on a campus LAN.

1 Introduction

Most exact-solution approaches to Mixed Integer Programming (MIP) problems are based on Branch and Bound, which solves optimization problems by implicit enumeration of the solution space, partitioning it into a search tree. Unfortunately, most of the practical MIP problems are NP-hard, and, in the worst case, may require a search tree of exponential size. At least in theory, Branch and Bound has a natural parallel structure. Therefore, the use of a sufficiently high number of processors can make the solution of large-scale problems more practical.

Among the possible implementation methods for Branch and Bound, a powerful technique is Branch, Cut, and Price (BCP). BCP is an implementation of

Branch and Bound in which linear programming is used to derive valid bounds during the exploration of the search tree. Even if the parallelization of BCP is considerably more complex than basic Branch and Bound, currently there are many well known parallel BCP implementations, along with frameworks that allow quick development of customized code [1–8].

While parallel BCP solvers for "traditional" parallel systems are rather customary, the potential of computing grids [9,10] seems to have been only partially exploited at the state of the art [8, 11, 12]. This paper represents a step in this direction, since it describes our experience in the development and performance analysis of a grid-enabled platform for solving large-scale optimization problems.

The developed system is composed of two frameworks, BCP-G and MetaPBC, and of a web portal, SWI-Portal. BCP-G is a customized version of COIN/BCP, an open source framework developed within the IBM COIN-OR project [1]. The original COIN/BCP framework, based on the use of PVM libraries [13], has been provided with a new MPI communication API able to exploit the MPICH-G2 system, a grid-enabled MPI implementation [14, 15]. MetaPBC is instead a brand new framework that we are developing, and it is based on a decentralized master/worker schema [16]. In order to make the system as user-friendly as possible, we have also developed a web portal (SWI-Portal) that manages users and jobs. All of them will be described here. The paper also presents two example solvers that have been developed for testing purposes, a solver for the p-median problem [17, 18], and a generic solver for MIP problems.

In the next section, we introduce the Branch, Cut and Price algorithms and the COIN/BCP framework. The architecture of our grid-enabled system is presented in section 3. Section 4 describes several case studies and the obtained performance results. The paper closes with the conclusions and a discussion on our future work.

2 Branch, Cut and Price algorithms and COIN/BCP framework

Branch and Bound algorithms are the most widely used methods for solving MIP problems. A MIP problem is an optimization problem where the objective function and the constraints are linear, and some of the variables are integer. Branch and Bound is a strategy of exploration of solution space based on implicit enumeration of solutions. Generally, it is an exact method, but it is also possible to stop the search when some prefixed condition is reached, thus obtaining a suboptimal solution. As is well known, Branch and Bound is made up of two phases: a *branching* one, where the set of solutions are partitioned into disjoint subsets, and a *bounding* one, where they are evaluated and the subsets not including the optimal solution are deleted.

Branch and Cut algorithms use a hybrid approach, joining the Branch and Bound method, used to explore the solution space, and the method of cutting planes, used for the bounding phase. The cutting planes method finds the optimal solution introducing a finite number of cuts, that is, inequalities satisfied by all the feasible solutions, but not by the optimal current solution of the problem with relaxed integrality constraints (LP relaxation) [19].

Branch and Price algorithms are instead based on column generation [20]. This method is used to solve problems with a very large number of variables. It uses initially only a small subset of the problem variables and of the respective columns in the constraints matrix, thus defining a reduced problem. In fact, in the original problem, there are too many columns and great part of them will have the respective variables equal to zero in an optimal solution.

Branch, Cut and Price joins the two methods used by Branch and Cut and Branch and Price, producing dynamically both cutting planes and variables [1].

2.1 COIN/BCP

COIN/BCP is an open-source framework that implements the Branch, Cut and Price algorithms for solving MIP problems [21]. It offers a parallel implementation of the algorithm using the message-passing library PVM (Parallel Virtual Machine) [13]. Its functions are grouped in four independent computational modules:

- Tree Manager (TM), which is the master process. It is responsible for the entire search process, starts new processes and checks their status, sends the problems to the slave processes and stores the best solutions. Finally, it recognizes the end of the search, stopping all processes.

- Linear Programming (LP), which is a slave process. It performs the most complex computational work, since it is responsible for the branching and bounding phases. It uses a sequential solver to solve the LP relaxation.

- Cut Generator (CG), a slave process that creates globally-valid inequalities not satisfied by the current solution of LP relaxation, sending them to the LP that requested them.

- Variable Generator (VG), which performs the column generation. It creates variables with reduced costs, and sends them to the requested LP.

COIN/BCP implements a Branch, Cut and Price single-node pool algorithm, where there is a single central list of candidate sub-problems to be processed, owned by the tree manager. The modules communicate with each other by exchanging messages through a message-passing protocol defined in a separate communications API. The standard version of the framework exploits the PVM run-time system. The first phase of our work was to implement a new parallel interface based on MPI, in order to make it possible to use the framework in a grid environment using the Globus Toolkit and MPICH-G2 [14,15].

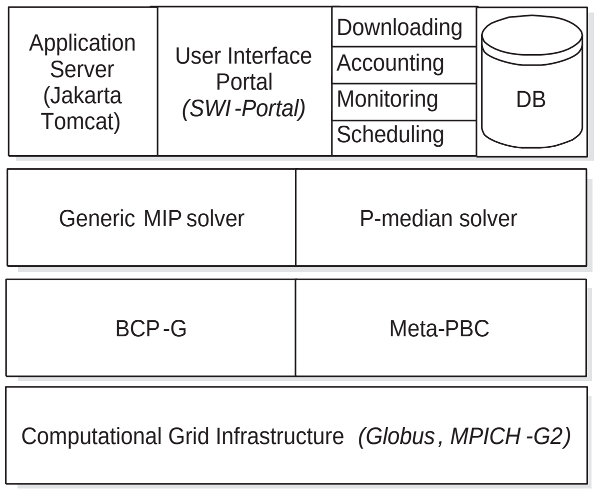

Application Server (Jakarta Tomcat)
User Interface Portal (SWI-Portal)
Downloading
Accounting
Monitoring
Scheduling
DB
Generic MIP solver
P-median solver
BCP-G
Meta-PBC
Computational Grid Infrastructure (Globus, MPICH-G2)

Figure 1: System architecture

3 System description

The architecture of the grid-enabled platform developed is shown in Fig. 1. In the figure, the upper layer is the portal interface, in the middle there are the two solvers *p-Median solver* and *MIP solver*, which are implemented on the base of BCP-G and Meta-PBC frameworks. All of which rely on the lower layer (the Globus and MPICH-G2 frameworks).

3.1 BCP-G

BCP-G is a framework that we have implemented extending COIN/BCP. As mentioned before, this required the development of a new communication interface written in MPI. Our MPI communication interface is implemented by the two classes `BCP_mpi_environment` and `BCP_mpi_id`, which manage the communications between computational modules and the process ids, respectively. In particular, we have added to the old system new functions, to initialize the MPI environment and to determine the number of processes started by `mpirun`. The MPI interface differs from the PVM one, since it includes no spawn functionality to start dynamically new processes. If the number of started processes is not equal to the number of processes requested by the user, an exception is generated. This new interface, which is now integrated in the COIN-OR framework [21], allows the use of this framework in a Globus grid environment through the grid-enabled implementation of MPI, MPICH-G2. The user has simply to write a Globus rsl script and, through the `globusrun` command, he can launch the solver execution [22].

3.2 Meta-PBC

Meta-PBC is a parallel framework for solving MIP problems based on the Branch and Cut algorithm. It is not the porting of existing software, but it has been developed from scratch for an on-going research project. Our idea in designing this library was to create a parallel implementation, which could take advantage of the best sequential B&C solvers, such as the commercial solvers ILOG CPLEX or Dash Optimization Xpress-MP. In Meta-PBC, these sequential solvers are therefore executed on a purposely-developed parallel layer, which manages their workload. Meta-PBC consists of three modules: *manager*, *worker* and *tree monitor* [16]. The *manager* is the master process. It is responsible for the initialization of the problem, the I/O and manages the message handling between the workers. The *worker* is a sequential solver of Branch and Cut with some additional functionality to communicate in the parallel layer. The workers communicate with each other through the parallel API to know the state of the overall solution process. The *tree monitor* collects and visualizes information about the search tree. The parallel interaction between modules is achieved by a separate communication API in a way similar to COIN/BCP. In particular, an abstract message environment is used, which can be implemented on the top of any communication protocol supporting basic message passing functions. The current version is implemented in MPI. The processes can thus be executed on the Grid with MPICH-G2.

3.3 SWI-Portal

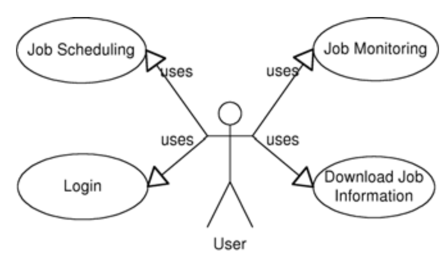
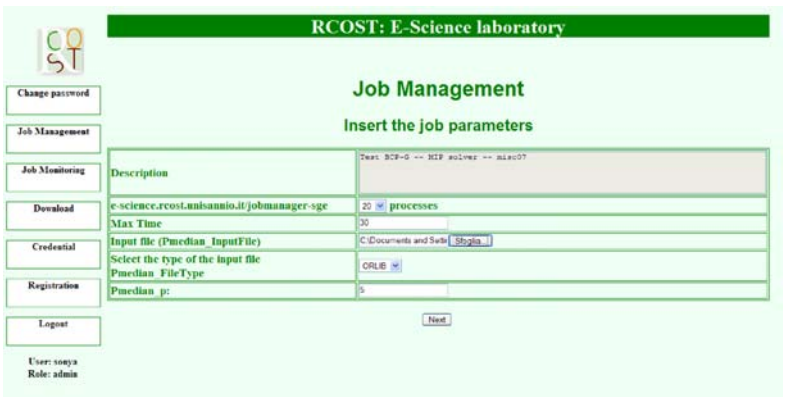


Figure 2: Use case diagram of the SWI-Portal.

The SWI-Portal (Solver Web Interface-Portal) is the interface to our system. Users interact with the portal, and hence with the solvers and the grid, through this interface. This allows them to submit a new job and to solve a problem, to monitor their job, to view their output and to download the results. SWI-Portal is implemented using the Java Server Pages technology (JSP). It consists of an user interface and of a set of Java classes, wrapping of the most important and useful Globus functions. Furthermore, it interacts with a database collecting information on users, job and resources.

SWI-Portal is composed of four subsystems, which correspond to the four use cases in Fig. 2. The account subsystem is responsible for managing user



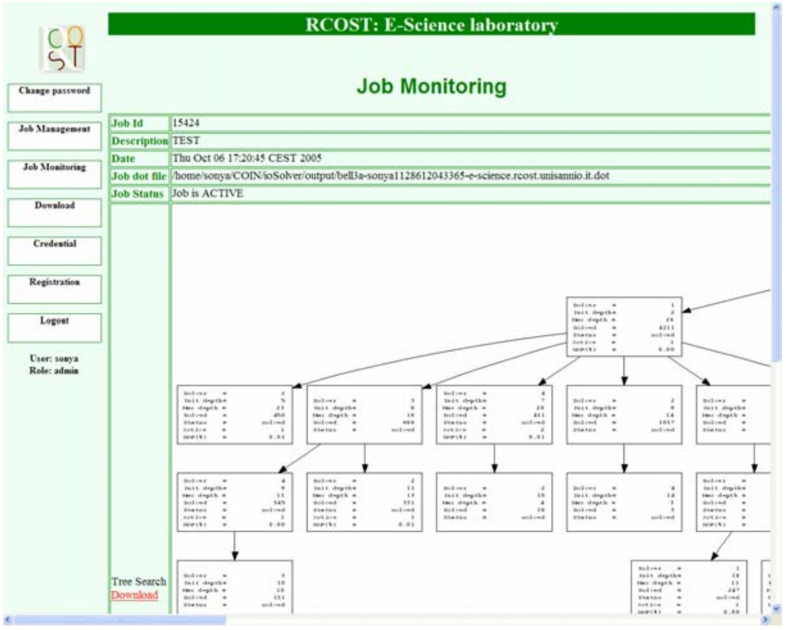
RCOST: E-Science laboratory

Job Management

Insert the job parameters

Description	Test BCP-G -- MIP solver -- misc07
e-science.rcost.unisannio.it/jobmanager-sge	20 processes
Max Time	30
Input file (Pmedian_InputFile)	C:\Documents and Setti
Select the type of the input file Pmedian_FileType	ORLIB
Pmedian_p:	5

(a) Scheduling subsystem



RCOST: E-Science laboratory

Job Monitoring

Job Id	15424
Description	TEST
Date	Thu Oct 06 17:20:45 CEST 2005
Job dot file	/home/sonya/COIN/ioSolver/output/bell3a-sonya1128612043365-e-science.rcost.unisannio.it.dot
Job Status	Job is ACTIVE

(b) Monitoring subsystem

Figure 3: Screenshots of the SWI-Portal.

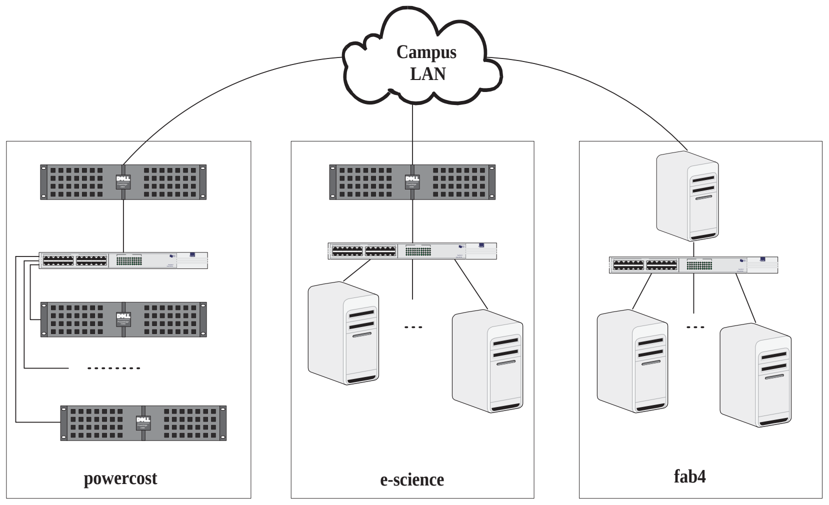


Figure 4: Grid environment

access in conjunction with the users DB. This subsystem allows a user to register into the system, and to enter in the portal giving his login and password.

The scheduling subsystem (Fig. 3(a)) currently supports explicit scheduling; the user has to specify the hosts on which he wishes to run his jobs. He must insert a set of parameters describing his problem, and the scheduling system invokes the Globus system to start the run. The subsystem also records information about the runs in the database. It creates automatically the parameter file necessary for the solver, using the information supplied by the user, and creates a Globus rsl script describing the running job. The grid layer is responsible for the transfer of the files to all the hosts selected to execute the job.

From the pages of the monitoring subsystem (Fig. 3(b)), the user can check the status of the search, and consult any other information about all the started processes (such as output, error, rsl, and search tree). Users can download through the Download Subsystem all information regarding their jobs and/or cancel them from the server.

4 Computational experience and performance analysis

4.1 Grid environment

We have configured an experimental grid environment made up of three Rocks [23] clusters (*fab4*, *e-science* and *powercost*) at three different sites of the University of Sannio. Fig. 4 shows the architecture of our grid environment. The clusters have a front-end with public IP and compute nodes with hidden IPs.

They provide 2.8 GHz Xeon processors, 1 GB RAM (four on *fab4* and sixteen on *e-science*, respectively) and 3.0 GHz Xeon processors, 2 GB RAM (44 on *powercost*), as in Table 1. The intra-cluster connection is always GigaEthernet. The three clusters are on the same campus LAN, and are currently connected by a very slow connection, whose performance is discussed below. We installed on each front-end the Globus Toolkit and the local Sun Grid Engine (SGE) scheduler. The Sun Grid Engine is a distributed resource management (DRM) software that provides functions such as job submission, monitoring and management, to utilize effectively the resources within the cluster. In particular, the Globus gatekeeper uses the SGE scheduler as its job-manager.

	powercost	**e-science**	**fab4**
Number of nodes	44	16	4
Processors type	Xeon 3.0 MHz	Xeon 2.8 MHz	Xeon 2.8 MHz
Memory	2 GB	1 GB	1 GB

Table 1: Cluster system structure.

On each front-end, we installed also MPICH-G2, which allows inter-cluster and intra-cluster communications. MPICH-G2, based on Globus Toolkit services, allows to run MPI applications on a grid environment. It uses TCP for inter-machine messaging and a vendor-supplied MPI (where available) for intra-machine messaging. MPICH-G2 requires point-to-point communication between the nodes where the jobs are running. Unfortunately, this requires that all compute nodes have public IP addresses, but this is in contrast with a classical cluster configuration like ours, where the compute nodes have private IPs.

In order to solve this problem, and to use all the processors available, we have chosen a solution based on the Realm Specific IP (RSIP) framework and protocol [24]. RSIP is a network address translation technology that performs a function similar to NAT. It allows the communication between two hosts belonging to different address spaces. In our solution, we installed on each front-end an RSIP server and on each compute node an RSIP client. For instance, when a compute node of *e-science* (RSIP client) wishes to contact a node of *fab4*, it queries the RSIP server for a port number and a public IP address. The client then tunnels the packets to the RSIP server, which strips off the tunnel headers and sends the packets to the target node. On incoming packets, the RSIP server looks up the client IP, based on port number, adds the tunnel header and sends them to the RSIP client.

In order to evaluate the detrimental effects on performance of the slow inter-cluster LAN connection and of the overhead introduced by the RSIP protocol, we measured the bandwidth and the transmission latency of the cluster connections through the Intel MPI Benchmarks suite (IMB). This suite, also known as Pallas MPI Benchmarks [25], provides a set of benchmark tests measuring the most important MPI functions. In particular, we used the *PingPong* benchmark,

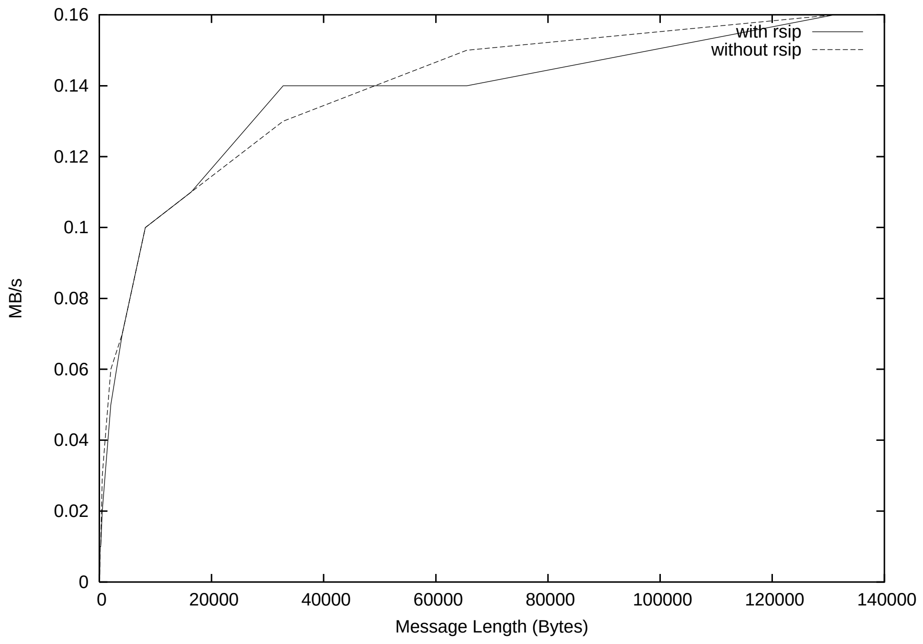


Figure 5: Intercluster bandwidth with and without RSIP

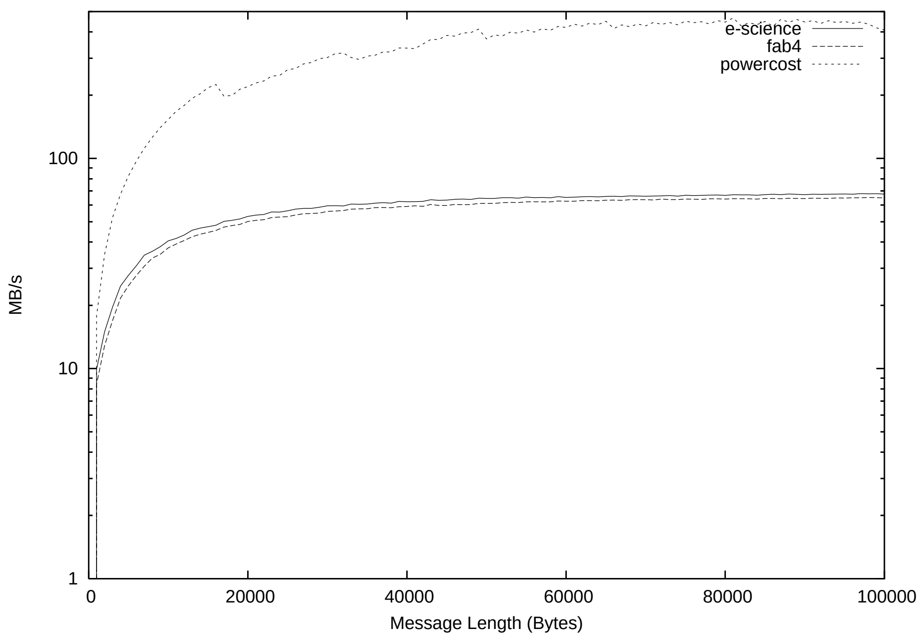


Figure 6: Intracluster bandwidth

which performs point-to-point communications, to measure the bandwidth of the intercluster connection with and without RSIP (Fig. 5), and of the intra-cluster interconnect on *e-science*, *fab4* and *powercost* (Fig. 6). Table 2 shows the transmission latency measured in all the previously-mentioned conditions.

	Latency (μs)
inter-cluster with RSIP	1051423
inter-cluster without RSIP	971584
powercost	595
e-science	2067
fab4	1994

Table 2: Transmission latency

4.2 Case studies

BCP-G and Meta-PBC are general-purpose frameworks for solving MIP problems. To use them, it is necessary to implement some problem-specific functions (see again Fig. 1). On the base of BCP-G, we have implemented a *p-median solver*. The p-median problem is an NP-hard problem widely addressed in the literature (see [18] for a survey). To solve this problem, we implemented a Branch, Cut and Price algorithm with a simple procedure to choose the core problem, a logical reduction test to fix some variables, a column and row generation algorithm to solve the LP relaxed problem and cutting planes. The details of this approach are given in [17,18,26]. Moreover, we have implemented generic MIP solvers based on both frameworks (BCP-G and Meta-PBC) to solve MIP problems in general form. In the following we will consider only the MIP solver based on BCP-G, as Meta-PBC framework development is still in progress. Some preliminary results on Meta-PBC can be found in [16].

In the computational experiments with the MIP solver, we used instances from the MIPLIB library [27], which, since its introduction, has become a standard test set and is commonly used to compare the performance of mixed integer optimizers. In particular, we measured the performance of our grid solver on two instances from MIPLIB, *misc07* and *stein45*.

Measuring and analyzing the performance of a grid-enabled code is an interesting matter, since the resulting figures are, at least in principle, widely affected by the problem, by the code structure and communication patterns exploited, by the grid hardware used, by the middleware and run-time systems coming into play, and, above all, by the communication performance inside the clusters making up the grid and among them. The results are inevitably configuration-specific, but, in our opinion, can be used by developers tackling similar problems on similar hardware for preliminary rule-of-thumb considerations.

The hardware configuration used for our tests is undoubtedly much less powerful than state-of-art scientific grids. However, it is representative of computing

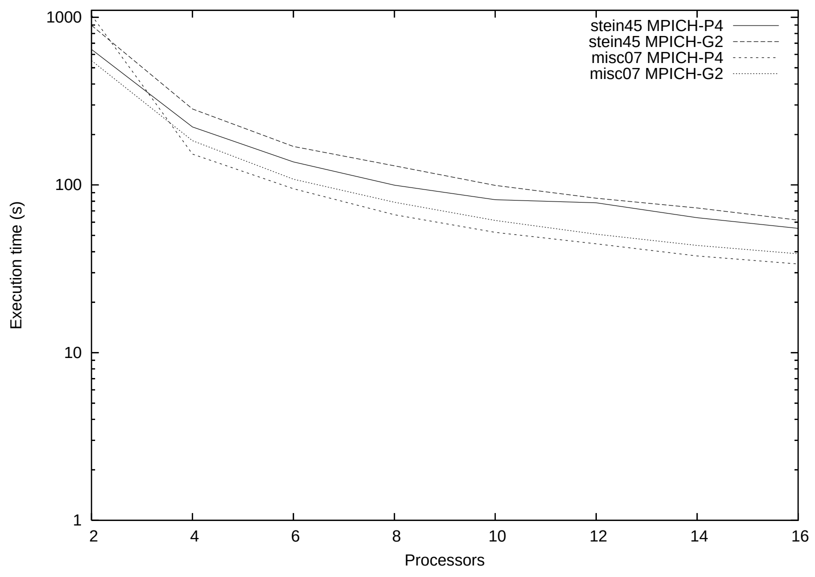


Figure 7: Comparison of MPICH-P4 and MPICH-G2 response times for the *stein45* and *misc07* problems of the MIPLIB Library on a single cluster

hardware and networks widely available in academic and research environments. In these environments, the natural question that arises is if it is convenient to use jointly clusters of different speed and size, even if they are connected by slow networks. The obvious alternative is to use only one cluster (possibly the most powerful of them all) and to make rid of the overheads due to the grid infrastructure. Furthermore, assumed that the use of a small grid is convenient, how can be the tasks allocated in order to maximize performance? The following performance figures were selected among the results of our experimentations in order to provide concise responses to the above problems.

The first issue considered is to detect possible performance losses due to the use of a grid environment. We have compared the performance of our solver in a *single cluster* using both MPICH-G2, suitable for the use in grids, and MPICH-P4, the traditional MPI implementation for clusters based on the p4 communication library [28, 29]. The results of our tests (Figure 7) clearly show that MPICH-G2 introduces a negligible performance penalty in a single cluster.

However, the previous result is not sufficient to deduce that the use of a grid environment, particularly on a slow and/or loaded network, is always satisfactory as far as performance figures are concerned. We have measured the completion time for the two problems (*misc07*, *stein45*) for increasing number of tasks using two different scheduling policies. In the first case, as the degree of parallelism rises the tasks have been allocated first inside *powercost*, the largest cluster, and then outside, on the two small clusters *e-science* and *fab4*. In the second case, instead, the tasks have been allocated first on the small clusters,

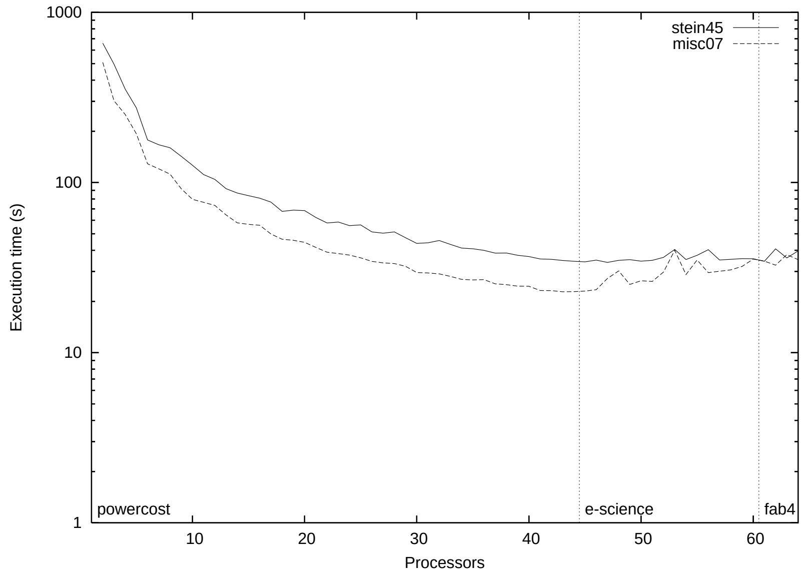


Figure 8: Response times for the *stein45* and *misc07* problems on a variable number of processors

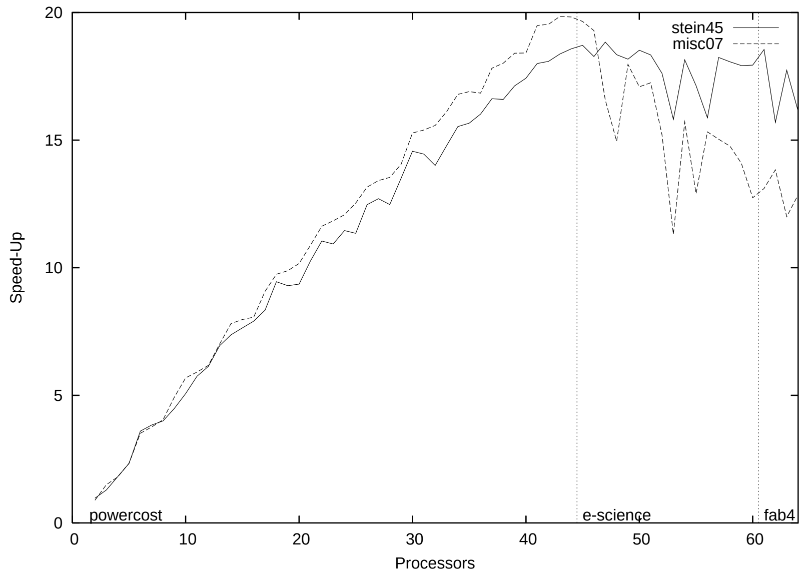


Figure 9: Speedup for the *stein45* and *misc07* problems

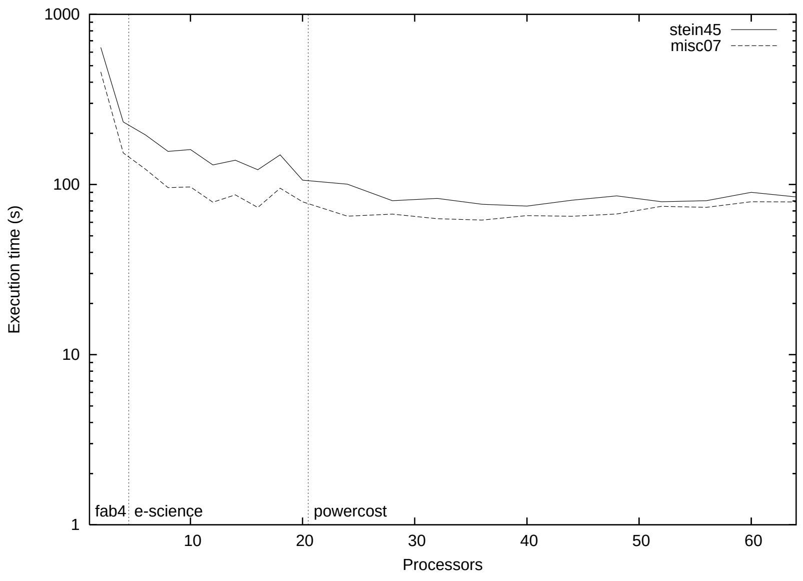


Figure 10: Response times for the *stein45* and *misc07* problems using an alternate scheduling policy

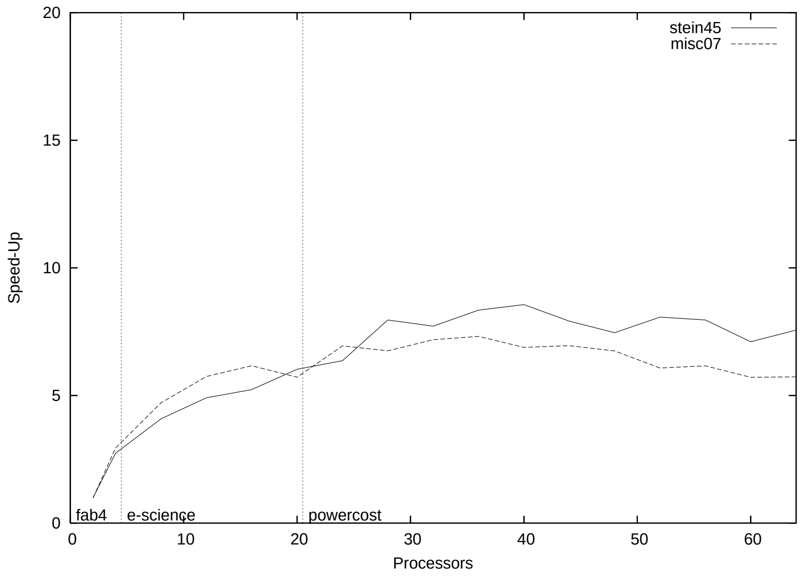


Figure 11: Speedup for the *stein45* and *misc07* problems using an alternate scheduling policy

and then on the large one. The response times and the measured speedups are presented in Figures 8-10 and 9-11, respectively. For different input data, the performance behavior has turned out to be very similar. These additional results are not reported here for brevity's sake.

The apparently "strange" result obtained is that in both cases, from about forty processors onwards, the use of additional processors does not involve any significant gain in performance. As this is independent of the allocation policy, it is inevitably due to the centralized approach used by our solver. As parallelism degree increases, the master process becomes a bottleneck and the use of further processors is insignificant, or even detrimental. However, it should also be noted (please see Figg. 8 and 10 for a comparison) that the two scheduling policies do not lead to the same performance results, as the use of the larger cluster first allows to obtain much lower response times.

The communication pattern adopted by the solver throughout the computation is master/slave. All data are transferred between master and slaves (in the two directions) through point-to-point communications, using standard send and receive. For the two problems that are the object of our tests, the amount of the data globally exchanged by the master during one run increases with the number of slaves, from a minimum of about 20 MB (1 slave) to a maximum of about 30 MB (63 slaves), and it is equally shared among the slaves. Communication time unbalance is only due to possible differences in communication channels. In our tests, we have found that the slaves on the slow network spend about 80% of their total execution time in communication. This figure has to be compared to the 5% of total execution time spent by the slaves on "fast" networks.

The conclusions that can be drawn from our tests are:

- the availability of parallel machines should not lead to the misplaced belief that every parallel code lends itself well to its use on hundreds of processors or more. Parallelization strategies that are successful for a reduced number of processors (e.g., master/slave) often turn out to be inadequate even on medium-sized parallel machines.

- the possibility to use several parallel machines connected in a grid does not come for free. Grid management and data transport introduce overheads, but they typically are fairly limited. However, their impact on performance has to be properly evaluated.

- the best option to execute low-parallelism applications is obviously the use of a single large cluster. When this is not possible, the use of small machines arranged in a grid is not so bad, even on a slow network, as documented by the speedups shown in the two sections of Fig. 11 on the left.

5 Related work

In the last years, many software packages implementing parallel branch and bound have been developed. SYMPHONY [1] is a parallel framework, similar to COIN/BCP, for solving mixed integer linear programs. COIN/BCP and SYMPHONY are combined in a new solver under development, ALPS [2]. Some other parallel solver are PUBB [3], PPBB-Lib [4] and PICO [5]. PARINO [6] and FATCOP [7, 8] are generic parallel MIP solvers, and the second one is designed for grid systems.

The grid portal allows users to access grid facilities by a standard Web Browser. A grid portal offers an user interface to the functionalities of a grid system. Currently there are many grid portals, including the Alliance Portal [30], the NEESgrid Portal [31], the Genius Portal [32] and the IeSE Portal [33]. Also, there are many grid portal development frameworks including the Gridsphere Portal [34], the GridPort Portal [35], the Grid Portal Development Toolkit (GPDK) [36] and the OCGE Portal Toolkit [37].

The literature dealing with the management and the performance of MPI applications in grid environments made up of private IP clusters is relatively limited. Park et al. [38] present MPICH-GP, an extension of MPICH-G2 for supporting Private IP, whereas Velusamy et al. [39] describe a solution based on IMPI standard with Network Address Translation mechanism and Das et al [24] propose a solution based on RSIP. Papers on similar topics are written by Heymann et al. [40] and Choi et al. [41]. Chen and Shmidt [42] present a performance analysis on hierarchical grid system with different bandwidths between clusters.

6 Conclusions and future work

In this paper, we have presented a grid-enabled system made up of two solvers and a portal interface for solving large-scale optimization problems. Moreover, we have described the configuration of our experimental grid environment, which is made up of three clusters located in a campus LAN. We have discussed the problems encountered using MPICH-G2 in such environment, where the compute nodes have hidden IPs, showing the solution based on RSIP and its performance evaluation. The obtained results show that our system can be successfully used in low-medium parallelism environments (tens of processors), whether made up of a single cluster or of multiple small systems arranged in a grid.

In our future work, we wish to change the architecture of the solver system and to adopt a decentralized approach. The idea is to divide the search tree in many sub-trees and assign one of them to each cluster, which will solve it individually. In this way, the degree of parallelism that could be successfully exploited in a grid should make the system fit also for use in medium-high parallelism computing environments.

As regards the SWI-Portal, we will extend its functionalities adding a system for monitoring available resources and implicit scheduling. Implicit scheduling will make the system capable of choosing automatically the best set of hosts where to submit the user's job.

References

[1] T.K. Ralphs, L. Ladanyi, and M.J. Saltzman. Parallel Branch, Cut, and Price for Large-Scale Discrete Optimization. *Mathematical Programming*, 98:253–280, Sep 2003.

[2] T. K. Ralphs, L. Ladanyi, and M. J. Saltzman. A library hierarchy for implementing scalable parallel search algorithms. *Journal of Supercomputing*, 28(2):215–234, 2004.

[3] Y. Shinano, M. Higaki, and R. Hirabayashi. Control schemas in a generalized utility for parallel branch and bound. In *Proc. of the 1997 Eleventh International Parallel Processing Symposium*, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[4] S. Tschoke and T. Polzer. *Portable Parallel Branch-And-Bound Library PPBB-Lib User Manual.* Department of computer science Univ. of Paderborn, Nov 1996.

[5] J. Eckstein, C.A. Phillips, and W.E. Hart. Pico: An object-oriented framework for parallel branch and bound. Technical report, Rutgers University, Piscataway, NJ, Aug 2000.

[6] J. Linderoth. *Topics in Parallel Integer Optimization.* PhD thesis, School of Industrial and Systems Engineering, Georgia Inst. of Tech., Atlanta, GA, 1998.

[7] Q. Chen and M. C. Ferris. Fatcop: A fault tolerant condor-pvm mixed integer programming solver. Technical report, University of Wisconsin CS Department Technical Report 99-05, Madison, WI, 1999.

[8] Q. Chen, M.C. Ferris, and J.T. Linderoth. Fatcop 2.0: Advanced features in an opportunistic mixed integer programming solver. *Annals of Op. Res.*, (103):17–32, 2001.

[9] M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. *Software: Practice and Experience Journal*, 32:1437–1466, Nov 2002.

[10] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid. Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 2001.

[11] Kento Aida and Tomotaka Osumi. A case study in running a parallel branch and bound application on the grid. In *Proc. of the The 2005 Symposium on Applications and the Internet (SAINT'05)*, pages 164–173, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Lucia M.A. Drummond, Eduardo Uchoa, Alexandre D. Goncalves, Juliana M.N. Silva, Marcelo C.P. Santos, and Maria C. S. de Castro. A grid-enabled distributed branch-and-bound algorithm with application on the steiner problem in graph. Technical report, Universidade Federal Fluminense, Instituto de Computacao, Dec 2004. `http://www.ic.uff.br/PosGrad/RelatTec/Download/rt_02-05.pdf.gz`.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderan. *PVM 3 User's Guide and Reference Manual*, Sep 1994.

[14] L. Ferreira, B. Jacob, S. Slevin, M. Brown, S. Sundararajan, J. Lepesant, and J. Bank. *Globus Toolkit 3.0 Quick Start.* IBM, Aug 2003.

[15] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Dist. Comp.*, 63:551–563, May 2003.

[16] I. Vasil'ev and P. Avella. Computational experiments with pbc: Parallel branch and cut library. In *Proc. of XIII Conference "Optimization methods and its applications"*, pages 293–300, Irkutsk, Russia, Jun 2005.

[17] P. Avella and A. Sassano. On the p-median polytope. *Mathematical Programming*, (89):395–411, 2001.

[18] P. Avella, A. Sassano, and I. Vasil'ev. Computational study of large-scale p-median problems. *Mathematical Programming*, (available online), 2006. `http://www.optimization-online.org/DB_HTML/2003/03/625.html`.

[19] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, 1999.

[20] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[21] T.K. Ralphs and L. Ladanyi. *COIN/BCP User's Manual*, Jan 2001. `http://www.coin-or.org/Presentations/bcp-man.pdf`.

[22] Globus Alliance. *WS GRAM: Developer's Guide*, 2005. `http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/developer`.

[23] Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno. NPACI Rocks: tools and techniques for easily deploying manageable Linux clusters. *Concurrency and Computation: Practice and Experience*, 15:707–728, Apr 2003.

[24] D. Das, R. Sabharwal, S. Saraswati, P. N. Anantharaman, and J. Oh. A network architecture for enabling execution of mpi applications on the grid. *International Journal of Information Technology*, 11(4):74–83, 2004.

[25] Intel MPI benchmarks (formally known as Pallas MPI Benchmarks), 2006. `http://www.intel.com/cd/software/products/asmo-na/eng/cluster/clustertoolkit/307696.htm`.

[26] P. Avella and A. Sforza. Logical reduction tests for the p-median problem. *Annals of Operations Research*, (86):105–115, 1999.

[27] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library MIPLIB 3.0. *Optima*, (58):12–15, 1998.

[28] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[29] William D. Gropp and Ewing Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI.* Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[30] Alliance Portal Project. *Scientific Portals.* Argonne National Labs, Nov 2002. `http://www.extreme.indiana.edu/alliance/docandpres/SC2002PortalTalk.pdf`.

[31] George E. Brown. *Towards a Vision for the NEES Collaboratory.* NEES Consortium Development Project, Oct 2002. `http://www.curee.org/projects/NEES/docs/outreach/VisionWhitePaperV3.pdf`.

[32] R. Barbera, A. Falzone, and A. Rodolico. The genius grid portal. In *Proc. of Computing in High Energy and Nuclear Physics*, pages 24–28, La Jolla, California, Mar 2003. `https://genius.ct.infn.it`.

[33] K. Kleese van Dam, S. Sufi, G. Drinkwater, L. Blanshard, A. Manandhar, R. Tyer, R. Allan, K. O'Neill, M. Doherty, M. Williams, A. Woolf, and L. Sastry. An integrated e-science environment for environmental science. In *Proc. of Tenth ECMWF Workshop*, pages 175–188, Reading, England, 2002.

[34] Jason Novotny, Michael Russell, and Oliver Wehrens:. Gridsphere: An advanced portal framework. In *Proc. of 30th EUROMICRO Conf.*, pages 412–419, Rennes, Fr., Aug 2004. IEEE. `http://www.gridsphere.org/gridsphere/wp-4/Documents/France/gridsphere.pdf`.

[35] M. Thomas, S. Mock, J. Boisseau, M. Dahan, K. Mueller, and D. Sutton. The gridport toolkit architecture for building grid portals. In *Proc. of the 10th IEEE International Symp. on High Perf. Dist. Comp.*, Aug 2001. `http://gridport.net`.

[36] Jason Novotny. The grid portal development kit. *Grid Computing*, pages 657–673, May 2003. `http://doesciencegrid.org/projects/GPDK`.

[37] M P Thomas, J Burruss, L Cinquini, G Fox, D Gannon, L Gilbert, G von Laszewski, K Jackson, D Middleton, R Moore, M Pierce, B Plale, A Rajasekar, R Regno, E Roberts, D Schissel, A Seth and W Schroeder. Grid portal architectures for scientific applications. In *Scientific Discovery Through Advanced Computing (SciDAC 2005)*, pages 596-600, San Francisco, CA, USA, Jun 2005. Journal of Physics Volume 16, 2005.

[38] Kumrye Park, Sungyong Park, Ohyoung Kwon, and Hyoungwoo Park. MPICH-GP: A private-ip-enabled mpi over grid environments. In *Proc. of Second International Symposium on Parallel and Distributed Processing and Applications (ISPA04)*, pages 469–473, Hong Kong, China, Dec 2004.

[39] V. Velusamy, P. Bangalore, and P. Raman. Communication strategies for private-ip-enabled interoperable message passing across grid environments. In *Proc. of First International Workshop on Networks for Grid Applications*, San Jose, CA, Oct 2004. `http://www.broadnets.org/2004/workshop-papers/Gridnets/Velusamy_V.pdf`.

[40] Elisa Heymann, Miquel A. Senar, Enol Fernández, Alvaro Fernández, and José Salt. Managing mpi applications in grid environments. In Marios D. Dikaiakos, editor, *Grid Computing: Second European AcrossGrids Conference, Lecture Notes in Computer Science*, volume 3165, pages 42–50, Jan 2004.

[41] Siyoul Choi, Kumrye Park, Saeyoung Han, Sungyong Park, Ohyoung Kwon, Yoonhee Kim, and Hyoungwoo Park. An nat-based communication relay scheme for private-ip-enabled mpi over grid environments. In *Proc. of International Conference on Computational Science (ICCS04)*, pages 499–502, Krakóv, Polland, 2004.

[42] Chunxi Chen and Bertil Schmidt. Performance analysis of computational biology applications on hierarchical grid systems. In *Proc. of 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004)*, pages 426–433, Chicago, Illinois, USA, Apr 2004. IEEE Computer Society.