Towards an Intelligent GRID Scheduling System

Ralf Gruber[1], *Vincent Keller*[1], *Pierre Kuonen*[5], *Marie-Christine Sawley*[4], *Basile Schaeli*[2], *Ali Tolou*[1], *Marc Torruella*[6], *Trach-Minh Tran*[3]

[1]LIN-STI, Ecole Polytechnique Fédérale, CH-1015 Lausanne, Switzerland
[2]LSP-I&C, Ecole Polytechnique Fédérale, CH-1015 Lausanne, Switzerland
[3]CRPP, Ecole Polytechnique Fédérale, CH-1015 Lausanne, Switzerland
[4]CSCS, CH-6928 Manno, Switzerland
[5]Ecole d'Ingénieurs et d'Architectes, CH-1705 Fribourg, Switzerland
[6]Universitat Politecnica Catalunya, E-08034 Barcelona, Espana

Abstract

The main objective of the Intelligent GRID Scheduling System (ISS) project is to provide a middleware infrastructure allowing a good positioning and scheduling of real life applications in a computational GRID. According to data collected on the machines in the GRID, on the behaviour of the applications, and on the performance requirements demanded by the user, a heuristic cost function is evaluated by means of which a well suited computational resource is detected and allocated to execute his application. The monitoring information collected during execution is put into a database and reused for the next resource allocation decision. In addition to providing scheduling information, the collected data allows to detect overloaded resources and to pin-point inefficient applications that could be further optimised.

1 Introduction

The development of GRID technology for computational purposes is promising. By harnessing a great number of different systems in a transparent manner, a user can have access to the computer architectures that are well suited to the constraints of his applications.

The different communication needs of applications demand a GRID that can offer different parallel computer architectures: SMP and/or NUMA machines for shared memory parallel applications, a NoW (Network of Workstations) interconnected by a bus for embarassingly parallel applications, scalable but cost-effective networked clusters for applications dominated by point-to-point communications, and more expensive machines with faster networks for communication intensive applications.

There is currently little feedback about applications that are not adapted to the hardware infrastructure, and little incentive to do so: if for instance a user notices that the network is too slow and hampers the performance of its application, he may try to find another machine to run it. On the

Proc. of 6th Int. Conf. PPAM 2005, Poznan, Poland, Lecture Notes in Computer Science 3911 (Springer, 2006) 751-757

other hand, when he runs an embarassingly parallel application on a costly NUMA machine, he will probably not recognise this as a problem. In the future, one would like to choose a well suited hardware for the application (according to peak performance, memory bandwidth of the processor, or inter-node network communication system), and this in a most automatic manner.

The ISS project is precisely aimed at solving this latter problem. The ISS middleware will be built on top of existing GRID middleware infrastructures such as Globus [1], Unicore [2], EGEE [7], or GridLab [3]. In a first phase, a static parameterisation of the test applications is used to decide on which machine of a GRID testbed the application should be submitted. In a second phase, the information on the behaviour of the application during execution is collected and put into a database. Scheduling decisions are then made in an automatic manner, taking constantly the monitored data into account. In a third phase, the accumulated data will be interpreted statistically to recognise overloaded resources that have to be complemented.

The long term goal is to determine the suitability of a platform using a more general cost function, which would not be restricted to computation costs. Other indirect costs could include for instance the waiting time of an engineer, or the licence of a commercial application.

Within this paper, we present ideas on how to parameterise the GRID hardware and the parallel applications [4], and how to use these parameters to decide on which machine a given application is to be executed. Moreover, a first statistical study on the CPU usage of the Pleiades.epfl.ch cluster is presented together with two application profiles coming from CFD and plasma physics. Such measurements will be used to automatically parameterise the applications in the next phase of the project.

2 Parameterisation of clusters and applications

2.1 Different types of machines in a GRID

Let us consider a cluster with P computational nodes, each node has a processor peak performance of R_∞ [Gflops/s], and a peak main memory bandwidth of M_∞ [Gwords/s] (1 word = 64 bits). The nodes are interconnected by a communication network with a total peak bandwidth of C_∞ [Gwords/s]. Then, one can define the following quantities

$$\begin{aligned} V_M &= \frac{R_\infty}{M_\infty} \\ V_C &= P\frac{R_\infty}{C_\infty}. \end{aligned} \tag{1}$$

These two parameters measure the number of floating point operations the processor can make during the transfer time of an operand from main mem-

Cluster	*Vendor*	*node*	*procs/ node*	*network 1*	*network 2*
NoW		Pentium 4	1	FE bus	
Pleiades1	Logics	Pentium 4	1	FE switch	
Pleiades2	DELL	Pentium 4	1	GbE switch	
Mizar	Dalco	Opteron	2	Myrinet	
Blue Gene	IBM	Power 4	2	Grid network	Fat Tree
Horizon	Cray	Opteron	1	3D Torus	
SX-5	NEC	vector	1	Switch	

Table 1: Some typical clusters.

Cluster	P	R_∞ [Gflops/s]	M_∞ [Gwords/s]	V_M	C_∞ [Gwords/s]	V_C
NoW	25	6.4	0.8	8	0.0016	100000
Pleiades1	132	5.6	0.8	7	0.2	3600
Pleiades2	120	5.6	0.8	7	1.8	360
Mizar	160	9.6	1.6	6	5	300
Blue Gene	4096	8	1	8	192	170
Horizon	1100	5.2	0.8	6.5	1760	3.3
SX-5	16	8	8	1	128	1

Table 2: Characteristic parameters of some clusters.

ory to cache (V_M) or from one computational node to another one (V_C).

Some typical machines are listed in Table 1, with their respective parameters in Table 2. The data corresponds to machines with one (NoW, Pleiades, Horizon) or two (Mizar, Blue Gene) processors per node. Specifically, the parameter V_M distinguishes between a vector machine ($V_M \approx 1$) and a RISC processor ($V_M \approx 7$). One also sees that the quantity V_C can vary from 1 for a vector machine to 100000 or even more for a bus-based machine. The cost of a machine often increases with decreasing values of V_C.

2.2 The Γ parameter

In the following analysis, we will assume that the tasks of a parallel application are well balanced, and that computations and communications do not overlap. Let assume that the execution time T on each computing node can be divided in two parts:

$$T = T_P + T_C \,, \tag{2}$$

where T_P is the time spent to compute and T_C the time spent to communicate and synchronise on each processor. Thus the speedup A of an application running on P processors can be expressed as:

$$A = \frac{PT_P}{T_C + T_P} = \frac{P}{1 + \frac{1}{\Gamma}} = eP \tag{3}$$

and e is the average CPU usage of the application or the efficiency (e=A/P). We define Γ as the ratio T_P/T_C and decompose T_P and T_C into application and hardware specific parameters. This allows to separate the two contributions:

$$\Gamma = \frac{T_P}{T_C} = \frac{O/r_a}{S/b} = \frac{O/S}{r_a/b} = \frac{\gamma_a}{\gamma_M}\,. \tag{4}$$

The quantity O denotes the number of operations per processor [flops] one has to perform during the execution of the application, and S is the amount of data (in 64-bit words) that has to be sent through the internode network by each processor [words]. The quantities b and r_a measure the peak effective bandwidth of the network for each processor [Gwords/s], and the peak performance of the application per processor [Gflops/s], respectively. If the data can be kept in cache, the value of r_a can be close to the peak performance R_∞ of a processor, or r_a can be related to the main memory bandwidth if the data has to be continuously loaded from main memory to cache or stored back to memory. In scientific applications, r_a varies between 10% and 100% of R_∞. The smaller r_a/R_∞, the bigger Γ, and the communication needs diminish.

We thus see that Γ is a parameter which expresses how a given hardware is suitable to efficiently run a given parallel application. For instance, a value of 1 means that the application spends as much time in communications than in processing, and is equivalent to a speedup of $P/2$, or $e = 0.5$. Γ should thus be as large as possible but experience shows that a value greater than 1 (around 2 or 3) corresponds to an acceptable match between the application and the hardware. Let us describe a few cost-effective application/machine combinations with values of $\Gamma > 1$.

2.3 The Γ of the different application/machine combinations

2.3.1 Embarassingly parallel applications

Embarassingly parallel applications are dominated by master/slave communications. No data is exchanged between slave nodes. In this case, $T_P >> T_C$ and thus $\Gamma >> 1$. As a consequence, very high γ_M communication networks such as a bus or the Pleiades1 cluster (see Table 2) can be used. A typical example is the *seti@home* project that collects computational cycles over the Internet. Other examples are massive data interpretation as it occurs in high energy physics, or the sequencing algorithms in genomics and proteomics.

2.3.2 Applications with point-to-point communications

Point-to-point communications typically appear in finite element or finite volume methods when a huge 3D domain is decomposed in subdomains [4] and an explicit time stepping method or an iterative matrix solver is applied. If the number of processors grows with the problem size, and the size of a subdomain is fixed, γ_a is constant, and, consequently, Γ does not change. The per processor performance is determined by the main memory bandwidth. The number O of operations per step is directly related to the number of variables in a subdomain times the number of operations per variable, whereas the amount of data S transfered to the neighboring subdomains is directly related to the number of variables on the subdomain surface. For huge point-to-point applications using many processing nodes, $\Gamma << 1$ for a bus, $2 < \Gamma < 10$ for the Pleiades1 cluster with a Fast Ethernet switch, $10 < \Gamma < 50$ for the Pleiades2 and Mizar clusters, and $\Gamma >> 100$ for Horizon. Hence, that kind of applications can run well on a cluster with a relatively slow and cost-effective communication network. Such an application has been discussed in [5].

2.3.3 Applications with multicast communication needs

The parallel 3D FFT algorithm is a typical example with important multicast communication needs. Here, γ_a decreases when the problem size is increased, and the communication network has to become faster. In addition, $r_a = R_\infty$ for FFT, γ_M is big, and, as a consequence, the communication paramater b must be big to satisfy $\Gamma > 1$. Such an application has been discussed in [4]. It has been showed that with a Fast Ethernet based switched network, the communication time is several times bigger than the computing time, even when the problem size is small. Such an application needs a faster switched network such as an efficient GbE, a Myrinet, a Quadrics, or an Infiniband network. If thousands of processors are needed, a special vendor specific machine such as Horizon or Blue Gene might be required.

3 Monitoring data from parallel applications

Each application has its own well suited parallel machine. This characteristic will in future be used to decide on which machine an application should be executed, which implies that the behaviour of an application has to be monitored for each run. To verify our model, data on the CPU usage was collected on the Pleiades1 cluster using the *sysstat* tool [6]. The gathering was made during the first 3 months of 2005, with snapshots being taken on each node every 10 minutes.

The top part of Fig. 1 shows the histogram of the 1682806 collected snapshots. The 10% zero CPU usage is due to non-allocated processors

when the scheduler blocks resources for a large job, to resources that are reserved for interactive testing and not used, to lost cycles due to a blocking in a parallel application, or to intensive I/O operations. The 100% usage peak is mainly due to single processor applications that represent about 20% of the total CPU time.

Parallel jobs running on Pleiades1 share their time between computations and MPI and I/O communications, and use on average 10 processors. The average utilization of CPUs is 64%, with two peaks around 55%, and 82%. This can be considered as a fair score by a low-cost cluster with a Fast Ethernet switch with V_M=3600 (see Table 2).

For the application analysis, we chose two user applications that consumed 17% and 9% of the total computing time during the considered period. Fig. 1 shows the distribution of CPU usage for one run of each application. The first application (middle of Fig. 1) comes from fluid dynamics. It used 32 processors and ran for 5570 minutes, leading to a profiling with 17824 (=557*32) snapshots. About 10% of the snapshots show a CPU usage of 0%, and 15% show a 100% usage. This application shows an average CPU usage of e=0.56, i.e. following eq. 3 a Γ of 1.27. It could run more efficiently on a machine with a better internode communication system, but we would need to determine whether the price/performance ratio would improve when going on a more expensive machine.

The second application (bottom graph of Fig. 1) comes from plasma physics. It also used 32 processors and ran for 1690 minutes, giving 5408 (=169*32) snapshots. Processors were idle for about 15% of the time. The efficiency was 75.5%, i.e. $\Gamma = 3.1$. This is a typical application that contributes to the peak around 82% CPU usage in the upper graph. The Pleiades1 cluster seems to be a well-suited machine for this application.

In the future, we shall need characteristic CPU usage profiles such as those given in Fig. 1 for each parallel application and machine in a GRID. Together with additional data on the behaviour of the applications, coming for instance from accounting data collected by the resource management system, we believe to be able to automatically parameterise them. We can then define a cost function based on these parameters through which it will be possible to decide on where to submit each application.

We have to mention that the zero CPU usage peak of the upper graph in Fig. 1 aggregates contributions from different sources: although I/O is the most frequent one, MPI message passing and idle processors in unbalanced jobs must be taken into account as well. In pathological cases, one task of a parallel job dies, and the other processors remain idle until the job is killed by the scheduling system.

These first results show that improvements must be made: the Γ model must include I/O, and being able to distinguish between the sources of inefficiencies would be most welcome. Monitoring already had a positive impact: badly behaving applications have already been detected and improved.

4 Conclusions

The goal of the ISS project is to make it possible to automatically detect the type of hardware that is well suited for a given application. We described a parameterisation of parallel machines and applications that allows to tailor a computational GRID to a set of applications, and checked the validity of the parameters against real executions. Although it is in its initial stage, the ISS project already has an interesting side-effect: badly behaving applications can already be detected and improved. Future work includes extending the current model to correct the weaknesses that were described, and using the data collected during job execution to improve the scheduling decision. This will enable us to constantly adjust a computational GRID to the needs of the applications.

Acknowledgements

ISS is part of the the SwissGrid programme managed by the CSCS. This paper is a co-operative work within the CoreGRID Network of Excellence.

References

[1] I. Foster, and C. Kesselman (Eds.), "The GRID Blueprint for a new Computing Infrastructure", Morgan Kaufman, San Francisco, 1999

[2] D. Erwin (ed.), *UNICORE plus final report – uniform interface to computing resource*, Forschungszentrum Jlich, ISBN 3-00-011592-7, 2003.

[3] Ed Seidel, Gabrielle Allen, Andr Merzky and Jarek Nabrzyski, GridLab–a grid application toolkit and testbed, Future Generation Computer Systems, Volume 18, Issue 8, October 2002, Pages 1143-1153.

[4] R. Gruber, P. Volgers, A. De Vita, M. Stengel, and T.-M. Tran, "Parameterisation to tailor commodity clusters to applications", Future Generation Computer Systems, **19**, 111-120 (2003)

[5] R. Gruber, and T.-M. Tran, "Scalability aspects on commodity clusters", EPFL Supercomputing Review, **14**, 12-17 (2004)

[6] http://perso.wanadoo.fr/sebastien.godard/

[7] http://egee-intranet.web.cer.ch/egee-intranet

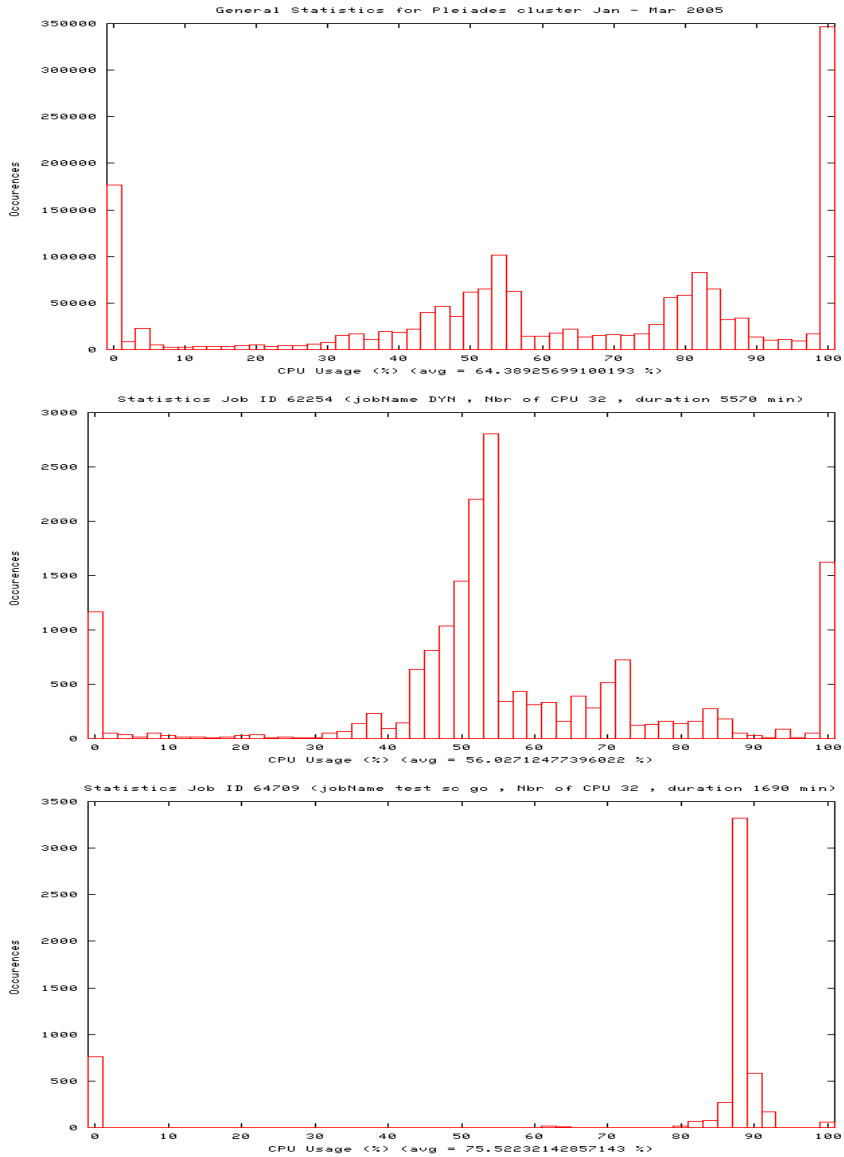

Figure 1: Up: CPU usage of all the 132 processors of the Pleiades1 cluster (V_M=3600) during the first 3 months in 2005. Average CPU usage was collected for each processor every 10'. The overall average CPU usage is 64%. Center: Profile of one job of a CFD application. Low: Profile of one job of a plasma physics application.