European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Application-oriented scheduling for HPC Grids

Kevin Cristiano[2], *Alain Drotz*[1], *Ralf Gruber*[1], *Vincent Keller* [1], *Peter Kunszt*[7], *Pierre Kuonen*[2], *Sergio Maffioletti*[7], *Pierre Manneback*[5], *Marie-Christine Sawley*[7], *Uwe Schwiegelshohn*[6], *Michela Thiémard*[1], *Ali Tolou*[1], *Trach-Minh Tran*[1], *Oliver Wäldrich*[4], *Philipp Wieder*[3], *Christoph Witzig*[8], *Ramin Yahyapour*[6], *Wolfgang Ziegler*[4]

[1]*École Polytechnique Fédérale, CH-1015 Lausanne, Switzerland*
[2]*École d'Ingénieurs et d'Architectes, CH-1705 Fribourg, Switzerland*
[3]*Forschungszentrum Jülich, D-52425 Jülich, Germany*
[4]*Fraunhofer SCAI, D-53754 St. Augustin, Germany*
[5]*CETIC, B-6041 Charleroi, Belgium*
[6]*IRF-IT, University of Dortmund, D-44221 Dortmund, Germany*
[7]*CSCS, CH-6928 Manno, Switzerland*
[8]*Switch, CH-8021 Zurich, Switzerland*

CoreGRID Technical Report
Number TR-0070
February 22, 2007

Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Application-oriented scheduling for HPC Grids

Kevin Cristiano[2], Alain Drotz[1], Ralf Gruber[1], Vincent Keller [1],
Peter Kunszt[7], Pierre Kuonen[2], Sergio Maffioletti[7], Pierre Manneback[5],
Marie-Christine Sawley[7], Uwe Schwiegelshohn[6], Michela Thiémard[1], Ali Tolou[1],
Trach-Minh Tran[1], Oliver Wäldrich[4], Philipp Wieder[3], Christoph Witzig[8],
Ramin Yahyapour[6], Wolfgang Ziegler[4]

[1]École Polytechnique Fédérale, CH-1015 Lausanne, Switzerland
[2]École d'Ingénieurs et d'Architectes, CH-1705 Fribourg, Switzerland
[3]Forschungszentrum Jülich, D-52425 Jülich, Germany
[4]Fraunhofer SCAI, D-53754 St. Augustin, Germany
[5]CETIC, B-6041 Charleroi, Belgium
[6]IRF-IT, University of Dortmund, D-44221 Dortmund, Germany
[7]CSCS, CH-6928 Manno, Switzerland
[8]Switch, CH-8021 Zurich, Switzerland

CoreGRID TR-0070

February 22, 2007

Abstract

The Intelligent Grid Scheduling Service (ISS) aims at finding an optimally suited computational resource for a given application component. An objective cost model function is used to decide it. It includes information on a parametrization of the components and the machines in a Grid, and on the availability of the clusters. The paper presents a detailed formulation of the environment and outlines the integration of the ISS model into the UNICORE-based VIOLA meta-scheduling Grid middleware. This document is an active collaboration between EPFL, EIA-FR, Forschungszentrum Jülich, Fraunhofer Gesellschaft, University of Dortmund, CETIC, CSCS, and Switch.

1 Introduction

The different communication needs of different HPC application components demand a Grid that can offer different parallel computer architectures: SMP or NUMA machines for shared memory parallel applications, a NoW (Network of Workstations) interconnected by a bus for embarrassingly 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 application components 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 other hand, running an embarrassingly parallel application on a costly NUMA machine, the user will probably not recognise this as a problem. In the future, one would like to choose a well suited hardware for an application component (according to peak processor performance, main memory bandwidth, or inter-node network communication system), and this in a most automatic manner.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

The ISS (Intelligent Grid Scheduling Service) project is precisely aimed at solving this latter problem. In a first phase, the ISS middleware will be integrated with UNICORE [5] and the MetaScheduling Service (MSS) [6] developed by the German VIOLA project [27]. In a later phase, ISS could also be embedded in other existing Grid middlewares such as Globus [21], EGEE [19], or GridLab [20].

The architecture of the ISS system is built around a Smart Grid Node (SGN) that includes a Data Warehouse (DW), a System Information (SI), a Resource Discovery System (RDS), a Resource Broker (RB), and a Monitoring System (MS). The RDS is looking for all eligible machines that can satisfy binary constraints such as user rights, existence of a program on the machine, memory size, availability of a token, or number of processors. The RB includes the Cost Function Model (CFM) in which the sum over all the relevant real and virtual costs is minimized with respect to constraints on time and money. The system submits the job to the machine having the lowest value of the cost function. This function includes costs related to CPU time, license fees, maintenance, interests on the investment, electrical energy, data transfer, and waiting time expressed in form of salaries or time-to-market losses. The data needed in the CFM on the computer architecture (node performance, memory bandwidth, network communication) and on the behaviour of an application on a machine (number of operations and memory accesses, number and size of messages) are collected after execution by the MS, put on the DW and reused to compute the CFM for the next submission. Data on the availability of the different machines in the Grid are delivered by the MSS and enter the CFM just prior to execution.

The ISS/VIOLA middleware [2, 27, 5, 6] will be validated by more than 100'000 job executions from the Pleiades clusters [23] collected by the VAMOS monitoring system during one year. A simulator has been written to apprise a number of free parameters in the CFM.

The ISS concept is first presented, with a short description of the Γ model [3] and a description of the different types of components that can be found in HPC applications. The cost function model is then detailed. It incorporates a set of free parameters and functions that have either to be given by the computing centres or are determined through simulation. The architecture of the VIOLA meta-scheduling environment is then discussed, followed by a detailed execution scenario by means of the real-life plasma physics application ORB5.

An implementation plan follows in Chapter 7. Different simulators are discussed in Chapter 8. In a first phase, the SwissGrid testbed will be used to run the first ISS/VIOLA prototype middleware. In a near future, the concept of the HPC Grid will be generalised to Switzerland

2 ISS concept

The ISS middleware is supposed to help deciding on which Grid resources a scientific application should be executed. Such an application consists of $k = 1, ..., n$ components, called C_k, to be executed on the machine i which is one of the r resources in a Grid. To help finding the adequate resource, all the application components and the machines are parametrized using the Γ model [3]. This parametrization and other information on the availability of the resources are used to determine the optimal machines by a cost function model.

2.1 Grid Architecture

Suppose that a Grid consists of $i = 1, ..., r$ machines, each one having P computational nodes (the indexes i and k are omitted in this chapter). Each node has a 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 perform during the transfer time of an operand from main memory to cache (V_M) or from one computational node to another

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

Table 1: Some typical machines.[1] For the Cray Machine, baby system is dual cores, production system one core

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
Pleiades2+	92	21.3	2.7	8	1.4	1400
Mizar	160	9.6	1.6	6	5	300
Blue Gene	4096	8	1	8	192	170
Cray XT3	1664	5.2	0.8	9.8	1760	3.3
SX-5	16	8	8	1	128	-

Table 2: Characteristic parameters of some clusters.

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 3.3 for a Cray XT3 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 Γ model

In the following analysis, we will assume that the tasks of a parallel application component C_k are well balanced, and that computations and communications do not overlap. Let assume that the total execution time T 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. The speedup A of a C_k running on p_k processors can be expressed as:

$$A = \frac{p_k T_P}{T_C + T_P} = \frac{p_k}{1 + \frac{1}{\Gamma}} = e p_k \tag{3}$$

where

$$T_C = T_S + T_L = \frac{S}{b} + LZ. \tag{4}$$

T_S is the time to transfer the data from one node to another one, S the message size in 64bit words, b the network communication bandwidth in words/s, T_L the total latency time in seconds, L the latency time per message, Z the number of messages, and e is the average CPU usage of C_k or the efficiency ($e = A/p_k$). In a GbE, for a message size of 200 64bit words, $T_S \approx T_L$.

We define Γ as the ratio T_P/T_C and decompose T_P and T_C into component and hardware specific parameters. For $T_L \ll T_S$, one can separate the two contributions:

$$\Gamma = \frac{T_P}{T_C} = \frac{O/r_a}{S/b} = \frac{O/S}{r_a/b} = \frac{\gamma_a}{\gamma_M}\,. \tag{5}$$

The quantity O denotes the number of operations per node [flops] one has to perform during the execution of C_k, and S is the amount of data (in 64-bit words) that has to be sent through the internode network by each node [words]. The quantities

$$\begin{aligned} b &= C_k/ < d > P \\ r_a &= Min(R_\infty, V_a * M_\infty) \\ V_a &= O/W \end{aligned} \tag{6}$$

measure the peak bandwidth of the network per node [Gwords/s], the peak performance of the application component C_k per node [Gflops/s], and the average number of times data can be found in cache, respectively. The quotient $< d >$ in the equation for b is the average distance between two nodes in the communication network and W is the number of 64bit words that have to be transferred from main memory to cache. If $V_a \geq V_M$, $r_a = R_\infty$. If $V_a < V_M$ r_a is directly related to the main memory bandwidth. 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.

One sees that Γ can be used to get a good insight on the suitability of a given hardware to run C_k efficiently. For instance, a value of $\Gamma = 1$ means that C_k spends as much time in communications than in processing, and is equivalent to a speedup of $p_k/2$, or $e = 0.5$. In fact, Γ should be as large as possible but the larger Γ is, the more expensive the communication network. We have to find a compromise. Experience shows that $\Gamma \geq 2$ corresponds to a cost-effective match between C_k and the hardware. Let us describe a few of such cost-effective component/machine combinations.

2.3 HPC applications

2.3.1 Embarrassingly parallel applications

These applications do not demand inter-node communications. A big number of cases have to be distributed among many slave nodes, the results collected and handled by a server. 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, the Pleiades1 cluster (see Table 2), or even the Internet can be used. A typical example is the *seti@home* project that collects computational cycles over the Internet. Other examples of such applications are the immense amount of independent data in high energy physics that has to be interpreted, the sequencing algorithms in proteomics, parameter studies in plasma physics to predict optimal magnetic fusion configurations, or a huge number of data base accesses for statistical reasons.

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 [9] 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 transferred to the neighboring subdomains is directly related to the number of variables on the subdomain surface, and O/S becomes big. 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 Cray XT3. Hence, that kind of applications can run well on a cluster with a Fast Ethernet or a GbE switch.

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 parameter b must be big to satisfy $\Gamma > 1$. Such an application has been discussed in [3]. It has been showed that with a Fast Ethernet based switched network, the communication time is several times bigger than the computing time. It needs a fast switched network, such as Myrinet, Quadrics, Infiniband, or special vendor specific networks such as those of a Cray XT3 or an IBM BlueGene.

2.3.4 OpenMP applications

There are a few applications that demand a shared memory computer architecture. The parallelism of the component is expressed with OpenMP. A typical example is the one described in [10]. This implies that a HPC Grid should also include SMP nodes that can run OpenMP applications such as the new multi-cores and multi-processors units (Intel Woodcrest or AMD Socket F)

2.3.5 Components based applications

An application can be separated into components. If inter-component communication is not too big, each component can run on a separate machine. This is the reason why we talk about components instead of applications. However, most of the present HPC applications consist of one single component.

3 Cost Function Model

3.1 Mathematical formulation

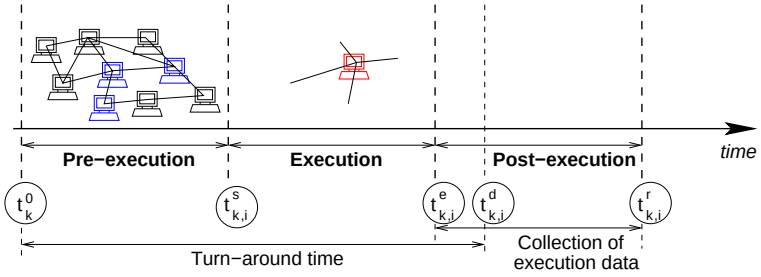

Figure 1: ISS job submission timing

The choice of a well suited machine depends on user requisites. Some users would like to obtain the result of their application execution as soon as possible, regardless of costs, some others would like to obtain results for a given maximum cost, but in a reasonable time, and some others for a minimum cost, regardless of time.

We will describe here in a few words the various elements that compose a cost function z being able to satisfy users' requests. This cost function depends on costs due to machine usage, denoted by K_e, license fees K_l, energy consumption and cooling K_{eco}, waiting results time K_w, and amount of data transferred K_d. All these quantities depend on the application components (C_k), on the per hour costs (K_i) on machine (R_i) with altogether P_i computational nodes, on the number of processors (p_k) used in the computation for each component, and on data transfer costs over the Internet. The user can prescribe the two constraints K_{MAX} (maximum cost) and T_{MAX} (maximum turn around time). The optimization problem writes:

$$\min z = \beta K_w \left(\bigcup_{k=1}^{n} (C_k, R_i, p_k) \right) + \sum_{k=1}^{n} \mathcal{F}_{C_k}(R_i, p_k)$$

$$\text{such that} \quad \sum_{k=1}^{n} \Big(K_e(C_k, R_i, p_k) + K_l(C_k, R_i, p_k) + K_{eco}(C_k, R_i, p_k) + K_d(C_k, R_i, p_k) \Big) \leq K_{MAX}$$

$$\max(t^d_{k,i}) - \min(t^0_k) \leq T_{MAX}$$

$$(R_i, p_k) \in \mathcal{R}(C_k),$$

$\forall\ 1 \leq k \leq n$, where

$$\begin{aligned} \mathcal{F}_{C_k}(R_i, p_k) &= \alpha_k \Big(K_e(C_k, R_i, p_k) + K_l(C_k, R_i, p_k) \Big) \\ &+ \gamma_k \Big(K_{eco}(C_k, R_i, p_k) \Big) \\ &+ \delta_k \Big(K_d(C_k, R_i, p_k) \Big) \quad [\text{ECU}], \\ &\alpha_k,\ \beta,\ \gamma_k,\ \delta_k \geq 0, \\ &\alpha_k + \beta + \gamma_k + \delta_k > 0, \end{aligned}$$

and $\mathcal{R}(C_k), k = 1, ..., n$ is the eligible set of machines for component C_k. We express the money quantity as Electronic Cost Unit ([ECU]). The quantities t^0_k and $t^d_{k,i}$ represent the job submission time and the time when the user gets the result, respectively (see Fig. 1).

In our model, the parameters α_k, β, γ_k, and δ_k are used to weight the different terms. They can be fixed by the users and/or by a simulator. For instance, by fixing $\alpha_k = \gamma_k = \delta_k = 0$ and $\beta \neq 0$, one can get the result as rapidly as possible, independent of cost. By fixing $\beta = 0$ and $\alpha_k, \gamma_k, \delta_k \neq 0$, one can get the result for minimum cost, independent of time. These four parameters have to be tuned according to the policies of the computing centres and user's demands. In the case of the Swiss Grid Initiative, the overall usage of the machines should be high. For instance, increasing β will increase usage of underused machines. One recognizes that a simulator, presented in section 8, is needed to estimate these parameters. In fact, the user's (resource consumer) and the computing center's (resource furnisher) interests are complementary, the first ones would like to get a result as soon as possible and for the smallest costs, and the second ones would like to get highest profit. The simulator will be used to try to satisfy both somewhat contradictory goals. This implies a constant tuning of the free parameters.

3.2 CPU costs K_e

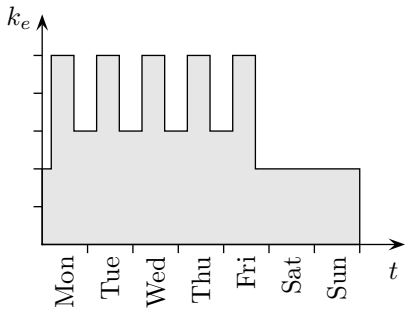

Figure 2: Example of CPU costs as a function of daytime.

$$K_e(C_k, R_i, p_k) = \int_{t^s_{k,i}}^{t^e_{k,i}} k_e(C_k, R_i, p_k, \varphi, t)\, dt\, [\text{ECU}].$$

Each computing center has its specific accounting policy, but often they just bill the number of CPU hours used. Figure 2 shows an example of $k_e(t)$ when day time, night time and weekends have different CPU costs.

The CPU costs include the investment S_c^i made at the start of the service period T_0^i, the maintenance fees S_m^i, the interests S_b^i that have to be paid to the bank, the personnel costs S_p^i, the infrastructure S_I^i including the building, the electricity installations, and the cooling, the management S_a^i overhead, the insurance fees S_f^i, and the margin S_g^i. If a real bill is sent to a user, sales tax has to be included. Presently, the costs for CPU time, data storage and archiving are not separated. In future, a special (costly) effort has to be made to guarantee data security. Note, that the energy costs E_h^i per hour and node are taken care of by a separate term in the cost function.

The price/performance ratio of the most recent machines appearing on the market reduces typically by a factor of close to two every year. This implies that the investment S_c^i should enter the CPU costs in a non-linear manner. It is reasonable to define a regression curve $\rho(T, R_i)$ for each machine in the Grid that measures the depreciation of the resource as a function of time

$$\rho(T, R_i) = \frac{S_c^i r_i ln(y_i)}{1 - y_i^{-r_i T_i}} y_i^{-r_i T}$$

with

$$\int_{T_0^i}^{T_0^i + T_i} \rho(T, R_i) dT = S_c^i$$

that takes this fact into account. The machine installation date is T_0^i, the life time in years of a machine is T_i and T is the running time in years. Choosing $y_i = 2$, $r_i = 1$, and $T_i = 3$ implies that the value of a machine reduces by a factor of 2 every year, and that the machine will be closed after 3 years.

To compute the CPU costs of C_k it is supposed that $k_e = 1$, not changing during the week time. Admitting that the machine with P_i nodes runs with an efficiency of $e_i\%$ over the year ($d = 8760$ hours/year), the CPU cost K_e of C_k (p_k nodes, execution starts at t_s^k, and ends at t_e^k) is

$$K_e(C_k, R_i, p_k) = p_k \left[\frac{S_c^i}{1 - y_i^{-r_i T_i}} \left(y_i^{-\frac{r_i}{de_i}(t_s^k - t_0^i)} - y_i^{-\frac{r_i}{de_i}(t_e^k - t_0^i)} \right) + S_i(t_e^k - t_s^k) \right]$$

where

$$S_i = (S_m^i + S_b^i + S_p^i + S_I^i + S_a^i + S_f^i + S_g^i)/(de_i P_i).$$

The new quantity S_i denotes the fix costs per CPU hour for one node, and t_0^i is the age of machine i in hours. With the normalisation of r_i by de_i, the times t_s^k and t_e^k are measured in hours (upper case times are in years, lower case times are in hours). All those values can be given by the computing centre through a GUI described later on. With the ISS model, we hope that it will be possible to estimate T_i, i.e. the time at which a machine should be replaced by a more recent one.

The φ parameter introduces the **priority** notion (see [22] for details). Some computing centres do not permit priority ($\varphi = 1$ for all users). Others accept preemption for users who have to deliver results at given times during the day. A good example is weather forecast that has to be ready at 6 pm such that it can be presented after the news at 8. This implies that the needed resources have to be reserved for the time needed to finish at 6, and this every day. All jobs running on those nodes at start time of the weather forecast must be checkpointed and rerun after 6. The CPU time of preempted jobs should cost more, whereas the checkpointed jobs should benefit from a cost reduction.

If priority can be used without preemption, it is necessary to define a very strict policy. In this case, a high priority job jumps ahead in the input queue, increasing the waiting time of all the jobs that are pushed back. As a consequence, higher priority should imply higher CPU costs, and lower CPU costs for all those jobs that end with higher turn-around times.

In the academic world (as at CSCS), a user often gets a certain monthly CPU time allocation. When this time is passed, the priority automatically is lowered. As a consequence, his jobs stay longer in the input queue, or, according to the local policy, he only enters a machine when the input queue of higher priority jobs is empty.

During a first phase, priority is put to 1 for all C_k.

3.3 License fees

$$K_l(C_k, R_i, p_k) = \int_{t_k^s}^{t_k^e} k_l(C_k, R_i, p_k, t)\, dt\,[\text{ECU}].$$

A license fee model is very complex. The most simple model is to directly connect the license fees to the CPU costs, $K_l = aK_e$. In some cases the computing centre pays an annual fee and puts this fee into the CPU time, $a = 0$. Clearly, those users who do not use this program are not happy to pay for other users. Another simple model is to pay only if the program is really used. Then, the fee can directly be proportional to the CPU costs, $a > 0$. This model is applied when the CFD code FLUENT is used in a project including academia and industry. In a first phase, we will restrict ourselves to these two models.

Note that the licensing problem also affects the availability of tokens. Specifically, if not enough tokens are free, the program has to wait until he can get them. In a first step, we propose to solve the token problem in the prologue phase. If there is no token at t_0^k, then the machine is not eligible.

3.4 Costs due to waiting time

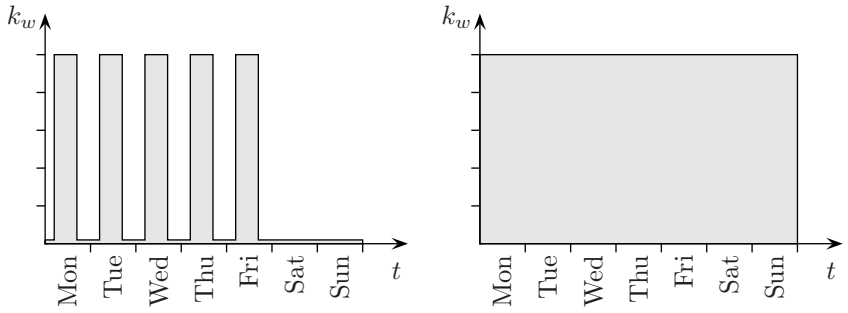

Figure 3: Examples of waiting cost graphs. Left: Engineer's salary cost function $k_w(t)$ due to waiting on the result. Right: Time-to-market arguments can push up priority of the job.

$$K_w(C_k, R_i, p_k) = \int_{t_k^0}^{t_k^e} k_w(C_k, t)\, dt\,[\text{ECU}].$$

This cost is machine and application component dependent since t_k^e is machine and component dependent. It could be engineer's salary or a critical time-to-market product waiting cost.

Figure 3 shows an example of k_w concerning engineer's salary. Here, it is supposed that the engineer looses his time only during working hours. A more sophisticated function could be yearly graphs also including unproductive periods like vacations. Figure 3 also shows an example of k_w of a critical time-to-market product.

But this cost has to be computed over all application components. It could be written as following:

$$K_w\left(\bigcup_{k=1}^{n}(C_k, R_i, p_k)\right) = \int_{t_1^0}^{t_n^e} k_w\left(\bigcup_{k=1}^{n}(C_k), t\right) dt\,[\text{ECU}].$$

This parameter could also be used to tune the overall usage of the whole machine park of a user community. Increasing β in the cost function will activate machines that are underused. Putting $\beta = 0$ in the simulator offers the opportunity to recognize overused machines, i.e. type of resources that should be purchased in future.

3.5 Energy costs

$$K_{eco}(C_k, R_i, p_k) = \int_{t_k^s}^{t_k^e} k_{eco}(C_k, R_i, p_k, t)\, dt\,[\text{ECU}].$$

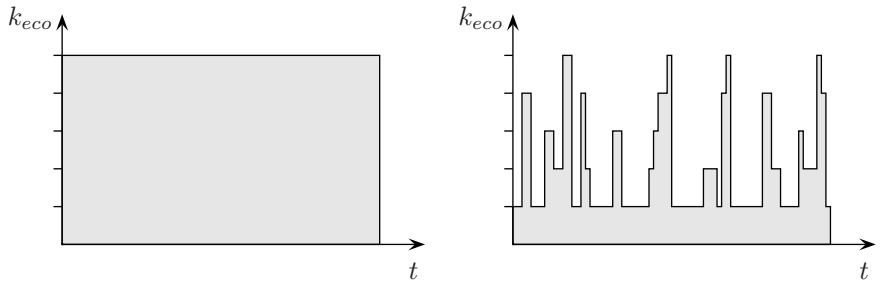

Figure 4: Examples of graphs for the energy costs. Today (left): Excessive costs of energy consumption and cooling. Future (right): Energy consumption reduction due to frequency adaptation to application component needs. Computer manufacturers are invited to open for on-line frequency underclocking.

Energy costs over the lifetime of a node are a non-negligible part in the cost model. It enters strongly when the machine becomes old, and the investment costs become a small part of the CPU costs. For components that are memory bandwidth bound, the frequency of the processor could be lowered. The energy consumption grows with the second power of the frequency, a reduction by a factor of 2.5 of the processor frequency reduces its energy consumption by a factor of 6. Tests have been made with a laptop computer. When reducing frequency from 2 GHz to 800 MHz, the overall performance of a memory bandwidth bound application was only reduced by 10%. We have to mention here that for low-cost PCs energy costs (power supply + cooling) over 5 years can become comparable to investment costs. Thus, in future it is crucial to be able to underclock the processor, adapting its frequency to the application component needs [7]. This could reduce the worldwide PC energy consumption by a factor and could free in the near future many nuclear power plants. Computer manufacturers must be convinced to be able to have energy consumption graphs as the one depicted at the right of Figure 4.

In fact, the hourly energy costs for one node corresponds to

$$E_h^i = \int_{t_k^s}^{t_k^e} E_i(t) F_i dt,$$

where $E_i(t)$ and F_i are the hourly energy consumption of one node (electricity and cooling), and the price per kWh, respectively.

3.6 Data transfer costs

Let us consider that different application components run on different servers located in different computing centers. The following data has then to be transferred between the different sites:

- Transfer of the component and its input data between the client and the computing center (client-server, cs)
- Data transfer between the different components (server-server, ss)
- Data transfer during execution to the client, for instance for remote rendering (server-visualisation, s-v)
- Transfer of the final result to the client (server-client, sc)

Then:

$$K_d(C_k, R_i) = K_{d,cs}(C_k, R_i) + K_{d,ss}(C_k, R_i) + K_{d,sv}(C_k, R_i) + K_{d,sc}(C_k, R_i)$$

In Switzerland there is no precise model that estimates these K_d quantities. Presently, the traffic into the commodity Internet is charged, but only during peak traffic periods (Monday to Friday, 08:00-20:00), 1ECU/GB for academic users, 3ECU/GB for others. In addition, there are flat rates for connecting to the

Item	*Pleiades 1* $i=1$	*Pleiades 2* $i=2$	*Pleiades 2+* $i=3$
T_0^i	01.01.2004	01.01.2006	01.01.2007
Nodes	Pentium 4	Xeon	Woodcrest
Architecture	32bits	64bits	64bits
Operating System	Linux SUSE 9.0	Linux SUSE 9.3	Linux SUSE 10.1
P_i	132	120	92
Procs/node	1	1	4
R_∞	5.6 Gflops/s	5.6 Gflops/s	21.33 Gflops/s
M_∞	0.8 Gwords/s	0.8 Gwords/s	2.67 Gwords/s
V_M	7	7	8
Network	Fast Ethernet switch	GbE switch	GbE switch
y_i	1.5	2	2
r_i	1/year	1/year	1/year
T_i	4.5years	3years	3years
E_i	0.4 kW	0.4 kW	0.4 kW
u_i	0.8	0.72	0.76
F_i	0.1 /kWh	0.1 /kWh	0.1 /kWh
S_c^i	320k	270k	420k
S_m^i	20k	0	0
S_b^i	16k	14k	21k
S_p^i	100k	85k	135k
S_I^i	30k	28k	22k
S_a^i	50k	40k	70k
S_f^i	0	0	0
S_g^i	0	0	0
S_i	0.23	0.22	0.40
E_h^i	0.04	0.04	0.04
$\rho(i, 01.01.2007)$	46k	93k	290k
K_ρ^i	0.05	0.12	0.47
K_i	0.32	0.38	0.91

Table 3: Characteristic parameters for the Pleiades clusters.

Internet in dependence of the bandwidth (K_dc)[1] and size of the university (K_ds). In the case of a specific university that transfers about 160 TB/year, the mix of these costs result in an estimated GB transfer price of the order of 2.5ECU/GB (= 1ECU + (K_dc+K_ds)/(160TB)).

3.7 Graphical user interface

The cost model must be tuned for each machine by each administrator in a non-centralized manner. This means that the "server side" of ISS must provide a simple tool (like a GUI application or a webpage) to tune the cost model parameters.

Note that these parameters should also be tuned with a simulator.

3.8 Example: The Pleiades clusters

Let us give an example of how to determine the CPU and energy costs of the three Pleiades clusters.

In Table 3 all the values representing costs are given in arbitrary units. A "k" after a number means "thousand". The interests S_b^i, the personnel costs S_p^i, and the management overhead S_a^i are distributed among the three machines according to the initial investment S_c^i. The infrastructure costs are distributed

[1] large universities have 10 Gbit/sec

with respect to the number of nodes. For the Pleiades 1 machine y_1 has been chosen such that after 5 years the value of one node corresponds to the value of one Pleiades 2 node after 3 years. The idea behind is that a single node of Pleiades 1 and Pleiades 2 have the same performances, even though Pleiades 1 has been installed 2 year before. The quantity $\rho(i, 01.01.2007)$ corresponds to the basis value of the machine i at first of January 2007.

The result

$$K_i = K_\rho^i + S_i + E_h^i$$

reflects the total hourly costs (investment, auxiliary, and energy) of one computational node at 01.01.2007, and K_ρ^i is the hourly node cost contribution due to the investment costs. The newest installation, Pleiades 2+ , consisting of the most recent Woodcrest nodes with two dual cores each one is 3 to 5 times more powerful than Pleiades 1 or Pleiades 2. This factor depends on the type of applications. Thus, from a user point of view, the Woodcrest machine is clearly the most interesting machine to choose, since 4 Pleiades 1 or Pleiades 2 nodes cost about 50% more than one Woodcrest node. The performance/price ratio is about 50% better for Woodcrest than for the two other machines.

4 ISS/VIOLA architecture

4.1 Overall ISS/VIOLA architecture

The overall architecture of the ISS/VIOLA system is depicted in Fig. 5. The different modules and services are presented in the following sections.

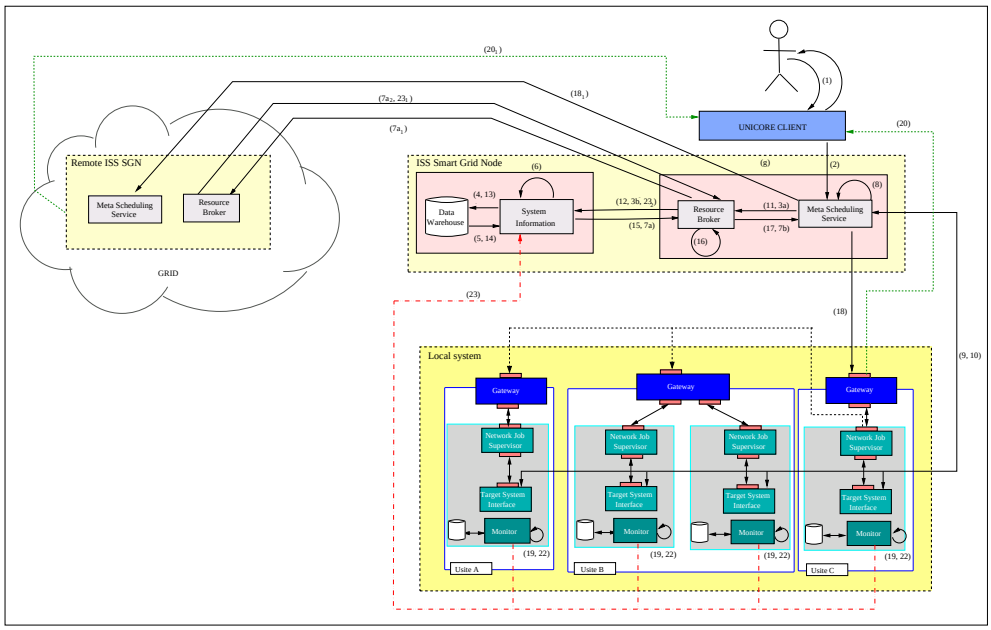

Figure 5: The overall architecture of the ISS/VIOLA system

4.2 UNICORE

Please note that any reference to the UNICORE Grid middleware made in this paper is related to UNICORE version 5 [17], the production-ready version of UNICORE. The succeeding version, UNICORE version 6, is currently developed in a number of European projects.

A workflow is in general submitted to a UNICORE Grid via the UNICORE Client (see Fig. 6) which provides means to construct, monitor and control workflows. In addition, the client offers extension capabilities through a plug-in interface, which has for example been used to integrate the Meta-Scheduling Service

into the UNICORE Grid system. The workflow then passes the security Gateway and is mapped to the site-specific characteristics at the UNICORE Server before being transferred to the local scheduler.

The concept of resource virtualisation manifests itself in UNICORE's Virtual Site (Vsite) that comprises a set of resources. These resources must have direct access to each other, a uniform user mapping, and they are generally under the same administrative control. A set of Vsites is represented by a UNICORE Site (Usite) that offers a single access point (a unique address and port) to the resources of usually one institution.

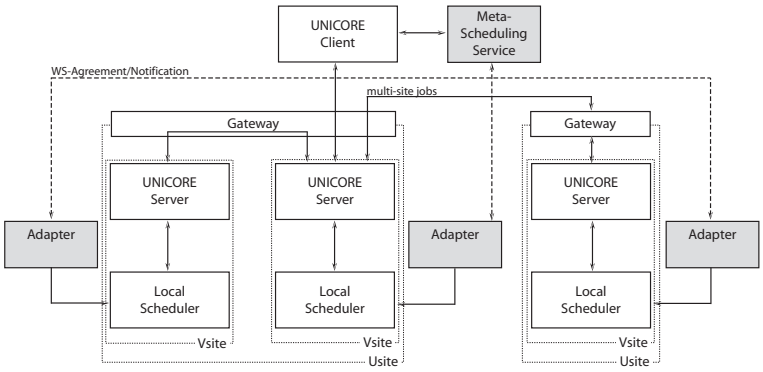

Figure 6: Architecture of the VIOLA meta-scheduling environment

4.3 MetaScheduling Service

The meta-scheduler is implemented as a Web Service receiving a list of resources preselected by a resource selection service (a broker for example, or a user) and returning reservations for some or all of these resources. To achieve this, the MetaScheduling Service first queries selected local scheduling systems for the availability of these resources and then negotiates the reservations across all local scheduling systems. In the particular case of the meta-scheduling environment the local schedulers are contacted via an adapter which provides a generic interface to these schedulers. Through this process the MetaScheduling Service supports scheduling of arbitrary resources or services for dedicated times. It offers on one hand the support for workflows where the agreements about resource or service usage (aka reservations) of consecutive parts should be made in advance to avoid delay during the execution of the workflow. On the other hand the MetaScheduling Service also supports co-allocation of resources or services in case it is required to run a parallel distributed application which needs several resources with probably different characteristics at the same time. The meta-scheduler may be steered directly by a user through a command-line interface or by Grid middleware components like the UNICORE client through its SOAP interface (see Fig. 6). The resulting reservations are implemented using the WS-Agreement specification [31].

4.4 Resource Broker

The Resource Broker (RB) is responsible for two distinct tasks : the cost function calculation and the starting of resource discovery process described in detail in section 4.11 of this document. The cost function has been described in section 3. The RB is the only part of ISS that connects to the VIOLA Metascheduling Service.

The RB computes a list of best suited machines for each component and sends it to the MSS for decision.

This list uncludes all machines for which $z_{min} \leq z \leq z_{min} + tol$, where tol is a tolerance value to be given for each component.

4.5 Data Warehouse (DW)

The DataWareHouse (DW) is the repository of all the informations related to the application components, to the resources found, to the services provided by the V-Sites, to the monitoring after each execution, and

to some other useful information (like the cost of an hour of an engineer taken into account in the cost function). Specifically, the DW contains the following informations:

1. **Resources :** Application independent hardware quantities.
2. **Services :** Which services does the machines provide (software, libraries installed, etc...).
3. **Monitoring :** Application dependent hardware quantities collected after each execution.
4. **Applications :** Γ model quantities computed after each execution.
5. **Other :** Other informations needed for the cost function such as cost of one hour engineering time, tolerance, priority, or for the resource discovery process (information about the neighborhood, ...).

The Data Warehouse includes stable information and volatile information. In this context, the stable part of the DW uses a schema for resource modelling which includes some information about the cost function, information about grid resources and also other kind of information needed.

The volatile part of the DW is managed by a database in which some information about the network and other information related to the resource discovery, the monitoring are stored.

4.6 System Information (SI)

The System Information is the frontend of the Data Warehouse. It receives information from the Monitoring Module (MM) if the chosen machine was a local one, from the remote RB through the local RB if the chosen machine was a remote one.

The SI has the capability to estimate, using the Γ model, how a component will behave on an unknown machine, according to the behavior known on a known machine.

All historical data about a component needed in the cost function computation are sent to the RB.

4.7 Monitoring Module (MM)

The Monitoring Module (MM) collects the information about the behavior (MFLOPS/s rate, memory needs, cach misses, communication, network relevant information, etc..) of the component during its execution. At the end of the execution, the MM prepares and sends data to the SI. These data will be resued later for the evalutation of the cost function.

4.8 VAMOS: Attribute monitored data to application components

The goal of VAMOS is to monitor the behaviour of a specific application component and to collect application-oriented data such as the CPU usage figures as the one for the whole machine (Fig. 7). For this purpose, the system has to map hardware monitored data (Ganglia for instance) to the accounting data specific to the application and the user (the local RMS).

On the accounting files it is possible to get information about start and end of the execution, and on the number of processors that have been reserved during this period of time. VAMOS supposes that all the reserved processors have fully been attributed to one single application (no node sharing). For HPC applications, this makes sense since most of the existing parallel HPC applications are coded such that in each task the computing time between two barriers is about the same. If, in such a situation, one node is part-time taken to run on another program, all the other tasks must wait at the barrier.

Each application runs differently on different computational resources. The Γ model presented in section 2.2 enables a parametrization of the behavior of an application on a machine. Parameters valid on one machine can also be used to predict the behavior of the same application on another machine. These parameters can be determined with historic monitored data stored after each execution.

The VAMOS tool has been implemented with this background model. It uses the well accepted Ganglia monitoring system and the RMS data on users and accounting (an interface to OpenPBS, Torque and PBSPro is implemented).

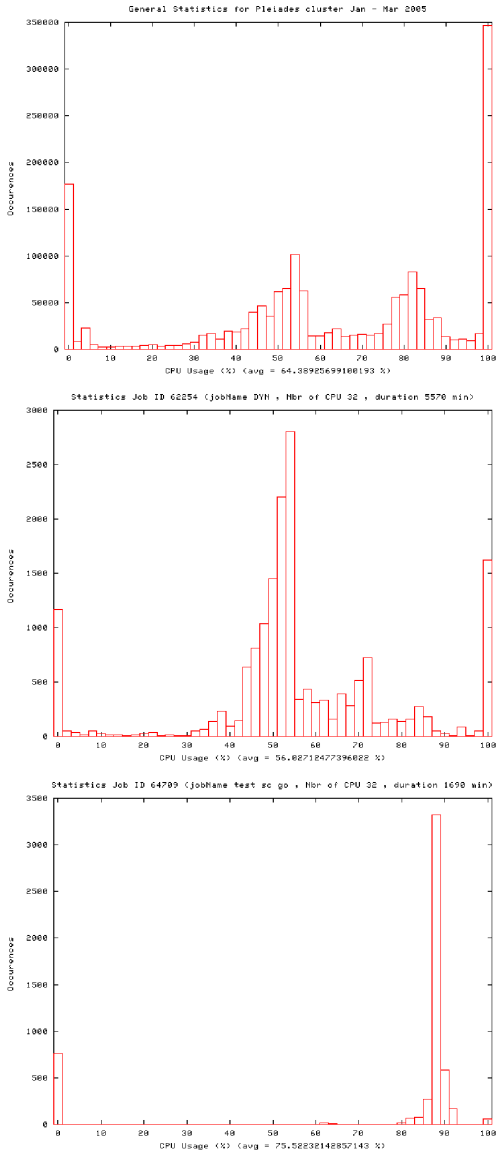

Figure 7: Up: CPU usage of all the 132 processors of the Pleiades1 cluster (V_M=3600) during the first 3 months in 2005. Average CPU usage was collected for each processor every 10'. The overall average CPU usage is 64%. Center: Profile of one job of a CFD application. Low: Profile of one job of a plasma physics application.

As an example, data on the CPU usage was collected on the Pleiades1 cluster using VAMOS [7]. 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. 7 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 during which processors are idle. 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. 7 shows the distribution of CPU usage for one run of each application. The first application (middle of Fig. 7) 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. 7) 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.

We have to mention that the zero CPU usage peak of the upper graph in Fig. 7 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 have to 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.

We show in Figure 8 the behavior of the SpecuLOOS fluid dynamics code on 3 different machines of the Pleiades cluster (see 3.8). Data have been collected with VAMOS. The conditions of these 3 runs were the same on each cluster : 32 processing elements running the same problem for 10 hours.

The number of iteration performed during this time was 1291 on Pleiades1, 1827 on Pleiades2 and 1206 on Pleiades2+. Thus, according to table 3, the CPU cost per 1000 iterations was 7.36 on Pleiades1, 6.65 for Pleaides2 and 6.04 for Pleiades2+. It the meantime, we have discover that using Nemesis-MPICH instead of MPICH further reduces the costs on the Pleiades2+ cluster. As consequence, the most cost effective machine for this application is Pleiades2+.

4.9 Archiving Module

Periodicaly, the content of the DW is reduced by the SI. The eliminated data are stored in an archiving module (AM) for further statistical evaluation. The goal of these statistics here is to detect and to help to decide on future optimal hardware installations.

4.10 The ISS Smart Grid Node (ISS-SGN)

All the elements presented in section 4 (RB, MSS, SI and DW) form the, so called, ISS Smart Grid Node (ISS-SGN). ISS-SGN is an instance of the more generic concept of *Smart Grid Node* (SGN) as presented in [15]. A SGN is a grid node which has the capability to evolve progressively during his life time according to requests it receives from its environment and to actions it performs. The concept of SGN is a virtualisation of a computer network as sketched on figure 5. It can represent different types of hardware ranking from a single workstation to the front end of a local network or of a supercomputer. Each SGN is connected to other SGNs thus forming a network of GRID nodes and manages a local system (see figure 9). A SGN evolves thanks to information contained in its DW. This information is regularly updated using information gathered by the SI. One important mechanism to gather information on surrounding SNGs is *the resource discovery process*. This process is presented in the next section.

4.11 Resource discovery

As mentioned in the broker section (section 4.4), the SGN concept contains a resource discovery mechanism [15]. In this section, we present this process in the context of the ISS-SGN. When a workflow is sent to the initial ISS-SGN, called initial node n_0, the request is analysed, the local resources are checked using

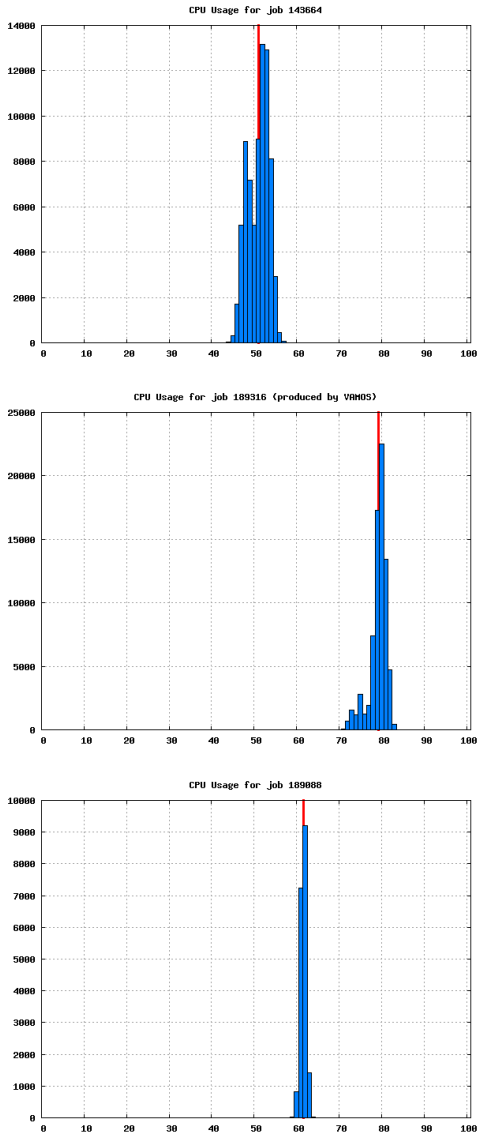

Figure 8: **The fluid dynamics code SpecuLOOS[8] CPU usage on different machines.** Up : CPU usage of SpecuLOOS on Pleiades1 cluster (32 Pentium IV processors with FastEthernet interconnect). Middle : CPU usage of SpecuLOOS on the Pleiades2 cluster (32 Xeon with GigaBitEthernet switch). Low: CPU usage of SpecuLOOS on the Pleiades2+ cluster (8 nodes of bi-dual cores Woodcrest processors with GigabitEthernet switch).

information contained into the DW and if necessary the resource discovery process is started. According to the list of its direct neighbours contained in the DW, the workflow is sent, through the RB, to remote SGNs. Each resource discovery request contains a unique identifier to avoid the creation of cycles during the resource discovery process; already received requests are skipped. When n_0 starts the resource discovery process, it inserts into the request a list containing its identity followed by identities of all its direct neighbours. During the resource discovery process, when a SGN receives a resource discovery request, it carries out the two operations presented below:

- it evaluates the request in order to determine if it can fulfil the request with the required QoS. If yes, it answers to n_0.

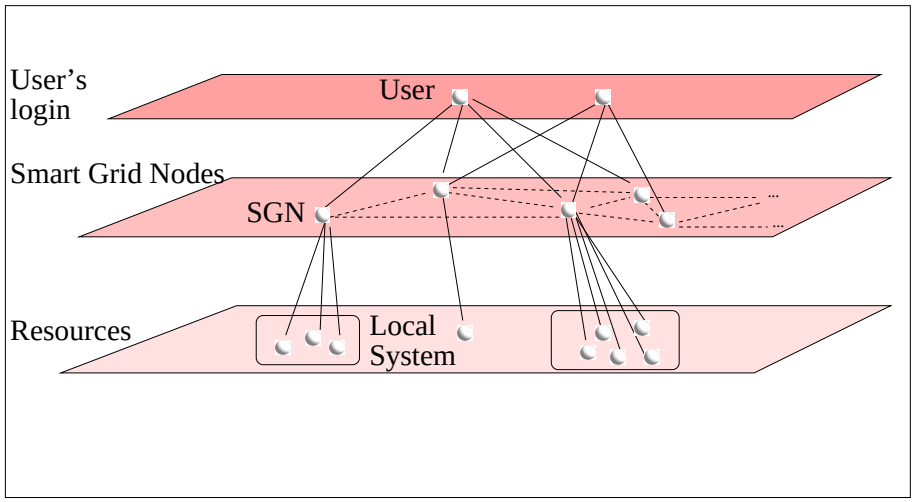

Figure 9: A SGN in its environment

- it adds to the received request a list containing the ID of all its direct neighbours and it forwards the resource discovery request to its direct SGNs that are not already present in the list it received.

The list associated to each request is used to avoid, as much as possible, to contact SGNs that were already contacted for the same request. The Fig. 10 shows the structure of these lists. The resource discovery process is parametrised by two values called *Neighbours Depth* and *Maximum absolute Depth*. The Neighbours Depth has two purposes: to avoid to completely flood the network by the request and to limit the size of the lists associated requests. When Neighbours Depth is reached, i.e. if there are enough discovered resources to fulfill the requested service, the resource discovery process is stopped. If not the resource discovery process continues. In this case the size of lists associated to requests is strictly limited by suppressing from these lists the identity of the oldest neighbours. This heuristics is based on the assumption that the probability for two SGNs of having common neighbours decreases with the distance between the SGNs. The second parameter, the Maximum absolute Depth, is used to limit the maximum propagation depth of requests in the GRID. When this depth is reached the resource discovery process is stopped. Figure 10 illustrates the use of lists associated to requests.

5 Scenario and example

5.1 Detailed scenario

Figure 5 presented in the previous section shows the integration of the ISS-SGN in the context of the UNICORE GRID middleware. This section describes the reference scenario of a job submission using ISS-SGN with a UNICORE/MetaScheduler environment.

The job submission process has been divided into 4 phases :

- Prologue (1-10)
- Decision (11-17)
- Submission (18-19)
- Epilogue (20-23)

Each flow of data is represented by a number and an arrow on Fig. 5. Thus, the reference scenario is presented in detail below.

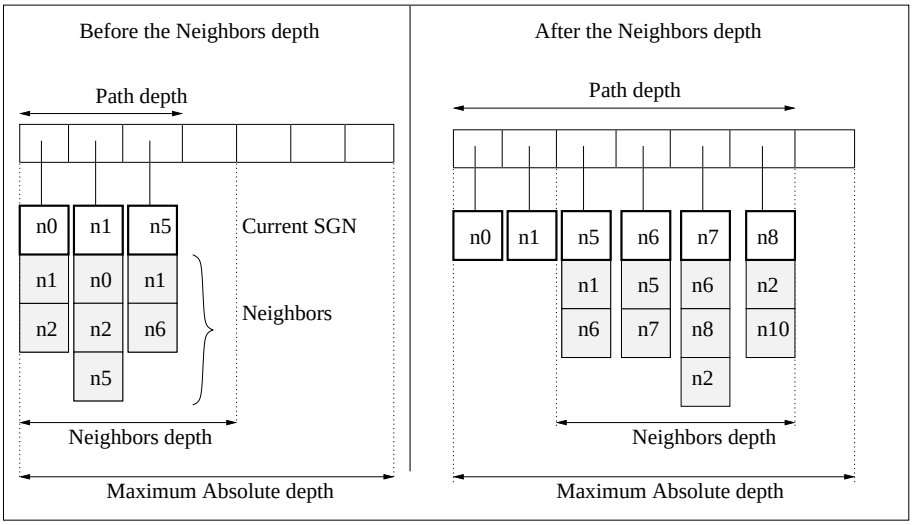

Figure 10: Description of the lists used during the resource discovery process.

1 User defines a workflow in the UNICORE client and associates the requested user QoS

2 UNICORE client submits the workflow to the metascheduler (MSS)

3*a* MSS sends the workflow to the RB

3*b* RB sends the workflow to the SI

4 SI requests information on the requested service from DW (requirements for the requested service)

5 DW sends the collected information to SI

6 SI analyses the collected information from DW and prepares a list of questions to know the availability of the eligible machines, to be answered by MSS. This list contains for each component of the workflow a set of eligible machines of the local system.

7*a* SI sends this list to the RB

$7a_1$ RB starts the resource discovery process presented in section 4.11.

7*b* RB sends the list created in 6 to the MSS

8 MSS prepares the query in order to check the availabilities and access rights of the local system

9 MSS checks for availability and access rights

10 Local systems return information about availabilities of local resources to the MSS

11 MSS sends availability information to the RB

12 RB requests Γ model information from SI

13 SI requests Γ model information from DW

14 DW sends Γ model information to SI (Parameters of available machines, cost of one hour engineer, ...)

15 SI sends Γ model information to RB

16 RB evaluates cost function and prepares a list of cost function values and tolerances. This list contains for each available machine, the number of nodes, the cost, the component, the MSS.

$7a_2$ Remote RBs send cost functions to RB

17 RB merges the list in 16 and the list in $7a_2$ and sends the merged list to MSS

18 MSS reserves the well suited resource

18_1 MSS negotiates with the remote MSS and starts the execution of the selected part of workflow on the remote node

19 During execution : the monitoring module (MM) save on a local database the component relevant data

20 Local system sends results to the UNICORE client

20_1 The remote node sends results to the UNICORE client

21 At the end of component execution, MM computes the data to be stored into the Data WareHouse

22 MM computes the relevant quantities to be sent to SI

23 MM sends to SI the relevant data to be stored into the DW

23_1 Remote RB sends data if needed to the RB in order to be stored

23_2 RB sends received data from remote node to SI in order to be stored

The lists of information to send between the different modules can be found in Appendix C.

5.2 Data flow example: Submission of ORB5

Let us follow the data flow of the real life plasma physics application ORB5 that runs on parallel machines with over 1000 processors. ORB5 is a particle in cell code. The 3D domain is discretized in $N_1 x N_2 x N_3$ mesh cells in which move p charged particles. These particles deposit their charges in the local cells. Maxwell's equation for the electric field is then solved with the charge density distribution as source term. The electric field accelerates the particles during a short time and the process repeats with the new charge density distribution. As a test case, $N_1 = N_2 = 128$, $N_3 = 64$, $p = 2'000'000$, and the number of time steps is $t = 100$. These values form the ORB5 input file.

Two commodity clusters at EPFL form our test Grid, one having 132 single processor nodes interconnected with a full Fast Ethernet switch (Pleiades), the other has 160 two processor nodes interconnected with a Myrinet network (Mizar).

In this example, we consider that we have a GRID containing only two ISS-SGN which have eligible machine for the requested job. The different steps in decision to which machine the ORB5 application is submitted are:

1 User defines a workflow using the ORB5 input file in the UNICORE client

2 UNICORE client submits the workflow to the metascheduler (MSS). This workflow contains the components and the ORB5 input file

$3a$ MSS sends the workflow to the RB

$3b$ RB sends the workflow to the SI

4 SI requests information from DW on ORB5 (requirements for ORB5)

5 DW sends information on ORB5 to SI

6 SI analyses information from DW: it selects the information (memory needed 100 GB) and prepares a list of questions to be answered by MSS. This list contains for each component of the workflow a set of eligible machines. In this case, the eligible machine is Mizar

$7a$ SI sends this list to the RB

$7a_1$ RB starts the resource discovery process. It sends the workflow sent by the UNICORE client to the remote RB

8 MSS prepares the query to Mizar

9 MSS checks for availability and access rights

10 Local systems return information to the MSS:
Mizar: 160 nodes, 4 GB per node, SFr. 2.50 per node*h, 32 nodes job limit, availability table (1 hour for 32 nodes), user is authorised, executable ORB5 exist)

11 MSS sends availability information to the RB

12 RB requests Γ model information from SI for Mizar

13 SI requests Γ model information from DW

14 DW sends Γ model information to SI : $\Gamma = 20$ for Mizar, 1 hour engineering time cost Sfr. 200.-, 8 hours a day

15 SI sends Γ model information to RB

16 RB evaluates cost function and prepares a list of cost function values and tolerances. This list contains for each available machine, the number of nodes, the cost, the component, the MSS. In this case, this list is composed only by information about Mizar (160 nodes, 4 GB per node, cost: SFR 3720.-)

$7a_2$ The remote RB sends a list containing for each available machine, the number of nodes, the cost, the component and the MSS to contact the machine if needed. In this case, this list is composed only by information about Pleiades (132 nodes, 2 GB per node, cost: SFR 3968.-)

17 RB merge the two lists and sends the merge list to MSS

18 MSS reserves and starts the execution on Mizar

18_1 ** no remote machine selected in this example **

19 During execution : the monitoring module (MM) save on a local database the component relevant data

20 Mizar sends results to the UNICORE client

20_1 ** no remote machine selected in this example **

21 At the end of component execution, MM computes the data to be stored into the Data WareHouse

22 MM computes the relevant quantities to be send to SI

23 MM sends to SI the relevant data to be stored into the DW. SI computes Γ model parameters (e.g. $\Gamma = 18.7$, $M = 87$ GB, Computing time=21h 32') and stores them into DW

23_1 ** no remote machine selected in this example **

23_2 ** no remote machine selected in this example **

6 Security aspects

Security is a crucial aspect in distributed systems where the sharing and the access of resources is often regulated by a centralized trusted entity; in peer networks the individual entities have to agree on the level of trust.

Grids can be used to harness computational power, provide access to unified data, or other intensive tasks. From a security viewpoint, a grid represents a high-value target for anyone who would want to gain unauthorized access. Grids need to be protected and secure because they represent a point of access to the resources of the different institutions involved.

From a Grid perspective, the following challenges are raised:

- How to manage heterogeneous environments?

  Without a common agreed and coordinated effort, organizing a multitude of hardware and software configurations owned by different institutions, providing services to multiple communities of users with different needs could become an impossible task, and a reason for a project to fail.

- How to deal with authorization and authentication?

  In a Grid project there are multiple layers of ownership: The network is owned and managed by the organization. Individual machines are also owned by the organization, but for practical purposes, are run by the person assigned to it. Finally, tasks that are run on the Grid are owned by the task originator, but the task has to make its way through the myriad possible authorization scenarios. Each of these layers call for authenticated and authorized access.

There are a number of authorization systems currently available for use on the Grid and they all have similar semantics. These systems give a description of the initiator, a description of an action being requested, details about the target resource to be accessed, and any contextual information such as time of the day, and they provide an authorization decision whether the action should be processed or rejected.

The current implementation of ISS is based on UNICORE that has a security model based on job authentication and secure transmission of data. The security model supports both job signing and data encryption, which protects remote users against data theft and data manipulation.

Relevant for the individual organizations participating in a Grid, UNICORE provides the following functionalities:

- Provision of user authentication mechanism based on X.509 certificates.
- Compatibility to the organization authorization mechanisms and policy; UNICORE IDs are mapped to local Unix user IDs reflecting access policies disk quotas etc.
- Site and system specific incarnation of UNICORE jobs driven by a declarative Incarnation Database that can be adapted to the organization's needs.
- Declarative description of available resources, both traditional capacity resources, like processor count, computation time, memory size, and capability resources, like available software packages and special hardware capabilities.

Additionally ISS has to take into account secure access to organization's resources during the resource discovery algorithm. Traversing organization's firewall to inspect local Data Warehouses of resources sitting in private networks demand of a high level of access that may conflict with the site access policies. Individual sites need to agree on access policies that somehow will be mapped and cope with their own internal policies; as a consequence of that, the discovery algorithm must take site's restrictions into account when trying accessing the site's resources Data Warehouse.

7 Implementation aspects

7.1 VAMOS : An implementation of the Monitoring Module

The **V**eritable **A**pplication **MO**nitoring **S**ervice is an implementation of the Monitoring Module. It has been installed on the Pleiades testbed (Pleiades1, Pleiades2, Pleiades2+ clusters). The model is quite simple : perform a mapping between hardware monitored data (using the Ganglia[13] service) and application relevant data (using the RMS/Local Scheduler Torque/Maui) and store the information in a local database to be reused.

Technical aspects

VAMOS has been written in PHP. It uses XML files to store configuration files. The main class is called every hour through the UNIX tool cron. Ganglia stores its relevant information in a round robin database, keeping information during 2 hours.

Scenario

The scenario for each machine (configuration file see 8.1.2) is the following :

1. get the list of running/submitted/stopped jobs
2. compare it with the list already stored
3. update the database. For each running job in the list :
   - get start time
   - get list of assigned nodes for that job
   - for each node, read information in the Ganglia round robin database from start time to present time
   - store information in the VAMOS database
4. update database. For each finished job in the list :
   - get start time
   - get stop time
   - get list of assigned nodes for that job
   - for each node, read information in the Ganglia round robin database from start time to stop time
   - store information in the VAMOS database
5. clean database from incorrect data

Note that the chosen metric information read from Ganglia (such as CPU utilization, network usage, etc...) is a table in the VAMOS DB. That database can grow rapidly.

Metrics stored in VAMOS

In its present version, VAMOS stores the following information :

- CPU usage (idle, system, user)
- Network usage (packets IN/OUT, bytes IN/OUT)
- Memory usage (Swap usage, memory usage)

What Ganglia monitors is what the Linux kernel (or Windows) provides. All this information is taken from the pseudo file system `/proc`. This is not sufficient to compute the Γ values. We need other quantities (such as MFlops/s rate, Cache misses, etc..) for each component. These quantities can be computed using direct access to hardware counters using PAPI [32, 12] which are accessible directly on Itanium or AMD Opteron based machines or through specific library (Perfctr on Pentium for instance available on Linux, Windows, etc.. OS's).

Results for Pleiades1, Pleiades2 and Pleiades2+ can be found on `http://pleiades.epfl.ch/~vkeller/VAMOS`

8 Simulators

8.1 Cost model simulator : ISS-SIM2

This simulator has been developed at EPFL.

Goals of ISS-SIM2

ISS-SIM2 has been designed to achieve 2 complementary goals :

1. To test different configurations for the cost model. It can test the CFM function weights as well as the functions used in the cost model. The aim is to understand how to tune the cost function model without using real production systems.
2. To predict the future machines to be added in the Grid that best improve the overall Grid performance. While one can parametrize hardware using the Γ model, ISS-SIM2 can add new *imaginary* machines in the Grid.

Hypothesis

ISS-SIM2 assumes that a resource discovery algorithm has been performed. The situation is the following : the middleware has a complete view of the Grid resources. Each resource has its own policy and is accessed by local users as well as by Grid users.

8.1.1 Model

A Grid is a set of r resources accessed by a number Grid clients. An universal Grid clock ensures that every transaction on the Grid is performed respecting an universal time (named *GUT* for **G**rid **U**niversal **T**ime). It exists one broker and one MetaScheduler Service for per simulated Grid.

Each resource of the Grid is a parallel machine with P computing elements of p processors of a given architecture (Intel x386, AMD Socket F, etc..), c cores each. A simple workstation is described as $P = p = 1$ (c varies with the type of processor). The model supposes that each resource has its own local clients, its own VAMOS system, its own local RMS, and its own local Scheduler with its scheduling table. This table keeps the information about the queues of the local resource starting at the Grid installation time and ending with the last submitted job.

Time is divided into seconds. The simulation is perform increasing the *GUT*.

8.1.2 Implementation

ISS-SIM2 has been implemented in Java using the Java Threads mechanisms to simulate the users. It uses XML to describe the resources (see 8.1.2). The first prototype uses its own description schema, future releases should adopt an official and standardized description schema (aka GLUE). Every client (local or Grid) is a Java Thread.

The database is a remote mysql DB accessed by the common Connector/J. Note that VAMOS hourly updates the DB with data from Pleiades1, Pleiades2, and Pleiades2+. It is possible to create an empty DB locally.

XML files are parsed using Xerces.

Machine description

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- This XML document describes the Pleiades2 cluster in real life. -->
<!-- it is the same document used by the VAMOS tool                  -->
<!-- Author : Vincent Keller (Vincent.Keller@epfl.ch)                -->
<config>
        <parallelSystem>
                <system_name>Pleiades 1</system_name>
                <nbrNodes>132</nbrNodes>
                <CPUType>Pentium 4</CPUType>
                <CPUClock>2.8</CPUClock>
                <CPUCPI>2</CPUCPI>
                <CPUWordLength>32</CPUWordLength>
                <CPUArch>i686</CPUArch>
                <CPUPerNode>1</CPUPerNode>
                <CoresPerCPU>1</CoresPerCPU>
                <RamBandwith>800</RamBandwith>
                <RamAmountPerNode>2000</RamAmountPerNode>
                <InterconnectType id="1">FE</InterconnectType>
                <InterconnectTopology>switch</InterconnectTopology>
                <InterconnectLatency>1.9</InterconnectLatency>
                <InterconnectBandwidth>12.5</InterconnectBandwidth>
                <InterconnectAvgDistance>1</InterconnectAvgDistance>
        </parallelSystem>


        <CFMValues>
                <alpha>0.00</alpha>
                <beta>0.00</beta>
                <gamma>0.00</gamma>
        </CFMValues>


        <ganglia>
                <server>pleiades1.epfl.ch</server>
                <path>/var/lib/ganglia/rrds/Pleiades</path>
        </ganglia>


        <scheduler>
                <name>maui</name>
                <server>pleiades.epfl.ch</server>
                <path>/usr/local/src/maui/maui-3.2.6/stats/</path>
        </scheduler>


        <RMS>
```

```
        <name>openpbs</name>
        <server>pleiades.epfl.ch</server>
        <path>/opt/pbs/bin</path>
    </RMS>


    <local_database>
        <server>linpc2.epfl.ch</server>
        <driver>mysql</driver>
        <name>pleiades1_db</name>
        <user>vkeller</user>
        <password>pleiades</password>
    </local_database>


    <QSTAT>
        <name>tmpjob</name>
        <server>pleiades.epfl.ch</server>
        <path>/tmp/runningJobsT_P1.txt</path>
        <pathQ>/tmp/runningJobsQT_P1.txt</pathQ>
    </QSTAT>


    <system_information>
        <server>linpc2.epfl.ch</server>
        <driver>mysql</driver>
        <name>sysinfo_db</name>
        <user>vkeller</user>
        <password>pleiades</password>
    </system_information>



</config>
```

8.2 Resource discovery simulator: SGN-Sim

This simulator has been developped at EIA-Fr.

8.2.1 Implementation

We have developed a simulator in order to test the proposed resource discovery algorithm and to implement a first prototype of Data Warehouse. SGN-Sim is based on the Smart Grid Node reference architecture presented in section 5.1.

The SGN-Sim has been developed in Java. We have used SimJava to simulate a Grid environment which is a process based discrete event simulation. In order to store some information during the simulation, a first Data Warehouse prototype has been realised using MySQL. Figure 11 presents the architecture of the simulator.

8.2.2 Simulator input files

- User request: The user request is described using an XML format as shown below.

```
<SGNRequest idType="user">
    <Service name="add">
     <Options>
          <Option type="library">
                opt1
           </Option>
           <Option type="compilationParameters">
                opt2
           </Option>
      </Options>
      <Parameters>
       <Parameter>
                 1
       </Parameter>
       <Parameter>
                 2
      </Parameter>
     </Parameters>
     </Service>
</SGNRequest>
```

- Grid topologies: We have developed a Grid topology generator in order to generate automatically different Grid topologies. These topologies are represented using XML. We present below an example of XML Grid topologies.

```
<GRID>
      <NODE>
            <INFO>
                <NAME> m5.eif.ch</NAME>
                <PORT> 5555</PORT>
            </INFO>
            <SERVICES>
                <SERVICE>
                        <NAME>mul</NAME>
                </SERVICE>
                <SERVICE>
                        <NAME>add</NAME>
                </SERVICE>
             </SERVICES>
             <NEIGHBORS>
                <NEIGHBOR>
                        <INFO> ...</INFO>
                        <SERVICES> ... </SERVICES>
                </NEIGHBOR>
      </NODE>
...
```

Remark: We generate a graphical output file in order to visualize the resource discovery process. This output file is written using the dotty format in order to check the resource discovery algorithm.

8.2.3 First results

The performance of SGN-Sim resource discovery algorithm, described in section 4.11, has been analysed in order to assess its effectiveness in a Grid environment. We have simulated a Grid network composed by

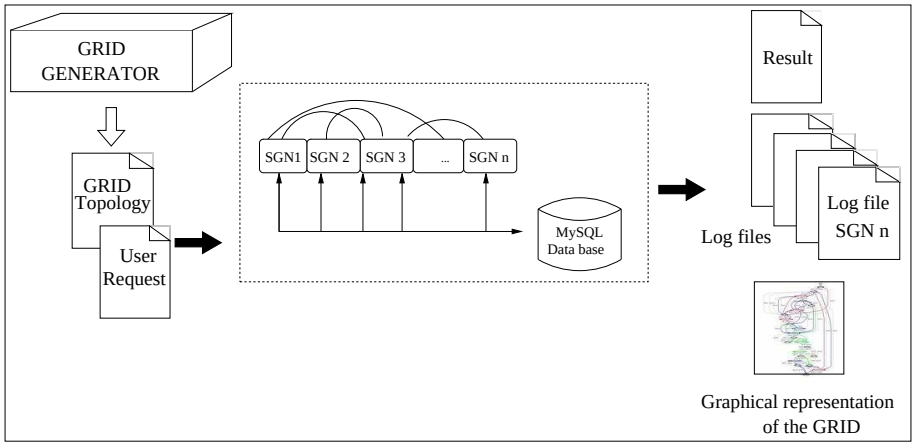

Figure 11: Architecture of SGN-Sim

Smart Grid Nodes. We increased the number of Grid nodes from 100 to 2000 in order to analyse the number of requests sent during the resource discovery process. Results shown in Figure 13 are obtained with a 3D torus topology and random Grid topology (Figure 12). Each node of the random Grid topology has an average degree fixed to 6. We fixed the Neighbours Depth to 1′000′000 in order to visit all the nodes of the Grid. It appears that the number of requests sent during the resource discovery process is very high. Several nodes are visited several times and the number of requests sent is almost similar to the number of requests sent using a simple broadcast (a node sends a request to all its neighbours excepted to the neighbour which has sent to it the same request it wants to propagate).

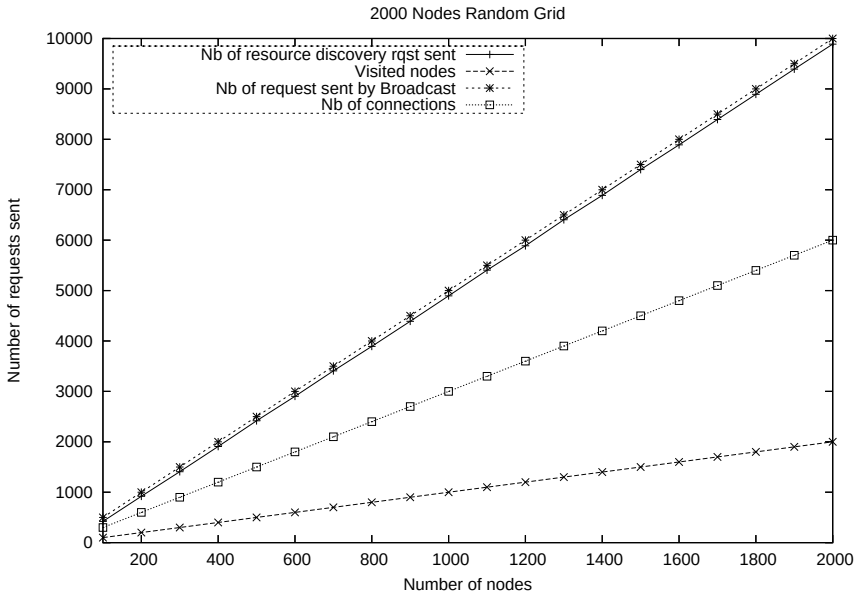

Figure 12: Number of requests sent during the resource discovery process with a randomGrid

These results can be explained by the presence of cycles into the Grid topologies. Indeed, during a resource discovery process, Fig. 14 and Fig 15 present the results obtained avoiding cycles (the presence of cycles is stored into the DW, according to this information the request propagation is managed in order to avoid entering into cycles). It appears that the number of requests sent is very low and all nodes are visited

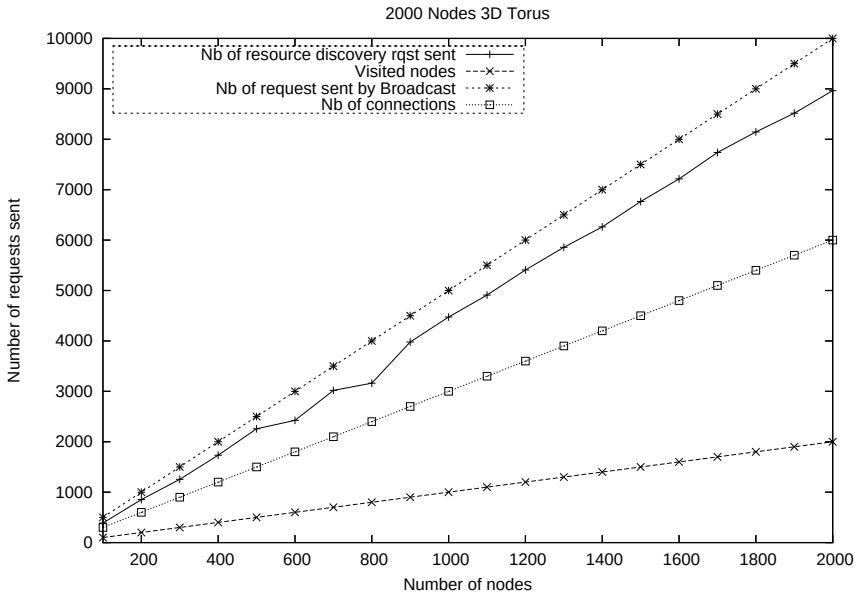

Figure 13: Number of requests sent during the resource discovery process with a Torus 3D

(almost) only one time.

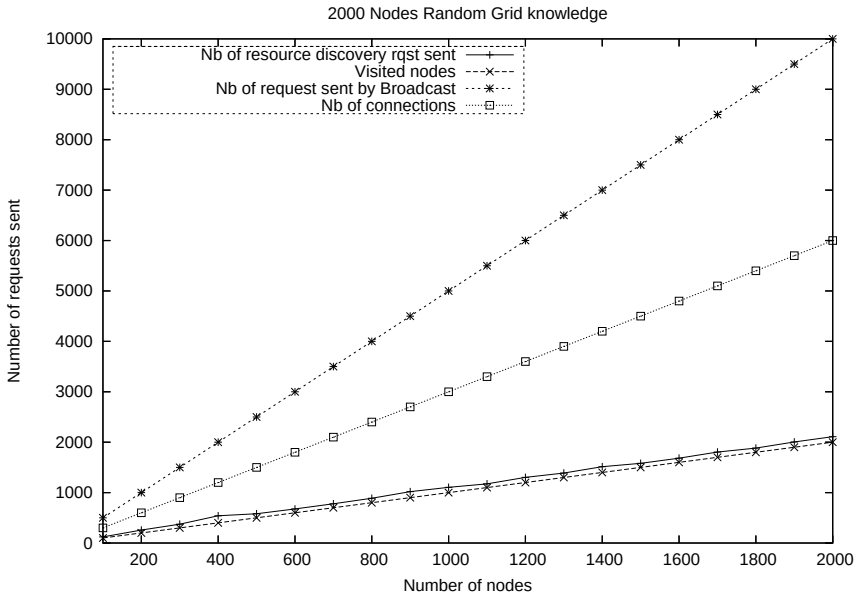

Figure 14: Number of requests sent during the resource discovery process with a randomGrid avoiding cycles

9 First testbed at EPFL

Three departments are involved in ISS within the EPFL : the Central IT Domain (DIT), the Engineering Faculty through the ISE (Institut des Sciences de l'Énergie) and the Plasma Physics Research Center

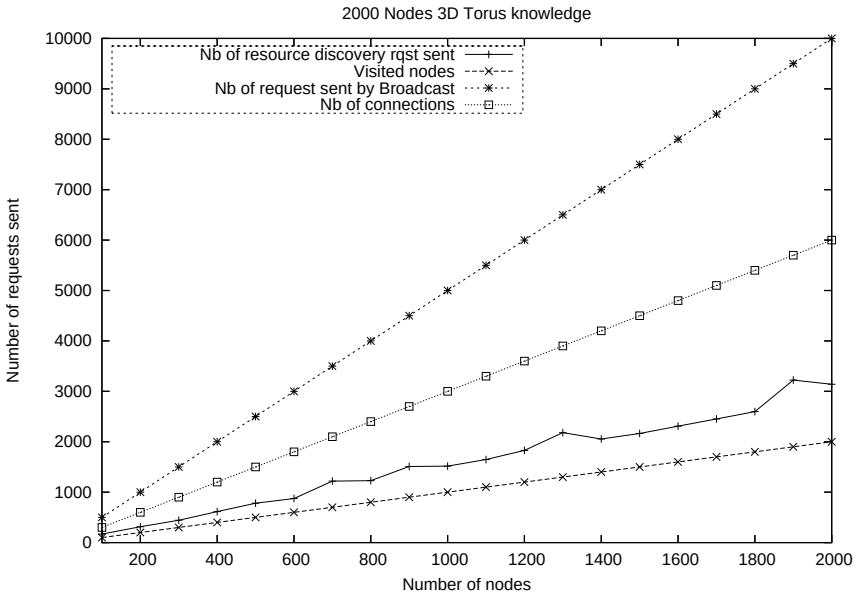

Figure 15: Number of requests sent during the resource discovery process with a Torus 3D avoiding cycles

Machine	*Type*	*Node type*	*# Nodes*	*# CPU / # core per node*	*Network*	*Loc.*
Pleiades1	Cluster	P4 HT	132	1/1	FE Switch	SGM
Pleiades2	Cluster	XEON	120	1/1	GbE Switch	SGM
Pleiades2+	Cluster	Woodcrest	92	2/2	GbE Switch	SGM
LINPC's	NoW	heterogeneous	32	1/1	FE bus	SGM
Mizar	Cluster	Opteron	448	2/1	Myrinet	DIT
Mizar	NUMA	Itanium	16	16/1	NUMA	DIT
Alcor	Cluster	Woodcrest	24	2/2	Myrinet	DIT
Greedy	NoW	heterogeneous	250	-	FE bus	DIT

Table 4: First EPFL testbed machines

(CRPP). The first testbed will integrate several machines (Table 4) and specialized applications (Table 5) from Mechanics, Fluid Dynamics and Plasma Physics. The alpha users will come from CSCS, EPFL and EIA-Fr.

The machines are interconnected through the Fast Ethernet campus network.

The UNICORE/Metascheduler environment has been installed on the testbed. A first alpha version of ISS should be ready by July 2007 and the beta version by the end of 2007.

10 The CoreGRID cooperation

The ISS implementation in UNICORE/MetaScheduler is part of the SwissGRID initiative and realised in a co-operation between CoreGRID partners. It is planned to install the UNICORE/MetaScheduler/ISS middleware by the end of 2007 to guide job submission to all HPC machines in Switzerland.

Within CoreGRID, the integration of ISS into UNICORE is a collaboration between 8 institutions, the first 6 are CoreGRID partners:

1. École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne (Switzerland)

Application	*Type*	*Area*
SpecuLOOS	Point-to-Point communication dominated	CFD
ORB5	Multicast communication dominated	Plasma Physics
OpenMP Helmholtz Solver	Multicast communication dominated	CFD

Table 5: First EPFL applications test set

2. École d'Ingénieurs et d'Architectes, CH-1075 Fribourg (Switzerland)
3. ForschungsZentrum Jülich, D-52425 Jülich (Germany)
4. Fraunhofer Gesellschaft SCAI Institut, D-53754 St. Augustin (Germany)
5. University of Dortmund, D-44221 Dortmund, Germany
6. CETIC, B-6041 Charleroi, Belgium
7. Swiss National Supercomputing Center, CH-6928 Manno (Switzerland)
8. Switch, CH-8021 Zurich, Switzerland

Acknowledgements

ISS is a Swiss project within the Swiss Grid Initiative managed by the Swiss National Supercomputing Center CSCS. CoreGRID is an European Network of Excellence (NoE) funded by the European Commission's IST programme under grant #004265. Thanks go to Michel Reymond (EPFL) for his help in validating the CPU cost model.

A Definitions and restrictions

In this section, we define the terms used in this technical report.

A.1 Definitions

- 1 Site = an entity managed by one MetaScheduling Service.
  In the UNICORE model,
  - 1 Site = 1 USite.

  In the ISS project,
  - 1 Site = 1 SGN.
- 1 Machine = an entity managed by one 1 Scheduler.
  In the UNICORE model,
  - 1 Machine = 1 VSite.
- 1 Node = one entity having one Uniform Memory Access (UMA)
- 1 Processing element = L2 cache (in reference to the Intel's Woodcrest architecture)
- 1 Core = L1 cache (in reference to the Intel's Woodcrest architecture)
- 1 application = a set of components (one or more)

A.2 Restrictions

We consider that :

- 1 Component is executed on, at most, one machine
- 1 distributed process is executed on, at most, one node
- 1 parallel process is executed on, at most, one core

In the ISS project, an application is characterized by:

- a number of components n
- number of processors p_k needed by C_k
- a workflow (C_k)
- memory size
- type of application
- number of Thread T_k for each p_k

B Requests between modules

UNICORE client → MSS (2)

- $(\text{Workflow})_k$
- $(\text{Constraints})_k$

MSS → RB → SI (3)

- $(\text{Workflow})_k$
- $(\text{Constraints})_k$

RB → Remote RB (7a_1)

- $(\text{Workflow})_k$
- $(\text{Constraints})_k$
- Information about the RB (how to contact RB from remote RB)

SI → RB → MSS (7)

- $(C_k)_i$
- $(p_k)_i$
- $(T_k)_i$
- M_k
- Software requirements

MSS → Local Systems (8, 9)

- User known?
- Software exists?
- Hardware properties sufficient?

- Component constraints
- Availabilities $(p_k$, input queues$)_i$

LS → MSS (10)

- (User rights)$_i$
- Availabilities $(p_k$, input queues$)_i$

MSS → RB (11)

- List of eligible machines with
- Availabilities $(p_k)_i$

SI → RB (15)

- $\Gamma(C_k, p_k)_i$

Remote RB → RB (7a_2)

- all machines $(C_k, p_k, z)_i$ such that
- $z_{min} < z < z_{min} + tol$

RB → MSS (17)

- all machines $(C_k, p_k, z)_i$ such that
- $z_{min} < z < z_{min} + tol$

LS (by MM) → SI (23)

- Monitored data on machine i about C_k

Glossary

A	Speedup	3
α_k	Free parameter in CFM (CPU costs and license fees)	5
β	Free parameter in CFM (Turn-around time)	5
application	HPC program consisting of $k = 1, .., n$ components	1
b	Effective per node internode communication bandwidth	3
C_k	kth application component	2, 3, 5, 6
C_∞	Peak communication network bandwidth of a machine	2
Γ	CPU time/communication time	2–4, 6
γ_a	O/S	3, 4
γ_k	Free parameter in CFM (Data transfer costs)	5
γ_m	r_a/b	3, 4
d	Number of hours per year = 8760	6
CFM	Cost Function Model	2
component	part of an application	1
Components based applications	Application described by a workflow	5

δ_k	Free parameter in CFM (Energy costs)	5
DW	Data WareHouse	12
DW	Data Warehouse, part of SGN	2
e	Efficiency	3
e_i	Efficiency of machine i	6
EGEE	CERN's Grid middleware development project	1
Embarrassingly parallel applications	Application which do not demand inter-node communications ($\Gamma >> 1$)	4
Gflops/s	10^9 double precision floating point operations per second	2
Globus	Grid management middleware	1
Grid	Set of resources	1
GridLab	A EU Grid middleware development project	1
Gwords/s	10^9 double precision words per second	2
HPC	High Performance Computing	1
i	Index for machine	2, 5
ISS	Intelligent Grid Scheduling Service	1
ISS-SIM2	ISS Simulator used to test the cost model	22
K_{MAX}	Maximum costs	5
k	Index for application component	2, 5
k_e	CPU cost function	6
K_d	Data transfer costs	5
K_e	CPU costs	5, 6
K_{eco}	Energy costs	5
K_l	Licence fees	5
K_w	Costs due to turn-around (waiting) time	5
M_k	Estimated memory size of (C_k, p_k)	30
M_∞	Peak memory bandwidth of a node	2
machine	Cluster or SMP managed by one RMS	6
MS	Monitoring System	2
MSS	MetaScheduling Service	11
MSS	VIOLA MetaScheduling Service	1
Multicast applications	Application where γ_a decreases when problem size grows	4
n	Number of components in an application	2, 5
node	Reservable computational unit, can be one processor, a NUMA, or a SMP	2
NoW	Network of Workstations	1, 2
NUMA	Virtual shared memory machine	1
O	Total number of operations performed in one node	3, 4

OpenMP	OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs.	4
OpenMP applications	Application which demand an SMP node	4
ORB5	Single component plasma physics application	2
P	Number of nodes of a machine (index i left out)	2
P_i	Number of nodes of machine i	5
φ	Priority	6
p_k	Number of processors needed by C_k	3, 5, 6
Pleiades Clusters	The Pleiades clusters are located in the Mechanics Department at EPFL / Switzerland. Find a description on http://pleiades.epfl.ch	9
Point-to-point applications	Application where γ_a keep a constant value when problem size grows	4
R_∞	Peak performance of a node	2–4
R_i	Resource on machine i	5, 6
r	Total number of machines in a Grid	2
r_a	Nodal peak performance of an application	3, 4
r_i	Parameter to normalise time	6
RB	Resource Broker, part of Smart Grid Node	2
RB	Resource Broker	12
RDS	Resource Discovery System, part of SGN	2
resource	Series of nodes demanded by C_k	2
RMS	Resource management System	6
S	Total number of words sent through the network by one node	3
S_a^i	Yearly management costs for machine i	6
S_b^i	Yearly interests to be paid to the bank for machine i	6
S_c^i	Initial investment for machine i	6
S_f^i	Yearly insurance fees for machine i	6
S_g^i	Yearly profit with machine i	6
SGN	Smart Grid Node	2, 16
SI	System Information, part of SGN	2
SI	System Information	13
S_I^i	Yearly infrastructure costs for machine i	6
S_m^i	Free parameter in CFM (Yearly maintenance fee for machine i)	6
SMP	Shared memory machine	1
S_p^i	Yearly personal costs for machine i	6
T_k	Estimated execution time of $(C_k, p_k)_i$	30
T	T_C+T_P	3
T^i	Life time of machine i	6
T_C	Internode communication time	3
T_P	Computing time on p nodes	3
T_{MAX}	Maximum turn-around time	5
t_k^0	Time of C_k job submission	5, 6

$t^d_{k,i}$	Time of C_k results available	5, 6
$t^e_{k,i}$	Time of C_k execution end	5, 6
$t^s_{k,i}$	Time of C_k execution start	5, 6
tol	Tolerance value added to a cost for a chosen machine	12
UNICORE	Grid middleware	1
V_C	R_∞/C_∞	2
VIOLA	Vertically Integrated Optical Testbed for Large Applications in DFN	1
V_M	R_∞/M_∞	2
workflow	Work described as a DAG or non-DAG where each leaf is component	11
y_i	Half value time of machine i	6

References

[1] Sergio Maffioletti. Grid generic architecture. Please contact author: sergio.maffioletti@cscs.ch.

[2] Keller, V., Cristiano, K., Gruber, R., Spada, M., Tran, T.-M., Kuonen, P., Wieder, P., Ziegler, W., Maffioletti, S., Nellari, N., Sawley, M.-C.: Integration of ISS into the VIOLA Meta-Scheduler environment. TR CoreGRID 25, 2005.

[3] Gruber, R., Volgers, P., De Vita, A., Stengel, M., Tran, T.-M.: Parameterisation to tailor commodity clusters to applications. *Future Generation Computer Systems*, **19**:111–120, 2003.

[4] Gruber, R., Keller, V., Kuonen, P., Sawley, M.-C., Schaeli, B., Tolou, A., Torruella, M., and Tran, T.-M., Towards an Intelligent Grid Scheduling System, Proc. of 6th Int. Conf. PPAM 2005, Poznan, Poland, Lecture Notes in Computer Science 3911 (Springer, 2006) 751-757

[5] Erwin, D., UNICORE plus final report – uniform interface to computing resource,Forschungszentrum Jülich, 2003, ISBN 3-00-011592-7

[6] Wäldrich, O., Wieder, P., and Ziegler, W., A Meta-Scheduling Service for Co-allocating Arbitrary Types of Resource, In *Proc. of Conference on Parallel Processing and Applied Mathematics PPAM 2005*, Poznan, Poland, 2005, to appear.

[7] Keller,V., VAMOS web frontend to the Pleiades clusters, http://pleiades.epfl.ch/ vkeller/VAMOS

[8] Dubois-Pèlerin, Y., Van Kemenade, V., and Deville, M., An object-oriented toolbox for spectral element analysis, J. Sci. Comput. 14 (1999) 1-29

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

[10] Gruber, R., Keller, V., Leriche, E., and Habisreutinger, M.A., Can a Helmholtz solver run on a cluster?, accepted to appear in Procs. of Cluster 2006

[11] Manneback, P., Bergére, G., Emad, N., Gruber, R., Keller, V., Kuonen, P., Noël, S., and Petiton, S., Proposal of a scheduling policy for hybrid methods on computationsl Grids, CoreGRID workshop (Pisa, 2005)

[12] Dongarra, J., London, K., Moore, S., Mucci, P., and Terpstra, D., Using PAPI for hardware performance monitoring on Linux systems, www.netlib.org/utk/people/JackDongarra/PAPERS/papi-linux.pdf

[13] http://ganglia.sourceforge.net/

[14] Andreozzi, S., Burke, S., Field, L., Fisher, S., Konya, B., Mambelli, M., Schopf, J.M., Viljoen, M., Wilson, A., GLUE Schema Specification version 1.2, Final Specification. December 2005,

[15] Cristiano, K., Kuonen, P., Smart Grid Node : Un nœud intelligent pour la grille, accepted to appear in Renpar'06, Perpignan, 2006.

[16] Gruber, R., Keller, V., Thiémard, M., Wäldrich, O., Wieder, P., Ziegler, W., and Manneback, P., Integration of Grid Cost Model into ISS/VIOLA Meta-Scheduler environment, accepted to appear in Proc. of UNICORE Summit (Dresden, 2006).

[17] UNICORE Open Source – Download, http://www.unicore.eu/download/unicore5/.

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

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

[20] Seidel, E., Allen, G., Merzky A., and Nabrzyski, J., GridLab–a grid application toolkit and testbed, Future Generation Computer Systems, Volume 18, Issue 8, October 2002, Pages 1143-1153.

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

[22] R. Gruber, V. Keller, M. Thiémard, O. Wäldrich, W. Ziegler, P. Wieder, P. Manneback, P. Kuonen, K. Cristiano, P. Kunstz, S. Maffioletti, M.C. Sawley, ISS concept, CoreGRID Integration workshop, Crakow, 2006

[23] http://pleiades.epfl.ch/

[24] The EUROGRID project, website, 2005. Online: http://www.eurogrid.org/.

[25] The UniGrids Project, website, 2005. Online: http://www.unigrids.org/.

[26] The National Research Grid Initiative (NaReGI), website, 2005. Online: http://www.naregi.org/index_e.html

[27] VIOLA – Vertically Integrated Optical Testbed for Large Application in DFN, website, 2005. Online: http://www.viola-testbed.de/.

[28] Streit, A., Erwin, D., Lippert, Th., Mallmann, D., Menday, R.,Rambadt, M., Riedel, M., Romberg, M., Schuller, B., and Wieder, Ph., UNICORE - From Project Results to Production Grids, L. Grandinetti (ed.), *Grid Computing and New Frontiers of High Performance Processing*, Elsevier, 2005, to be published. Pre-print available at: http://arxiv.org/pdf/cs.DC/0502090.

[29] G. Quecke and W. Ziegler, MeSch – An Approach to Resource Management in a Distributed Environment, In *Proc. of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000).* Volume 1971 of Lecture Notes in Computer Science, pages 47-54, Springer, 2000.

[30] Streit, A., Wäldrich, O., Wieder, Ph., and Ziegler, W., On Scheduling in UNICORE - Extending the Web Services Agreement based Resource Management Framework, In *Proc. of Parallel Computing 2005 (ParCo2005)*, Malaga, Spain, 2005, to appear.

[31] Andrieux,A. et. al., Web Services Agreement Specification, July, 2005. Online: https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/16.

[32] Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P. , A Portable Programming Interface for Performance Evaluation on Modern Processors , The International Journal of High Performance Computing Applications, Volume 14, number 3, pp. 189-204, Fall 2000, online : http://icl.cs.utk.edu/publications/pub-papers/2000/papi-journal-final.pdf