Computer Science Research and Development manuscript No.
(will be inserted by the editor)

Models and Internals of the IANOS Resource Broker

Vincent Keller[1] · **Hassan Rasheed**[1] · **Oliver Wäldrich**[1] · **Wolfgang Ziegler**[1] · **Ralf Gruber**[2] · **Marie-Christine Sawley**[3] · **Philipp Wieder**[4]

Received: date / Accepted: date

Abstract We present the Intelligent Application Oriented System (IANOS) resource broker models and internals. The aim of IANOS is to provide an understanding of and a solution to the problem of how to find the *best* resource (*best* in the sense "at a given moment and for the requirements of that specific application") for a given submitted application in order to better use a set of HPC resources (an HPCN Grid). The heart of IANOS is a cost model that minimizes the overall submission cost such as execution time cost, waiting time cost, licenses cost, energy cost, etc.. To estimate the execution time cost, a model that predicts the performance of an application on a resource is provided. This model is based on a parameterization of the application and the resources. IANOS has been deployed and tested successfully on an international testbed across Switzerland and Germany.

Keywords HPCN Application · Resource Brokering · Cost Model · Grid Efficiency

[1]FhG SCAI, Schloss Birlinghoven,
D-53754 Sankt-Augustin, Germany
E-mail: Vincent.Keller@scai.fraunhofer.de

[2]EPFL - Laboratory of Computational Engineering
CH-1015 Lausanne, Switzerland
E-mail: Ralf.Gruber@epfl.ch

[3]PH Departement/32/3-026
CERN
CH-1211 Geneve 23, Switzerland
E-mail: sawley@cern.ch

[4]TU Dortmund - ITMC
August-Schmidt Strasse 12
D-44227 Dortmund, Germany
E-mail: philipp.wieder@edu.udo

1 Introduction

*** I do not understand the message of this section fully. I agree to the initial statement. Then, you take Swiss resources as an example and present numbers partly far below 100% while you mention the resources being overbooked? ***

Harvesting the statistics of utilization of the HPC centers is often impossible. These critical (sometimes strategical) data are jealously kept from public view. Let us take a regional example with Switzerland where the ratio [$GFlops$] per capita is one of the highest in the world, the various Swiss high performance resources are over-booked. The Swiss National Supercomputing Center (CSCS) reports that the average usage of their four biggest resources in 2006 is over 60 % [3]. At EPFL (Swiss Federal Institute of Technology in Lausanne), the DIT (central computing center) reports usage of more than 50 % for 2007 on their three biggest resources [4,5] while the PLEIADES clusters (three departmental-size clusters) reports usage of more than 80 % for the year 2007. See Table 1.

One can assume that the situation is quite similar in the other world-wide HPC centers.

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 resource to run it. On the other hand, when he runs an embarrassingly parallel application on a costly NUMA architecture, he will probably not recognize this as a problem.

Machine type	#Nodes	location	Annual usage *in* [%]
NEC SX-5	16	CSCS	62.75
IBM SP-4	256	CSCS	65.74
Cray XT3	1664	CSCS	88.00
IBM p690	768	CSCS	55.79
Dalco Dual-Opteron	448	EPFL-DIT	79.93
SGI Altix	16	EPFL-DIT	48.97
IBM BlueGene/L	8192	EPFL-DIT	75.40
Dalco Woodcrest	24	EPFL-DIT	53.17
Dell Xeon 3000 Serie	120	EPFL-GM	92.00
Dell Xeon 5150 Serie	99	EPFL-GM	92.00
Logics Pentium IV	132	EPFL-GM	83.00

Table 1 Usage of the resources in several HPC sites in Switzerland for the year 2007 (2006 for CSCS). The Swiss National Supercomputing Center CSCS, the Computing Center at EPFL-DIT and the departmental-size clusters Pleiades at Institute of Mechanical Engineering at EPFL

The groundbreaking idea of IANOS is to offer a tool that can be used to (1) better schedule the HPC application submissions on the expensive HPC resources, (2) use the HPC resources on the needs of the HPC applications, (3) help users or developers to improve their application, (4) a tool to forecast the future optimal resources' type based on the current and future applications needs and finally (5) reduce power consumption.

The paper is organized as follows. Section 2 presents the parameterization of both the applications and the resources. These parameterization is mandatory for the heart of the algorithm used by the IANOS resource broker and presented in Section 3. To predict the execution time cost, we present in Section 4 a performance prediction tool and an execution time evaluation model that takes into account the relevant input parameters of a given application as well as the characteristics of the target architecture. First results are presented in Section 5 while Section 6 acts as a conclusion and a future work survey.

2 Parameterization

In [2] and [7], the authors present a parameterization of the resources and the application. They introduce two metrics (Γ and κ) that give a value for the fitting of the needs of an application in regards to what a resource offers. A short survey of this model follows.

The parameterization of the applications is generic. It is possible to describe a components-based application, each component could be parallel or not. The SpecuLOOS test case is a components-based application where the number of components is equal to one.

2.1 Parameterization of an application

It is supposed that a parallel application component C_k is well equilibrated, i.e. each parallel task takes the same computation and communication times during the execution. In this case, the component C_k can be characterized by:

- O: Number of operations [Flop]
- W: Number of main memory accesses [words]
- Z: Number of messages sent over the communication network
- S: Number of words sent over the communication network
- r_a: Peak node processor performance in [GFlops]
- R_a: Real node processor performance in [GFlops]
- V_a: Number of operations per word in cache [Flop/word]
- γ_a: Number of operations per word sent over the network in [Flop/word]
- n_{sd}: Number of parallel tasks (often subdomains) into which the component can be decomposed

One *Flop* (*Flop*) is a word long operation (such as an `ADD` or a `MUL`). A *word* (*word*) can be defined as 64 bits (8 Bytes). This implies that all operations are performed in double precision. The parameter V_a [Flop/word] can be defined as [2]

$$V_a = \frac{O}{W}\,, \tag{1}$$

and γ_a [Flop/word] by

$$\gamma_a = \frac{O}{S}\,. \tag{2}$$

These two parameters V_a and γ_a are critical to main memory access and network communication, respectively. In fact, the programmer of the application component should maximize V_a and γ_a. The bigger V_a the closer to peak performance runs the processor, whereas small values of V_a characterize components that are dominated by main memory bandwidth. The bigger γ_a, the less critical communication. The latency of the interconnection network can play an important role in the performance [7]. This implies that Z should be as small as possible, and message size S should be as large as possible.

The quantity n_{sd} is often defined by a domain decomposition technique in which the geometry is cut into subdomains such that each subdomain can be discretized by a structured mesh. Structured meshes show best node performance efficiencies.

2.2 Parameterization of a resource

An HPC resource r_i is considered as a homogeneous set of P computational nodes P_i, each connected with the others with at least one interconnection network.

A computational node P_i is characterized by:

- N_{CPU}: Number of processors per node
- N_{core}: Number of cores per processor
- m_{node}: Main memory size per node in number of words
- R_∞: Peak processor performance of a node [GFlops]
- g_p: Efficiency of core: Relation between real to peak processor performance
- M_∞: Peak main memory bandwidth of the node [Gwords/s]
- M_M: Real main memory bandwidth [Gwords/s]
- g_M: Efficiency of the motherboard: Relation between real to peak main memory bandwidth

The measured main memory bandwidth

$$M_M = g_M\, M_\infty \tag{3}$$

can differ quite substantially from the peak memory bandwidth M_∞ and can vary in NUMA-like computational nodes. If main memory is local M_M is larger than if data resides on an other processor. The parameter g_M depends on the maturity of the motherboard and on the compiler.

The quantity

$$V_M = \frac{R_\infty}{M_\infty}. \tag{4}$$

is the maximum number of operations a node can achieve during one **LOAD** or one **STORE** (moving one operand from main memory to the lowest level cache or back). Typically, V_M is between 5 and 12 for RISC processors.

The peak performance per node of an application component can be characterized by

$$r_a = \min(R_\infty, M_\infty V_a) = R_\infty \min(1, \frac{V_a}{V_M})\,. \tag{5}$$

Here, the quantity r_a measures the so-called peak performance, i.e., the performance an application component can never reach. If r_a is memory access dominated, i.e. if

$$V_a < \frac{V_M}{g_M} \tag{6}$$

the real performance R_a of the component is given by

$$R_a = g_M\, r_a\,. \tag{7}$$

It was found that g_M generally drops with the number of cores. This can be due to the maturities of the motherboards and the compilers. If

$$V_a > \frac{V_M}{g_M} \tag{8}$$

the measured performance is given by

$$R_a = g_p R_\infty\,, \tag{9}$$

where g_p is independent of M_∞ and N_{core}, but depends on the implementation of the algorithm. This quantity can vary quite substantially. For a LINPACK/LAPACK benchmark that is dominated by DGEMM operations, $g_p > 0.7$, but if the same matrix-matrix operation is Fortran or C++ compiled, $g_p < 0.25$.

Let us assume that a resource r_i is a homogeneous machine, for instance, a cluster made of nodes with the same characteristics, as SMP machines, NUMA machines, workstations, PS3, or even laptops.

A **Grid resource** (or simply **resource**) r_i can be characterized by:

- P: Number of nodes connected to the network
- R: Total peak performance of the resource r_i [GFlops]
- M: Total memory of the resource r_i [Gwords]
- C_∞: Maximum bandwidth of one link [Gwords/s]
- ℓ: Total number of network links
- b_∞: Maximum achievable network bandwidth per node [Gwords/s]
- V_c: R_∞/b_∞ [Flop/word]
- B: Message size such that transfer time = latency time [words]
- C: Total peak intercommunication network(s) communication bandwidth [Gwords/s]
- L: Latency of the network [ns]
- $\langle d\rangle$: Average distance between nodes in the interconnection network
- γ_M: Maximum number of operations possible during one data transfer [Flop/word]

Then,

$$\begin{aligned} R &= PR_\infty \\ M &= Pm_{node} \\ C &= \ell C_\infty\,. \end{aligned} \tag{10}$$

One can in addition define

$$b_\infty = \frac{C}{P\langle d\rangle} = \frac{\ell C_\infty}{P\langle d\rangle} \tag{11}$$

to be the average network communication bandwidth per node [Gwords/s], by

$$V_c = \frac{R_\infty}{b_\infty} \tag{12}$$

the number of operations the processor can maximally perform during the time needed to send one operand from one node to another node [Flop/word], and

$$B = b_\infty L\,. \tag{13}$$

This last parameter B in [words] is the message size that takes one latency time to be transferred from one node to another node.

Also per node, we can now compute the CPU time t_{comp}, the time t_b for network communication, and the latency time t_L, all in seconds [s] by

$$\begin{aligned} t_{comp} &= \frac{O}{R_a} \\ t_b &= \frac{S}{b_\infty} \\ t_L &= LZ\,. \end{aligned} \tag{14}$$

Since we suppose that the parallel tasks of the component are well equilibrated,

$$T_{exec} = t_{comp} + t_b + t_L = t_{comp} + t_{comm} = t_e - t_s \tag{15}$$

T_{exec} corresponds to the time between the start of execution, t_s, and the end of execution, t_e, and t_{comm} is the total network communication time. Here, we suppose that no overlap between computation and communication is possible. This clearly overestimates the total execution time.

The quantity

$$\gamma_M = \frac{R_a}{b_a} \tag{16}$$

expresses the number of operations per word sent over the network for an application. The IT literature commonly defines the communication time as

$$t_{comm} = \frac{S}{b_\infty} + LZ = \frac{S}{b_a} \tag{17}$$

where

$$b_a = \frac{Sb_\infty}{S + LZb_\infty} \tag{18}$$

is the real application network bandwidth. $b_a = b_a(\sigma)$ with $\sigma = \frac{S}{Z}$ includes the latency.

we can now introduce the quantity

$$\Gamma = \frac{\gamma_a}{\gamma_M}\,. \tag{19}$$

This quantity is at the origin of the so-called Γ-model [2]: Γ is 1 if the computation and the communication take the same time, $\Gamma = \infty$ if there is no communication (i.e. $S = Z = 0$), and $\Gamma = 0$ if there is no computation.

2.3 The $\Gamma - \kappa$ model

It is now possible to present the matching rule: the Γ – κ model. This model extends the capabilities of the Γ model [2,7] where:

$$\Gamma = \frac{\gamma_a}{\gamma_M}\,. \tag{20}$$

We remind that the quantity γ_a expresses the *interconnection needs of the application* and γ_M defines *what the network can give.*

On the other hand, we define κ

$$\kappa = \frac{V_a}{V_M}\,, \tag{21}$$

where the quantity V_a expresses the *main memory bandwidth needs of the application* and V_M defines the *maximum node memory bandwidth.*

We claim that using Eqs. [20, 21], it is possible to express the fitness of the resources to the application needs. Table 2 summarizes the $\Gamma - \kappa$ model. One can easily see that given Γ, it is also possible to determine if the application is parallel or not (or embarrassingly parallel); given κ, if the application needs a high memory bandwidth or a high peak performance processor node.

*** I most probably miss something but the text in each line of the table is identical while the formulas are referring to the two complementary parts of a set ***

	$\Gamma \leq 1$	$\Gamma > 1$
$\kappa \leq 1$	The node performance is bounded by the peak memory bandwidth M_∞. $t_{comm} \geq t_{comp}$.	The node performance is bounded by the peak memory bandwidth M_∞. $t_{comm} << t_{comp}$
$\kappa > 1$	The node performance is bounded by the peak CPU performance R_∞. $t_{comm} \geq t_{comp}$.	The node performance is bounded by the peak CPU performance R_∞. $t_{comm} << t_{comp}$.

Table 2 Decision table for the application's fitness based on their needs. We search resources with $\kappa > 1$ and $\Gamma > 1$

3 Cost Model

The heart of the IANOS Resource Broker is an objective function based on one criterion which is the overall cost of a submission of an application. This cost includes computing cost on the target resource. It also includes licenses cost, data transfer cost, ecological cost and waiting time cost. The brokering algorithm tries to

minimize the objective function.

The choice of a well suited resource depends (among other quantities) on a *Quality of Service* (QoS) requested by the user. 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 that objective function z being able to satisfy users' QoS request. This cost function depends on costs due to the execution of the application on the resource, 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 resource (r_i) with altogether P_i computational nodes, and on the number of processors (P_k) used in the computation for each component. The user can prescribe the two constraints K_{MAX} (maximum cost) and T_{MAX} (maximum turn around time). The optimization problem writes:

$$\min z = \beta K_w \left(\bigcup_{k=1}^{n} (C_k, r_i, P_k) \right) + \sum_{k=1}^{n} \mathcal{F}_{C_k}(r_i, P_k) \quad \forall\ 1 \leq k \leq n \tag{22}$$

such that

$$\sum_{k=1}^{n} \Big(K_e(C_k, r_i, P_k, \varphi) + K_l(C_k, r_i, P_k) + K_{eco}(C_k, r_i, P_k) + K_d(C_k, r_i, P_k) \Big) \leq K_{MAX}$$

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

$$(r_i, P_k) \in \mathcal{R}(C_k),$$

where

$$\mathcal{F}_{C_k}(r_i, P_k) = \alpha_k \Big(K_e(C_k, r_i, P_k, \varphi) + K_l(C_k, r_i, P_k) \Big) + \gamma_k \Big(K_{eco}(C_k, r_i, P_k) \Big) + \delta_k \Big(K_d(C_k, r_i, P_k) \Big) \quad [\text{ECU}], \tag{23}$$

$$\alpha_k,\ \beta,\ \gamma_k,\ \delta_k \geq 0, \tag{24}$$

$$\alpha_k + \beta + \gamma_k + \delta_k > 0, \tag{25}$$

and $\mathcal{R}(C_k), k = 1, ..., n$ is the eligible set of resources for application's component C_k. We express the money quantity as Electronic Cost Unit ([ECU]) knowing that a non-trivial "Supply and Demand" mechanism is leading the worlds' currencies system if one takes into account an international Grid system. This was the case for the first IANOS testbed across Germany (Euros) and Switzerland (Swiss Francs). 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.

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 and represent the mathematical formulation of the QoS given by the users for each submission. 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 centers and user's demands. For instance, increasing β will increase usage of underused resources. One recognizes that a simulator is needed to estimate these parameters.

In fact, the user's (resource consumer) and the computing center's (resource provider) 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 and a reasonable usage of their resources. A simulator will be used to try to satisfy both. This implies a constant tuning of the free parameters.

4 Performance and Execution Time models

To compute the execution time cost K_e, it is fundamental to predict how long the execution will last. Thus, it depends on the performance of a given application on a resource. It is supposed that the application-relevant parameters O, S, Z defined in Section 2 are resource-independent. They vary with the problem size. We estimate their complexities for the DGEMM (full matrix-matrix multiplication), SMXV (sparse matrix × vector multiplication) and SpecuLOOS [1] (a spectral and mortar element analysis toolbox for the numerical solution of partial differential equations and more particularly for solving incompressible unsteady fluid flow problems) applications to be:

$$O(N_1, N_2, N_3) = a_1 N_1 N_2^{(a_2)} N_3^{(a_3)}$$
$$Z(N_1, N_2, N_3) = b_1 N_1 N_2^{(b_2)} N_3^{(b_3)}$$
$$S(N_1, N_2, N_3) = c_1 N_1 N_2^{(c_2)} N_3^{(c_3)}.$$

The integer N_1 denotes the number of time steps, the number of iterations, or more generally the number of times the algorithm is executed. The integer N_2

describes a matrix size, or characterizes an algorithm for which the complexity is not known. These numbers must be given by the user or specified in the input files at submission time. Table 6 (page 8) shows an example of O, S and Z for the SpecuLOOS application. This spectral code is aimed at direct numerical simulation (DNS) or large eddy simulation (LES) of incompressible fluid flow problems. The computational load is directly related to the discretized Navier-Stokes equations in space and time. Therefore, the algorithm has a complexity that can be measured by three major quantities. These are N_1 the time step of the time integrator, N_2 the number of spectral elements and finally the polynomial degree of the Lagrangian interpolants is N_3. More details on the numerical method can be found in [1]

The parameters a_i, b_i, c_i, $i = 1, ..., 3$ are computed by means of minimization processes using measurements made in the past. Measurements of O, Z, and S for at least three different values of N_i are needed to determine them. In this example we concentrate on the CPU time, thus finding a_i. Since these parameters define the number of operations, they are independent of the hardware on which the execution has been made. This means that when we know O and the resource-dependent parameters, it is possible to predict the CPU time for three different values of N_1, N_2 and N_3 for a different resource.

The total number of operations is

$$O(T_{exec}) = \sum_{i=1}^{N_t} O(i\Delta t) \tag{26}$$

The execution time T_{exec} can be found at the end of the execution, and the number of operations performed in each time slot i of size Δt is delivered by the Monitoring Module of IANOS and enables to compute the total number of operations performed. To improve the values of a_i and to adjust them to possible modifications in the hardware and the basic software such as an improvement of the compiler or the used libraries, these parameters are determined by an optimization procedure. The error function

$$\Phi = \sum_{i=1}^{N_c} w_i (O_i - R_a^i P_i E_i T_i)^2 \tag{27}$$

is minimized. Here, N_c is the number of execution data that is considered, R_a^i is the average CPU performance (FLOPS/s) delivered by the Monitoring Module, P_i is the number of processors, E_i is the efficiency or the CPU usage, T_i is the measured CPU time used, and w_i is a weight that varies in time. The older the execution time measurement is, the smaller w_i is. This makes it possible to take into account improvements of hardware, compilers, or libraries.

The minimization procedure consists of the resolution of the system for a_i:

$$\frac{\partial \Phi}{\partial a_i} = 0, \forall i \tag{28}$$

The variable a_1 is linear and can be eliminated. The non-linear equations for a_i, $i = 2, 3$ are solved using a Levenberg-Marquardt nonlinear least-squares algorithm.

5 First results

To test the IANOS framework, we choose SpecuLOOS [1]. SpecuLOOS uses a small amount of main memory. Parallelization is made in order to reduce the high overall computing time. The number of elements and the polynomial degrees in the three space directions are denoted by N_x, N_y, and N_z, and p_x, p_y, and p_z, respectively. There are $N_x \times N_y \times N_z$ elements for a 3D test case. For the purpose of simplicity, we assume that

$$\begin{aligned} N_2 &= \sqrt[3]{N_x \times N_y \times N_z} \\ N_3 &= max(p_x - 1, p_y - 1, p_z - 1) = p \end{aligned} \tag{29}$$

while N_1 denotes the number of time steps of the simulation. Then, the number of operations per iteration step is thought to be proportional to Eq. 30

$$O(N_1, N_2, N_3, N_{CG}) \approx N_1 N_{CG} N_2^{3.0} N_3^{4.0} \ [GFlop] \tag{30}$$

where N_{CG} is the number of conjugate gradient iterations.

5.1 SpecuLOOS on one computational node

First of all we run SpecuLOOS on one node, thus no communication is taken into account. We consider only computation. One time iteration of SpecuLOOS is divided into three main parts: (1) computes the tentative velocity (through a Helmholtz equation solved by preconditioned conjugate gradient method), (2) computes the pressure (through a sophisticated conjugate gradient) and (3) corrects the tentative velocity. More than 90 % is spent in the second part (the conjugate gradient). The execution time is spent essentially in the pressure computation of step (2).

N_1	N_2	N_3	T_{exec} $[s]$	N_{CG} # iter	T_{CG} $[s]$	$\frac{T_{CG}}{T_{exec}}$	$T_{CG,1}$ $[s]$
1	6	7	40.1	198	32.8	0.818	0.17
1	6	9	119.3	247	103.8	0.870	0.42
1	8	5	43.2	205	33.9	0.785	0.17
1	8	7	116.4	268	106.7	0.917	0.40
1	8	9	394.3	344	342.3	0.868	1.00
1	10	5	105.2	259	83.4	0.793	0.32
1	10	7	311.0	339	265.4	0.853	0.78

Table 3 SpecuLOOS on one node of Pleiades2. The number of conjugate gradient iterations N_{CG} is an average value over all time steps for the pressure. $T_{CG,1}$ is the time spent in one iteration of the conjugate gradient

Table 3 presents the results of SpecuLOOS on one node of the Pleiades2 cluster. In this table we concentrate on the behavior of one iteration of the conjugate gradient $T_{CG,1}$. We can then compute the behavior of the code with N_{CG}. Based on these results, we get the scaling law shown in Eq. 31.

$$T_{CG} = 2.01 \cdot 10^{-6} \cdot N_1 \cdot N_{CG} \cdot N_2^{(2.9)} \cdot N_3^{(3.3)} \quad (31)$$

T_{CG} depends on N_1, N_2, N_3 and N_{CG}. Comparing Eq. 31 with the theoretical scaling law (Eq. 30), we realize that the scaling with respect to the number of elements is the same whereas some lower exponent is obtained for the polynomial degree. We have to mention here that the number of conjugate gradient iterations steps depends on N_2 and N_3 and on other parameters that affect the matrix condition. Due to this fact, we need to include all the operations over all the conjugate gradient steps in the scaling law. Then we get Eq. 32.

$$T_{exec} = 1.15 \cdot 10^{-5} \cdot N_1 \cdot N_2^{(3.91)} \cdot N_3^{(4.19)} \quad (32)$$

T_{exec} depends on N_1, N_2 and N_3. As a consequence, $N_{CG} = 5.72 \cdot N_2^{1.01} \cdot N_3^{0.9}$.

5.2 SpecuLOOS in parallel

SpecuLOOS is a parallel code that uses MPI. We develop a small library that catches all the MPI calls. Then, analyzing the arguments, it is straightforward to measure the S and Z quantities for a specific run. We have to mention here that the IANOS Monitoring Module includes a modified kernel that automatically save the S, Z and O quantities. For these first results, the Monitoring Module was not available on all the production resources.

The execution time evaluation model (ETEM) authorizes to predict T_{exec} with a different number of computational nodes. Table 4 presents the results by varying N_2 and N_3 with a different number of nodes (here $P = 16$). Table 5 presents the results of the same case ($N1 = 1$, $N_2 = 8$ and $N_3 = 7$) but with different values of P.

			T_{exec} $[s]$		
N_1	N_2	N_3	*meas.*	*estim.*	δ
1	8	7	10.6362	10.1316	0.0498
1	10	7	26.1612	26.6085	0.0168
1	12	7	53.8883	57.2796	0.0592
1	14	7	98.0889	107.4067	0.0867
1	16	7	166.7650	181.9406	0.0834
1	18	7	264.0150	285.0532	0.0738
1	8	9	29.5992	33.0406	0.1041
1	8	11	83.2354	92.3402	0.0986
1	8	13	228.7220	232.5157	0.0163
1	8	15	538.2590	543.3537	0.0093

Table 4 Execution Time Evaluation Model on Pleiades2. $P = 16$, N_2 and N_3 vary.

				T_{exec} $[s]$		
N_1	N_2	N_3	P	*meas.*	*estim.*	δ
1	8	7	2	71.05	81.06	0.123
1	8	7	4	38.33	40.52	0.054
1	8	7	8	20.82	20.26	0.027
1	8	7	16	10.63	10.13	0.049
1	**8**	**7**	**32**	**6.17**	**5.06**	**0.219**
1	8	7	64	3.99	2.53	0.577

Table 5 Variation of the number of nodes P with the same case $N_2 = 8$, $N_3 = 7$ and $N_1 = 1$. $T_{exec} = T(N_1, N_2, N_3) * (32/P)$. The reference value is $P = 32$ (in bold).

6 Conclusion and future work

This paper presents a new and original approach for resource brokering in a HPC Grid environment based on the needs of the applications. Grid resources and HPC applications are parameterized so that it is possible to quantify what an application needs and what a resource provides. With a non-trivial resource brokering algorithm based on a cost function, it is then possible to pinpoint the best suited resource for a submitted application at a given time. The cost function estimates the total cost of an application on potential resources in the Grid. To do that, the execution time

			T_{exec} [s]			Z #			S [MBytes]		
N_1	N_2	N_3	*meas.*	*estim.*	δ	*meas.*	*estim.*	δ	*meas.*	*estim.*	δ
1	8	7	6.17	5.06	0.219	2745	3352	0.221	28.69	42.92	0.496
1	10	7	15.72	12.84	0.183	6783	5922	0.127	74.12	79.57	0.074
1	12	7	29.90	28.98	0.031	8067	7908	0.019	141.41	140.24	0.008
1	14	7	54.02	54.97	0.017	9459	8649	0.085	249.26	239.22	0.041
1	16	7	89.48	92.14	0.029	5629	8136	0.445	389.99	399.08	0.023
1	18	7	143.06	140.96	0.014	8354	6797	0.186	657.67	655.40	0.003
1	8	9	15.67	15.55	0.008	3435	4324	0.259	88.94	69.47	0.219
1	8	11	45.06	45.20	0.000	5361	5108	0.047	96.96	102.88	0.061
1	8	13	114.72	116.15	0.001	6267	5669	0.095	148.20	143.26	0.033
1	8	15	273.14	273.00	0.000	5553	6005	0.082	180.97	190.72	0.054
1	8	17	600.10	600.03	0.000	6199	6133	0.011	249.54	245.33	0.017

Table 6 Values of the Performance Prediction Model for the SpecuLOOS framework for one iteration. N_1 represents the number of time steps, N_2 the number of elements in each directions ($N_x = N_y = N_z = N_2$) and N_3 the polynomial degree of the Lagrangian interpolants (with $p_x - 1 = p_y - 1 = p_z - 1 = N_3$). The resource was Pleiades2 (a commodity cluster based on Xeon Serie 3000 processors and a GbE interconnection network). $P = 32$ for all the validation set.

of the application is predicted using an application's behavior model, input parameters and historical monitored data. The scaling rules of the application are used to predict the execution time. These rules can also be used to pinpoint poorly implemented algorithms if ones knows the theoretical scaling rule of a given application.

As far as we know, no other Grid-level resource broker is able to choose a resource based on the needs of the application. Even if some are "application-oriented", none of them try to quantify the needs of the applications and what the resources offer. The IANOS resource broker implements all the models and features developed in this paper. More detailed on the implementation can be found in [6].

Several adjustements must be done in the performance and execution time evaluation models. The model has been validated with a parallel application (SpecuLOOS) but also with other applications' kernels (such as matrix-matrix multiplication or sparse matrix × vector multiplication). We need to validate the model on other types of applications. We need a simulator to fine tune the weight parameters in the cost function.

References

1. Y. Dubois-Pélerin, V. Van Kemenade, M. O. Deville, *An Object-Oriented Toolbox for Spectral Element Analysis*, J. Sci. Comput.,**14**,1, pp. 1–29, 1999
2. Gruber, R. Volgers, P. De Vita, A. Stengel, M. Tran, T.-M., *Parameterisation to tailor commodity clusters to applications*, Future Generation Computer Systems, **19**, pp. 111–120, 2003
3. CSCS : Swiss National Supercomputing Centre,*Annual Report*, 2006
4. Menu J., *EPFL DIT HPC Clusters - batch statistics*, website, [Accessed on July 31, 2008] `http://dithpcbatch.epfl.ch/BatchStatistics.html`, 2008
5. Jeaunin M.*EPFL BG/L Utilisation by Group (January 2007 - December 2007)*, website : `http://hpc.epfl.ch/Members/mjaunin/bgl2007.pdf/download`, [Accessed on July 31, 2008], 2008
6. H. Rasheed, R. Gruber, V. Keller, W. Ziegler, O. Wäldrich, P. Wieder, P. Kuonen, M-C. Sawley, S. Maffioletti P. Kunszt, *IANOS : An Intelligent Application Oriented Scheduling Middleware For An HPCN Grid*, CoreGRID serie Grid Computing, pp. 237 – 248, Springer, 2008
7. Keller Vincent, *Optimal Application-Oriented Resource Brokering in a High Performance Computing Grid* (PhD thesis), 4221, EPFL, Lausanne (Switzerland) (2008)