Optimal number of nodes for computation in grid environments

Lorenzo Muttoni, Giuliano Casale, Federico Granata, Stefano Zanero
Politecnico di Milano
Dipartimento di Elettronica ed Informazione
Via Ponzio 34/5, 20133 Milano (Italy)
Tel: +39 (0)2 2399-3660; Fax: +39 (0)2 2399-3411
{muttoni, casale, granata, zanero}@elet.polimi.it

Abstract

In this paper we show that there exists an optimal number of nodes to be assigned to jobs for execution in grid systems, which depends on the distributions of computation and communication service times. We also show that the analytical models proposed for parallel computers are not accurate for grid systems. We therefore extend to grid environment the definitions of speedup, efficiency and efficacy that are usually given for parallel systems. We also adopt a queueing network model with three different types of workload to prove that in every case an optimal number of nodes exists and that the mean value of CPU and communication service times is just a scale factor for this optimum.

1. Introduction

Computational grids are increasingly becoming reliable and powerful platforms, which allow researchers to access the resources they need for the execution of computation intensive applications. Tapping fully into the potential of such systems to achieve good performance is still an open and challenging problem.

Towards this direction, the adoption of a well-designed scheduler is a fundamental step for any grid infrastructure. Many researchers in the last few years have proposed various scheduling algorithms for grid systems [7], [26], [16], [9], [24], [10]. Unfortunately the search for an efficient solution is made more complex by the difficulty of evaluating the performance of scheduling algorithms using large-scale benchmarks with reproducible results [25].

In this paper we focus on a specific topic: the issues related with the identification of the optimal number of nodes to involve in a computation in order to minimize the execution time of the application. This number depends on the characteristics of the executed task in terms of computation and communication requirements. Our aim is to show that the assumption of different statistical distributions to model execution and communication times leads to very different results.

We suggest how an analytical model proposed and used for evaluating the performance of parallel computers can be extended, under some assumptions, to grid systems. We also show how the variability of the characteristics of the nodes (computational power, I/O performance, load, ...) affects the equations of the model. A similar issue was recently investigated in [6].

To analyze the effects of these variations, we simulated the behavior of three different applications, each representative of a particular workload, by the means of a simplified queueing network model of a grid system. For each application, we show the existence of an optimal number of nodes, such that the allocation of additional resources does not produce any valuable increase of the overall performance. Moreover, we remark that an excessive fragmentation of the computation usually yields additional communication costs that increases total execution time if not balanced by an equal decrease of computation time.

We also show that the optimal number of nodes depends on the assumed distribution of the computation and communication times. This suggests that the results obtained for parallel systems are not fit for a grid environment, since the communication among nodes cannot be modeled any more by an exponential distribution [22, 14]. Indeed, as pointed out by [5], transfer times over the Internet are better modeled by heavy-tail distributions. This suggests that a deep understanding of the underlying distribution of Internet traffic is required, in order to design and evaluate efficient scheduling algorithms.

This paper is organized as follows: in section 2 we extend to grid systems a workload characterization method proposed for parallel system. The problems which arise while adapting this method are discussed. In section 2.3 we introduce a queueing network model for grid environment and we describe the simplifications needed to actually simulate

it. In section 3 the simulation environment (3), the parameters of the model (3.1), the results of the simulation (3.2) and some validation tasks (3.3) are presented. Section 4 draws the conclusions of this paper and outlines some possible future researches in this field.

2. Workload characterization

Let us consider the total execution time of a job T, which is dependent on the number of nodes of the grid that are scheduled to execute it, thus $T = T(n)$. T can be thought as the sum of two components: the total computation time T_{comp} and the total time T_{comm} spent on communication and synchronizations between nodes: $T(n) = T_{comp}(n) + T_{comm}(n)$.

Looking at this model, we can say that in most cases $T_{comp}(n)$ is a monotonically non-increasing function of n, since the addition of a new node provides new computational resources, while $T_{comm}(n)$ is monotonically non-decreasing because of the additional communication overhead imposed by the new node. However, the shape of these functions is strongly dependent on the kind of application and in some cases can even fall out of these rules. What we can usually see is that the application switches from computation bound to communication bound as n increases.

This model extends a similar model presented for parallel systems [15] with the only difference of using n (number of nodes) instead of p (number of processors on a machine).

Also, in parallel systems analysis the concept of speedup S is commonly used, defined as the ratio of the execution time of the job when executed on one processor to that when executed on p processors: $S(p) = T(1)/T(p)$.

Typical speedup curves for various types of applications running on parallel machines are shown in Figure 1. In general, the shape of the speedup curves is dependent on the characteristics of the load generated by the execution of the considered application [2].

2.1. Speedup, efficiency and efficacy in grid environments

We now give a first definition of the speedup in a grid environment as the gain in time of the job when executed on a partition of n nodes instead than on a single machine: $S(n) = T(1)/T(n)$. The problem with this extension to the grid environment is that $S(n)$ is not as effective in describing grid workload as $S(p)$ is for parallel workload, since parallel machines are composed of a number of identical CPUs, while on the grid there are heterogeneous machines.

In order to provide a better definition of the speedup, we now introduce the concept of *power units*, defined in
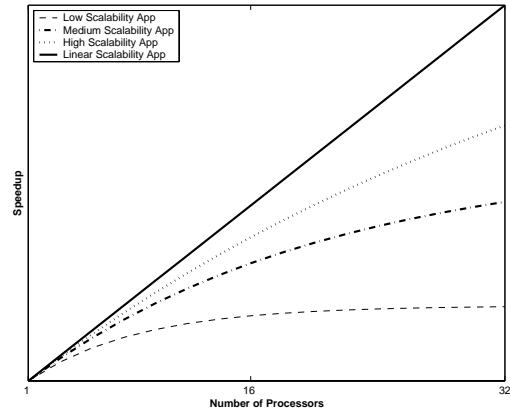

Figure 1. Representative speedup curves on parallel machines for various applications

a generic manner as a measure of atomic units of computation power available in the grid. This opens up the large problem of choosing the metric beneath this concept. It has been demonstrated many times [20] that neither the CPU clock frequency nor the megaflops are effective when comparing performance among different applications running on different architectures. Any measure of computational power, then, has to compare the effective performance of different machines on the actual application under consideration, taking into account its distinctive features. This is not easy to do, but could be conceptually achieved in the following way: for each node and each different workloads, we could define a coefficient l that, working as a weight for the CPU clock speed, can give a value of the computational power of *that* particular node, under *that* particular workload, with respect to the node architecture (x86, Sparc, PPC,...) taken as reference. Without any doubt this is not very practical, but on this common scale we could calculate the power units of each machine on the grid. So, let PU be the aggregated computational power (expressed in terms of power units) of a partition to which a single application has been scheduled. It is clear that PU depends on n somehow, but this relation changes with the type of the n machines that are partitioned together: thus it is not, in general, a single-valued function of n. Symbolically, let PU_i be the power of the i-th machine on the grid, if the partition is defined by the set $A = \{\ x \mid$ the x-th machine is on the grid $\}$, then $PU = \sum_{i \epsilon A} PU_i$ and $n = |A|$.

We can also observe that T_{comp} is actually dependent on PU, as opposed to T_{comm} which is dependent only on n and not on PU. So, T is a function of both n and PU and can be expressed as $T = T(n, PU) = T_{comp}(PU) + T_{comm}(n)$. The shape of these two functions is still dependent on the application we are considering: in other terms, this is still the workload characterization of each job. However, if the

application workload is correctly characterized, it is valid the relation $T_{comp}(PU) = k/PU$, where k is a proportionality constant.

We can now extend all the results on the speedup to the grid environment by considering a different definition of the speedup:

$$S(n, PU) = \frac{T(1,1)}{T(n, PU)} = \frac{T_{comp}(1) + T_{comm}(1)}{T_{comp}(PU) + T_{comm}(n)} =$$

$$= \frac{T_{comp}(1)}{T_{comp}(PU) + T_{comm}(n)}$$

It is easy to see that for $n = 1$ this new definition is reduced to the traditional one since $T_{comm}(1) = 0$.

Besides this, a definition of efficiency is also usually given for parallel systems: this is the ratio of the actual speedup versus the maximum theoretical speedup [8] and is given by the expression:

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{T(p)} \cdot \frac{1}{p} = \frac{T_{opt}(p)}{T(p)}$$

where $T_{opt}(p)$ is optimal execution time for p processors, while $T(p)$ is the measured execution time. It is important to remark that $T_{opt}(p)$ is $T(1)/p$ in the best-case with linear speedup and with no memory constraints.

Also the definition of efficiency can be extended to grid systems. Let T_{opt} be the optimal execution time using a single machine with the whole computation power of the grid partition: $T_{opt}(1, PU) = T_{opt}(PU)$. As said before, T_{opt} is in the best case $T_{opt}(1, PU) = T(1,1)/PU$ (again, we are ignoring here any effect given by the additional availability of memory). We could say, thus, that:

$$E(n, PU) = \frac{T_{opt}(PU)}{T(n, PU)} = \frac{T(1,1)}{PU \cdot T(n, PU)} = \frac{S(n, PU)}{PU}$$

Following [11] we can also extend the definition of efficacy, $\eta(p)$, which is the ratio of the speedup versus the cost, defined as $C(p) = 1/E(p)$:

$$\eta(p) = \frac{S(p)}{C(p)} = \frac{S(p)}{\frac{1}{E(p)}} = E(p) \cdot S(p)$$

This formula seamlessly translates to:

$$\eta(n, PU) = E(n, PU) \cdot S(n, PU) =$$

$$= \frac{S(n, PU)}{PU} \cdot S(n, PU) = \frac{S(n, PU)^2}{PU}$$

The interest for these extensions is clarified in the following subsection, in which we discuss if some of the previous results on speedup and efficacy for parallel machines can be extended to grid systems.

2.2. Analytical derivation of optimal performance metrics

It has been shown in [11] that for parallel systems maximizing $\eta(p)$ (defined with the cost function noted above) yields the maximum performance index power, which is the ratio of the system throughput to the response time [13].

In parallel environments we can calculate an analytical expression for this optimum as follows:

$$\frac{d\eta(p)}{dp} = \frac{d}{dp}\left[\frac{S(p)^2}{p}\right] = \frac{1}{p}\left[2S(p)\frac{dS(p)}{dp} - \eta(p)\right]$$

So, there is a stationary point (a maximum) for

$$\frac{d\eta(p)}{dp} = 0 \Rightarrow 2S(p)\frac{dS(p)}{dp} - \eta(p) = 0 \Rightarrow$$

$$\Rightarrow \frac{dS(p)}{dp} = \frac{1}{2S(p)} \cdot \eta(p) = \frac{1}{2}\frac{S(p)}{p}$$

This means that $\eta(p)$ achieves a relative maximum when the slope of $S(p)$ is $\frac{1}{2}\frac{S(p)}{p}$ (and thus positive). Let $\overline{p}$ be the maximum of $\eta(p)$. Assuming that $S(p)$ is a convex function, it follows that if it has a maximum, it achieves it in $p > \overline{p}$. So, choosing the number of processor as the optimum of $S(p)$ or as that of $\eta(p)$ leads to very different behaviors for the system: in the former, we optimize the response time; in the latter, we choose to maximize the performance index power (the ratio of throughput to response time).

As said before, it is of interest to verify if similar results hold for grid environments and if this information could be integrated into the scheduler to improve the overall performance of the system. Unfortunately, a number of questions arise when trying to derive similar equations for our model.

The expressions of $\eta(n, PU)$ and $S(n, PU)$ do not let us have simple results in terms of maxima and minima: a first reason is that it is only an approximation that different subsets of the grid of equal computing power PU and cardinality n yield the same speedup S. So, $S(n, PU)$ and $\eta(n, PU)$ are not, in general, single-valued functions. In addition, since the machines on the grid are not dedicated, the resources are not necessarily available for grid computations at all times; so, the available computational power in the partition is actually a stochastic process. Since the task of deriving an exact formulation for PU cannot be done easily, we leave it as a future work and we simply remark that probably, as can be seen from the experimental results reported in section 3, there should exist a couple (n, PU) which maximizes $\eta(n, PU)$. Adding nodes to the partition beyond this point yields benefits, expressed in terms of $S(n, PU)$, that are marginal with respect to the cost and we could even reach a point where $S(n, PU)$ actually decreases, since the decrease of T_{comp} can be lower than the corresponding increase of T_{comm}.
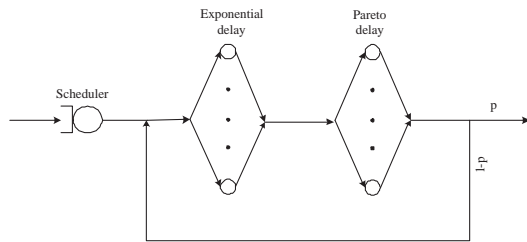
We observe that in general finding the optimum for our model leads to a difficult mixed, in the worst case even non-linear, integer optimization problem.

In the following sections, we propose a model of a grid computing environment and we use this model to show that an optimal number of processors indeed exists, in good analogy with what happens for parallel systems. We also show by simulation how this optimal point depends on workload and system characteristics.

2.3. The proposed model

For the sake of clarity we now briefly describe the components and the features of the ideal grid system we are going to model.

A grid system is basically a set of N computing nodes, interconnected by a network, which usually is the Internet. Between these nodes a number of different jobs circulates, dynamically allocated by a scheduler to each node.

The scheduler operates by knowing the static features of each node on the grid and also by forecasting local load conditions. The two most widely used grid resource information systems are the Metacomputing Directory Service (MDS) and the Network Weather Service (NWS). MDS is a grid information management system that is used to collect and publish system configuration, capability, and status information. Examples of the information that can be retrieved from an MDS server include operating system, processor type and speed, the number of available CPUs and software availability as well as their installation locations. NWS, instead, is a distributed monitoring system designed to track and forecast resource conditions. Examples of the information that can be retrieved from an NWS server include the fraction of CPU available to a newly created process, the amount of currently unused memory and the bandwidth with which data can be sent to remote hosts. As remarked before, each node of the grid can run also local applications, since it's not necessary for it to be dedicated.

In order to model this system, we need some assumptions. We will model a single application, scheduled to run on a subset of the grid (with n nodes, $n \ll N$).

The n tasks can obviously intercommunicate. A general model should take into consideration that the communication phase can happen at any time. However, it has been shown [23] that a wide family of parallel applications show computation phases alternated to I/O phases. Similarly, a lot of distributed computing applications have a computing phase, followed by either a communication phase (with probability q) or by another computation phase. This loop repeats a number of times: the probability that after a computation phase the loop ends and the calculation is complete is p. The diagram for this model is shown in Figure 2.

Figure 2. The proposed model for a grid computing system

A similar model for parallel MIMD architectures is shown in [4].

Since we do not model directly neither other applications on the grid that may be scheduled on the same processors, nor the computational load from local processes, we need a way to express these effects. So, we introduced then the concept of *residual power* on a node, that is simply its PU at any given instant. For this reason, we will model the service time of each node (i.e. its residual power) with an exponential distribution, which has a characteristically high variance. If the nodes were dedicated processors with no external load, a distribution with a lower variance (e.g. a uniform distribution) should be used instead.

Let be m the number of intercommunications between nodes, with $m < n \cdot (n - 1)$. We model the communication system by defining a factor of intercommunication f which expresses the likelihood that one of the n processes needs to communicate with another: $m = f \cdot [n \cdot (n - 1)]$. Each communication can be modeled as a pure delay with no queue, following a heavy-tail distribution, for instance a Pareto [5]. It is self-evident, however, that m delays between a fork and a join are exactly equivalent to the maximum of the m times of the delay, since, in any case, the join must wait for the slowest job.

To further simplify the simulation, we suppose to have an ideal scheduler that can allocate properly every task on the node, which is both best-suited to execute it and with the lowest possible load. Under this assumption, we can transform the nodes in pure delays with no queue, and eliminate the fork-and-join structure around the processor array, as we did with the Pareto delays of communication: the total delay is the equivalent to the maximum of the n processor delays. The schema of the resulting queueing network is shown in Figure 3.

Figure 3. A simplified version of the model in Figure 2

n	T_{CPU}	T_{Comm}	T	$S(n)$
1	1743,03	0,00	1743,03	1,00
2	871,51	22,35	893,86	1,95
4	435,76	34,08	469,83	3,71
8	217,88	41,05	258,93	6,73
16	108,94	46,72	155,66	11,20
32	54,47	53,62	108,09	16,13

Table 1. Mean values of the service time (in seconds) and speedup for the high scalable workload, LU

n	T_{CPU}	T_{Comm}	T	$S(n)$
1	2534,80	0,00	2534,80	1,00
2	1267,40	80,90	1348,30	1,90
4	126,16	759,86	633,70	3,44
8	316,85	158,15	475,00	5,70
16	158,42	190,94	349,36	8,15
32	79,21	230,14	309,35	9,66

Table 2. Mean values of the service time (in seconds) and speedup for the medium scalable workload, SP

3. Simulating the model

In order to solve our model we used a simulator for queuing networks, based on a discrete event paradigm applying the well-known method of Monte Carlo simulations. The experiment was repeated many times and the random number generator (Mersenne-Twister [17]) was re-initialized at each step with a different seed. We estimated all measures with a confidence interval of 95% and a precision of 5%.

We considered both the problems of correlation and initial transient: we solved the first problem using a spectral method to estimate the confidence intervals of the mean value of the desired performance measures[12].

The problem of the initial transient, instead, was addressed by removing just the most biased part of the data, since removing all would only increase the variance. To do so, we used the procedure presented in [21], that is based on the Shruben Test, but it's preceded by an heuristic algorithm that selects the length of the sequence to be tested for the steady state condition.

3.1. Definition of the model parameters

In order to define the parameters for the service centers of the model, we had to choose three applications representative of typical workloads. Using the data presented in [1, 18], we verified that three kernels of the NPB [19] benchmark suite (Class A) fit the speedup curves of figure 1: MG (Multi Grid) as representative of low scalable applications, SP (Pentadiagonal Solver) for medium scalable and LU (LU Matrix Factorization) for high scalable. We considered data presented in [1, 18] well suited to describe a grid environment since the experiments involved heterogeneous machines with different CPUs and amounts of memory. These conditions are much more similar to grid environments than a *beowulf* cluster or a supercomputer can be.

Since on the NPB web site there are no recent data for clusters with a significant number of nodes (from 1 to at least 32), the service time of CPU and communication system for all the proposed test cases were estimated by fitting the measured values reported in [18]. The difference between the measured and estimated values is less than 10%. The NAS benchmarks report only the total execution time; since for our model we need a value for both CPU time and communication time, we estimated these values. According to [3], T_{CPU} is proportional to $1/n$, where n is the number of processors involved in the computation; clearly the communication time, T_{Comm}, is the difference between the total execution time, T, and T_{CPU}.

Tables 1, 2 and 3 report the mean values used in the simulation.

It has to be remarked that the T_{Comm} value is an overestimate for the bandwidth available in a grid system, since it represents the communication time on a dedicated 100 Mbps Ethernet Switch. However, we decided to use this value since we want to show that our conclusions are true in the best case: it is straightforward that our results are true also in less optimistic situations.

We considered the following statistical distributions: *uniform*, *exponential* and *Pareto*. We recall that a Pareto distribution has a pdf given by

$$p(x) = \alpha k^{\alpha} x^{-\alpha-1} \quad \alpha, k > 0 \quad x \geq k$$

and, in general, it does not have all moments. For instance, if $\alpha \leq 2$ the distribution has infinite variance; if $\alpha \leq 1$ also

n	T_{CPU}	T_{Comm}	T	$S(n)$
1	74,97	0,00	74,97	1,00
2	37,49	5,36	42,84	1,75
4	18,74	8,68	27,42	2,73
8	9,37	11,46	20,83	3,60
16	4,69	14,25	18,93	3,96
32	2,34	16,40	18,74	4,00

Table 3. Mean values of the service time (in seconds) and speedup for the low scalable workload, MG

Figure 4. Experimental results with a low scalable application (MG)

the mean is infinite. It is clear that not all the permutations make sense. We chose to consider the following:

- $S_{CPU} \sim uniform$ and $S_{Comm} \sim exponential$: this is the model of a traditional *beowulf cluster* with homogeneous dedicated nodes and a dedicated communication system (e.g. 100 Mbps switched Ethernet);

- $S_{CPU} \sim uniform$ and $S_{Comm} \sim Pareto$: this case could model a distributed system in which the nodes are still homogeneous and dedicated with Internet as the interconnection system;

- $S_{CPU} \sim exponential$ and $S_{Comm} \sim exponential$: this is the model of a system in which heterogeneous nodes are shared among the parallel application and local processes: this fact increases by a significant amount the variability of CPU service time; the communication system is a light load communication system (e.g. 100 Mbps switched Ethernet);

- $S_{CPU} \sim exponential$ and $S_{Comm} \sim Pareto$: this is the case of heterogeneous nodes with local load and distributed over the Internet.

Since Internet file transfer times are distributed according to a heavy-tailed distribution [5], we chose a Pareto distribution with $\alpha = 1.8$ and mean μ; k can be calculated from this two parameters as $k = \mu(\alpha - 1)/\alpha$. The parameter α is independent from the specific size of the file. $\alpha = 1.8$ is not a pessimistic value: in general, as α goes down to a threshold ($\alpha = 1.3$), the commmunication performances quickly degenerate. Our choice, instead, represents an Internet connection with low to medium load.

The computation time was modeled with a uniform distribution when we did not consider the effect of residual power. The parameters are the mean and the variance, the first is equal to the values in tables 1, 2, 3 and the second was chosen equal to 5% of the mean, which was the measured value on our heterogeneous cluster. The choice of parameter λ of the exponential distribution is straightforward: it's equal to the reciprocal of the values in tables 1, 2, 3.
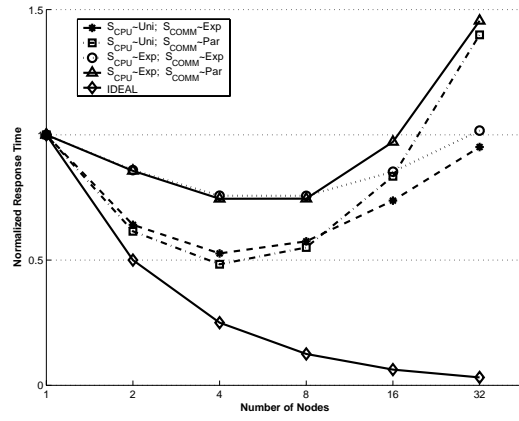
The last parameter of our model that has to be defined is p, the probability that a job leaves the system having completed its computation and communication phases. The parameter p is determined by the average number of "loops", L, performed by the application.

Let L be the average number of loops of an application. Then $L = p + 2p(1 - p) + 3p(1 - p)^2 + ... = \sum_{k=1}^{\infty} kp(1 - p)^{k-1} = \frac{p}{1-p} \sum_{k=1}^{\infty} k(1 - p)^k = 1/p$ which implies $p = 1/L$.

Since a parallel application typically alternates various computation and communication phases, for each of the considered applications we calculated the mean number of loops necessary for an execution (Class A) and we set the probability p to this value. The mean service times of both CPU and communication system were scaled by the number of loops. For what concerns the scheduler service time, it was overestimated in 1 second including the scheduling computation time and the time to query the grid information services like MDS and NWS.

3.2. Results and interpretation

Several simulations were executed for the three applications representative of each workload. For each of them we varied two parameters: the CPU service time distribution and the communication service time distribution.

Figures 4, 5 and 6 present the simulation results for the low scalable, medium scalable and high scalable applications respectively. In order to compare the results, the values were normalized to 1. It is important to remember that all the distributions have the same mean.

It is clear from the plots that, depending on the choice of the distributions, the optimal number of nodes changes (4-8 for low scalable applications, 8-16 for medium scal-
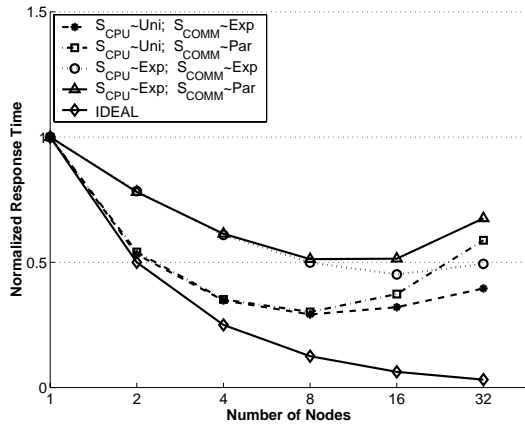
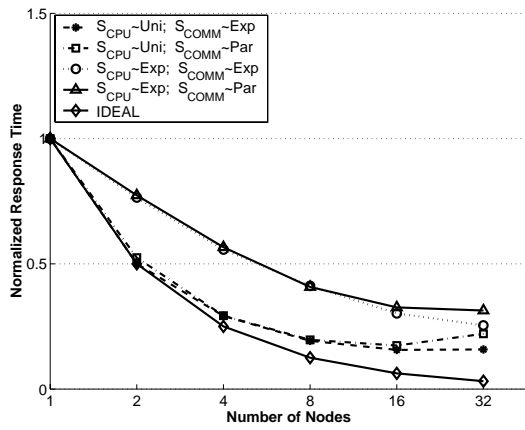**Figure 5. Experimental results with a medium scalable application (SP)**

Figure 6. Experimental results with a high scalable application (LU)

able applications and 16-32 for high scalable applications). For a low number of nodes (about 8 nodes) the dominating parameter is the computation time distribution: the four curves are virtually the same, since it has to be remembered that they are simulated values with a confidence interval. Instead, as the number of nodes grows, the distribution of the communication times affects the behavior of the system and the total response time. This fact is clear in Figures 4 and 5 where for more than 16 nodes the curve uniform-Pareto intersects and then overtakes the exponential-exponential curve. It is reasonable to presume to find the same trend in the high scalable application : this effect is only delayed (to more than 32 nodes), due to the workload characteristics.

3.3. Validation of the results

Real experimental data measured for grid performance are not yet available [19]. For this reason it is not possible to compare the results of the simulations with real values. Under the hypothesis that in the system (i.e. the partition of nodes involved in the computation) there is only one *grid application* running, the variability of computational power of the nodes is due only to the local load and it is possible to simplify our model in order to obtain an analytical solution. Under this hypothesis, the loop representing the succession of computation and communication phases can be eliminated: the service times of the CPU and the communication system become the *aggregated service time*, i.e. the total time used by the application for computation and communication. Moreover, as said before, the queueing centers of the two fork/join blocks of the model become delays and, for this reason, the service time of the whole blocks are now the maximum of n random variables. Let $Y = \max(X_1, ..., X_n)$ be a random variable defined as the maximum of n random variables and suppose that these variables X_i represent the service times of the n nodes involved in the computation. Assume also that X_i are mutually independent and with the same distribution. It follows that

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \frac{d[F_X(x)]^n}{dx} = n f_X(y)[F_X(y)]^{n-1}$$

is the probability density function of the maximum of n equidistributed random variables. The mean value of Y is

$$E[Y] = \int_{-\infty}^{\infty} n \cdot x \cdot f_X(y) \cdot [F_X(y)]^{n-1} dx$$

This value should be calculated for each distribution and for each number of nodes.

The values calculated in this way are coherent with simulation results with an accuracy of 5% and a confidence interval of 95%; the errors was less than 2%. These clearly validate our simulation results.

4. Conclusions and future work

In this paper we showed that the workload characterization of the applications running on a grid is fundamental to predict the expected performance of the system. It was also pointed out that the statistical distributions used to model the computation and communication time heavily affect the response time. For this reason scheduler designers have to consider that, to guarantee good average performance, it is crucial to consider both:

1. the characteristics of the workload of the applications to be scheduled,

2. the statistical model of computation time on non dedicated nodes and of Internet transfer time.

It is straightforward that applications involving no communication or negligible amount of communication (e.g. SETI@home) are a trivial case in which the response time is always decreasing when n increases.

One of the possible developments of this work is to find the number of nodes that actually maximizes the performance power index. In order to accomplish that it will be necessary to use the general model presented in Figure 3, choose a scheduling algorithm and apply a workload composed by a mix of different applications. Finally it could be interesting to derive an analytical expression of the PU as a function of the workload and the hardware and software architectures of the grid nodes.

References

[1] G. Alfonsi and L. Muttoni. Performance evaluation of a Windows NT based PC cluster for high performance computing. *Journal of System Architecture, to appear*, 2003.

[2] M. Calzarossa and G. Serazzi. Workload characterization: A survey. *Proc. of the IEEE*, 81(8):1136–1150, 1993.

[3] F. Cappello and D. Etiemble. MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks. In *Proc. of the 2000 ACM/IEEE Conf. on Supercomputing (CDROM)*. IEEE Computer Society Press, 2000.

[4] P. Cremonesi and C. Gennaro. Integrated performance models for SPMD applications and MIMD architectures. *IEEE Trans. on Parallel and Distributed Systems*, 13(12):1320–1332, 2002.

[5] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE /ACM Trans. on Networking*, 5(6):835–846, December 1997.

[6] J. Cuenca, D. Giménez, J. González, J. Dongarra, and K. Roche. Automatic optimisation of parallel linear algebra routines in systems with variable load.

[7] H. Dail, H. Casanova, and F. Berman. A modular scheduling approach for grid application development environments. Technical Report CS2002-0708, UCSD CSE, 2002. submitted to Journal of Parallel and Distributed Computing.

[8] D. L. Eager, J. Zahorjan, and E. D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Trans. on Computers*, 38(3):408–423, March 1989.

[9] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *Proc. of the 2nd IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID'02)*, May 2002.

[10] C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in grid computing. In *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *LNCS*.

[11] D. Ghosal, G. Serazzi, and S. K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Trans. on Software Engineering*, 17(5):443–453, May 1991.

[12] P. Heidelberger and P. D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Communications of the ACM*, 24(4):233–245, 1981.

[13] L. Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proc. of the Int'l Conference on Communications (ICC)*, volume 43, pages 1–10, Boston, MA, 1979. IEEE.

[14] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. *On the Self-Similar Nature of Ethernet Traffic . IEEE/ACM Trans. on Networking*, 1994.

[15] M. R. Leuze, L. W. Dowdy, and K.-H. Park. Multiprogramming a distributed-memory multiprocessor. *Concurrency - Practice and Experience*, 1(1):19–34, 1989.

[16] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *Proc. of the 11th IEEE Symp. on High-Performance Distributed Computing*, July 2002.

[17] M. Matsumoto and T. Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[18] L. Muttoni and P. Tonazzo. Cluster di PC in ambiente Windows NT. Master's thesis, Politecnico di Milano, Oct. 2000.

[19] NAS. *NAS Parallel Benchmark. http://www.nas.nasa.gov/Software/NPB*.

[20] A. Patterson and J. Hennessy. *Computer Organization and Design - The Hardware/Software Interface*. Morgan Kaufmann Publishers, 1998.

[21] K. Pawlikowski. Steady-state simulation of queueing processes: survey of problems and solutions. *ACM Computing Surveys (CSUR)*, 22(2):123–170, 1990.

[22] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE /ACM Trans. on Networking*, 3(3):226–244, 1995.

[23] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante. Models of parallel applications with large computation and I/O requirements. *IEEE Trans. on Software Engineering*, 28:286–307, March 2002.

[24] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan. Design and evaluation of a resource selection framework for grid applications. In *Proc. of the 11th IEEE Symp. on High-Performance Distributed Computing*, July 2002.

[25] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *Proc. of 8th IEEE Int'l Symposium on High Performance Distributed Computing*, august 1999.

[26] S. Vadhiyar and J. Dongarra. A metascheduler for the grid. In *Proc. of the 11th IEEE Symp. on High-Performance Distributed Computing*, July 2002.