Dynamic Load Balancing for a Grid Application

Menno Dobber, Ger Koole, and Rob van der Mei

Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam,
The Netherlands
{amdobber, koole, mei}@few.vu.nl
http://www.cs.vu.nl/~amdobber

Abstract. Grids functionally combine globally distributed computers and information systems for creating a universal source of computing power and information. A key characteristic of grids is that resources (e.g., CPU cycles and network capacities) are shared among numerous applications, and therefore, the amount of resources available to any given application highly fluctuates over time. In this paper we analyze the impact of the fluctuations in the processing speed on the performance of grid applications. Extensive lab experiments show that the burstiness in processing speeds has a dramatic impact on the running times, which heightens the need for dynamic load balancing schemes to realize good performance. Our results demonstrate that a simple dynamic load balancing scheme based on forecasts via exponential smoothing is highly effective in reacting to the burstiness in processing speeds.

1 Introduction

Often, grid environments are seen as the successors of distributed computing environments (DCEs). Nevertheless, these two environments are fundamentally different. A DCE environment is rather predictable: the nodes are usually homogeneous, the availability of resources is based on reservation, the processing speeds are static and known beforehand. A grid environment, however, is highly unpredictable in many respects: resources have different and usually unknown capacities, they can be added and removed at any time, and the processing speeds fluctuate over time. In this context, it is challenging to realize good performance of parallel applications running in a grid environment. In this paper we focus on the fluctuations in processing speeds. First, we conducted elaborate experiments in the Planetlab [1] grid environment, in order to investigate over which time scales the processing speed fluctuates. Experimental results show fluctuations over different time scales, ranging from several seconds to minutes. Second, we analyze the potential speedup of the application by properly reacting to those fluctuations. We show that dynamic load balancing based on forecasts obtained via exponential smoothing can lead to a significant reduction of the running times.

Fluctuations in processing speeds are known to have an impact on the running times of parallel applications, and several studies on analyzing the impact of the fluctuations on the running time have been conducted. However, these

L. Bougé and V.K. Prasanna (Eds.): HiPC 2004, LNCS 3296, pp. 342–352, 2004.

fluctuations are typically artificially created, and hence controllable (see for example [2]), whereas the fluctuations in grid environments are not controllable. In the research community several groups focus on performance aspects of grid applications. A mathematicians' approach is to develop stochastic models for processing speeds, latencies and dependencies involved in running parallel applications, to propose algorithms for reduction of the running time of an application, and to provide a mathematical analysis of the algorithm [3, 4, 5]. Such a mathematical approach may be effective in some cases; however, usually unrealistic assumptions have to be made to provide a mathematical analysis, which limits the applicability of the results. On the other hand, computational grid experts develop well-performing strategies for computational grids, i.e., connected clusters consisting of computational nodes. However, due to the difference in fluctuations between general grid environments and computational grids, the effectiveness of these strategies in a grid environment is questionable [6]. A third group of researchers focuses on large-scale applications with parallel loops (i.e., loops with no dependencies among their iterations) [7, 8], combining the development of strategies based on a probabilistic analysis with experiments on computational grids with regulated load. However, due to the absence of dependencies among the iterations of those applications, these strategies are not applicable to parallel applications with those dependencies. These observations heighten the need for an integrated analysis of grid applications (including dependencies among their iterations), combining a data approach with extensive experimentation in a grid environment.

The increasing popularity of parallel applications in a grid environment creates many new challenges regarding the performance of grid applications, e.g., in terms of running times. To this end, it is essential to reach a better understanding of (1) the nature of fluctuations in processing speeds and the relevant time scale of these fluctuations, (2) the impact of the fluctuations on the running times of grid applications, and (3) effective means to cope with the fluctuations. The goal of this paper is to address these questions by combining results from lab experiments with mathematical analysis. To address these questions we have performed extensive test-lab experiments in a grid environment called Planetlab [1] with the classical Successive Over Relaxation (SOR) application. First, we provide a data analysis showing how processing speeds change over time. The results show fluctuations over multiple time scales, ranging from seconds to minutes. Then, we focus on the impact of the fluctuations on the running times for SOR at different time scales. The results show a dramatic influence of fluctuating processing speeds on running times of parallel applications. Subsequently, we focus on a dynamic load balancing scheme to cope with the fluctuations in processing speeds. We show that significant reductions in running times can be realized by performing load balancing based on predictions via the classical exponential smoothing technique.

This paper is organized as follows. In Section 2 we will describe the Planetlab testbed environment and the SOR application used in our experiments. Section 3 will show the data collection procedure and results about the different time scales

of the fluctuations in processing speeds. In Section 4 different load balancing strategies will be presented. Finally, in Section 5 the results and in Section 6 the conclusions will be addressed.

2 Experimental Setup

Experiments were performed with a parallel application on a grid test bed. A main requirement for the test bed is that it needs to use a network with intrinsic properties of a grid environment: resources with different capacities and many fluctuations in load and performance of geographically distributed nodes. We have performed our experiments on the Planetlab test bed [1], which meets these requirements. PlanetLab is an open, globally distributed processor-shared network for developing and deploying planetary-scale network services.

The application has also been carefully chosen so as to meet several requirements. On the one hand, the application must have dependencies between its iterations, because most of the parallel applications have that property, while on the other hand the structure of the dependencies should be simple. A suitable application is the Successive Over Relaxation (SOR) application. SOR is an iterative method that has proven to be useful in solving Laplace equations [9]. Our implementation of SOR deals with a 2-dimensional discrete state space $M \times N$, a grid. Each point in the grid has 4 neighbors, or less when the point is on the border of the grid. Mathematically this amounts to taking a weighted average of the values of the neighbors and its own value. The parallel implementation of SOR is based on the Red/Black SOR algorithm [10]. The grid is treated as a checkerboard and each iteration is split into phases, red and black. During the red phase only the red points of the grid are updated. Red points only have black neighbors, and no black points are changed during the red phase. During the black phase, the black points are updated in a similar way. Using the Red/Black SOR algorithm, the grid can be partitioned among the available processors. All processors can update different points of the same color in parallel. Before a processor starts the update of a certain color, it exchanges the border points of the opposite color with its neighbors. Figure 1 illustrates the use of SOR over different processors.

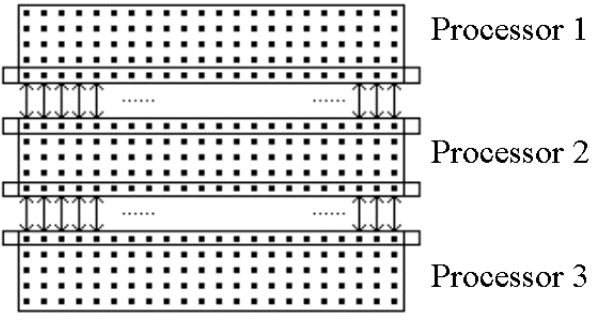

Fig. 1. Successive Over Relaxation

3 Analysis of Fluctuations in Processing Speeds

To characterize the fluctuations in processor speeds in a grid environment, we collected data about the processor speeds and the communication times by doing 10 runs of Red-Black SOR with a grid size of 5000×1000. Interrupted runs were omitted. One run consists of 1000 iterations, from which there are 3 warming up, 994 regular, and 3 cooling down iterations. Every iteration has two phases (see Section 2), which leads to 1988 data lines per run. To increase the running times such that parallellisation improves performance we repeated each iteration 50 times. This corresponds to a grid size of $25 \cdot 10^4 \times 10^3$. Table 1 lists the sites and node names we used during the experiments. For every run we used 4 independently chosen sites from that table. We collected data about calculation times and receive times of each node, and wait times and send times between all nodes. The calculation time is the time a node uses to compute one calculation of one iteration, the wait time is the time a node has to wait for data of its neighbors before it can do a new step, the send time is the time a node uses to send all the relevant data to its neighbors and receive the acknowledgement, and the receive time is the time a node uses to load the relevant data of its neighbors from the received-data table. We do not run other applications on the same nodes during our runs to create changing load on the processors.

Figures 2 and 3 show the calculation times as a function of the iteration number for a set of 250 iterations for different sites. Figures 4 and 5 show the results for the send times. The receive times were found to be negligible (mostly less than 0.5 ms).

The results presented in Figures 2 to 5 reveal a number of interesting phenomena. First, we observe that fluctuations in the calculation times and the send times are considerable. We also observe fluctuations on multiple time scales. On the one hand there are short-term fluctuations in both the calculation times and the send times, on the order of seconds. On the other hand, we observe long-term fluctuations, as can be seen from Figure 2. These fluctuations are presumably caused by a changing load at the processor. The long-term fluctuations in calculation times suggest that reduction of the running times can be realized by dynamically allocating more tasks to relatively fast processors. Second, Figures 2–5 show that the burstiness in the send times is larger than in the calculation times. We observe that the send times do not have a long-term effect, whereas the calculation times often show huge long-term fluctua-

Table 1. Used nodes in our experiments

Site	*Abbreviation*	*Nodename*
University of Utah	utah1	planetlab1.flux.utah.edu
University of Washington	wash1	planetlab01.cs.washington.edu
University of Arizona	arizona1	planetLab1.arizona-gigapop.net
California Institute of Technology	caltech1	planlab1.cs.caltech.edu
University of California, San Diego	ucsd1	planetlab1.ucsd.edu
Boston University	boston1	planetlab-01.bu.edu

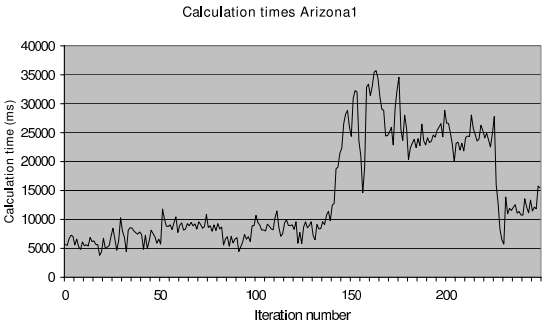

Fig. 2. Calculation times of 250 iterations in Arizona1

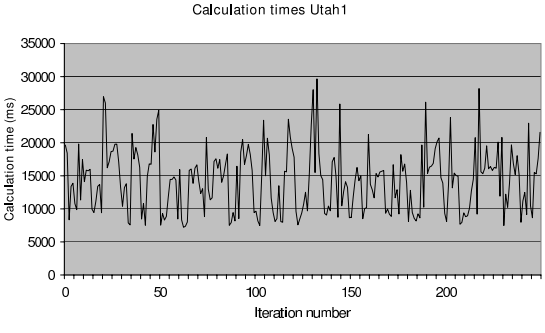

Fig. 3. Calculation times of 250 iterations in Utah1

tions. This observation also suggests that there is a great potential reduction in calculation times, which can be achieved by adapting the load according to the current speeds of the processors. Note that those findings correspond with the results about fluctuations of CPU availability in time-shared unix systems [11]. In this paper we do not investigate the causes of those fluctuations, but we are interested in how to deal with them. That corresponds with the idea that it will be hard to retrieve causes of fluctuations in the future grid.

4 Load Balancing Strategies

Load balancing is an effective means to reduce running times in a heterogeneous environment with fluctuating processing speeds. In this section we quantify the feasible reduction in running time by using different load balancing strategies. We consider two types of load balancing strategies: static and dynamic.

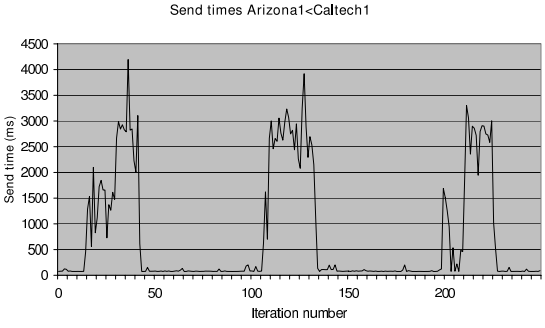

Fig. 4. Send times of 250 iterations from Caltech1 to Arizona1

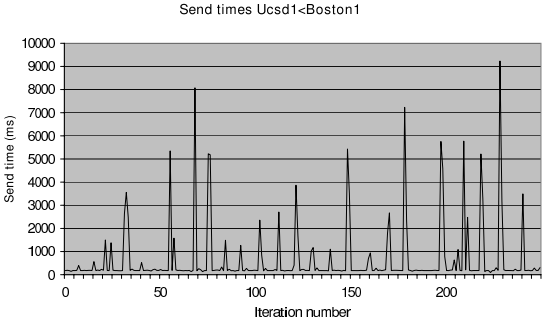

Fig. 5. Send times of 250 iterations from Boston1 to Ucsd1

4.1 Definitions

Static Load Balancing (SLB) strategies use a number of "cold iterations" to estimate the average processor speeds, in order to balance the load. Define

$S(n) :=$ total running time with SLB using the average calculation times of the first n iterations.

Note that the special case $S(0)$ corresponds to the running time of a run without load balancing, that is, with equal loads.

Berman et al. [12] show that forecasting the performance of the network is useful for Dynamic Load Balancing (DLB). Several prediction methods have been developed for network performance and CPU availability [13]. We use the method of Exponential Smoothing (ES) to predict calculation times (see also [14]). ES appears to be a very simple and effective method to reduce running times. ES is a forecasting method that on the one hand filters out outliers in the data, and on the other hand reacts quickly to long-term changes. Denote by y_n the realization of the n-th iteration step, and let $\hat{y}_n$ denote the prediction of y_n. Then ES is based on the following recursive scheme:

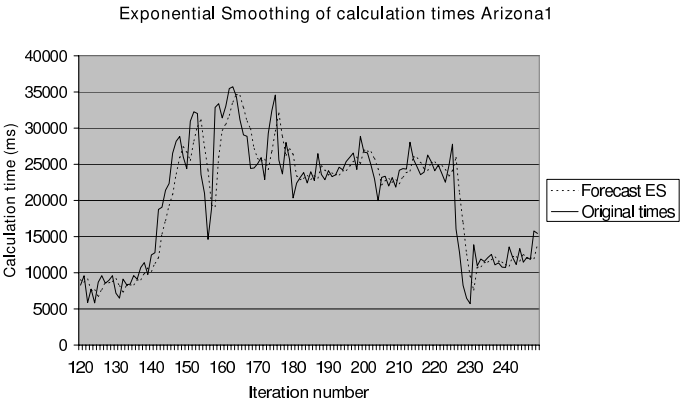

Fig. 6. Exponential Smoothing of calculation times of Arizona1

$$\hat{y}_n = \alpha y_{n-1} + (1-\alpha)\hat{y}_{n-1} \ . \tag{1}$$

Figure 6 shows the calculation times as a function of the iteration sequence number. The results show that our ES predictor performs very well: even the high fluctuations are well tracked by the forecasts.

In the context of the dynamic load balancing strategies the ES-based prediction $\hat{y}_n$ represents the predicted calculation time in the n-th iteration, $n = 1, 2, \ldots$.

If we want to change the load of the processors in the case of Dynamic Load Balancing we have to move rows in the grid from one processor to the other. To avoid excessive communication we introduce a parameter T indicating how often we move rows around. Define for the ES-based Dynamic Load Balancing strategies:

$$D(\alpha, T) := \text{running time with DLB using ES with parameter } \alpha, \text{ and load balancing every } T \text{ iterations.}$$

4.2 Calculation of Running Times

As described in Section 3 we collected data about calculation and send times by doing 10 runs of the Red-Black SOR. In the subsection before we defined several Static and Dynamic Load Balancing strategies. In this subsection we will describe the calculation methods we used to generate estimates of the running times of runs using the optimal static and dynamic strategies from the datasets of the original runs.

We assume a linear relation between the number of tasks (in SOR: the number of rows) and the calculation times of those tasks together, and also a linear relation between the amount of data (in SOR: the number of rows) sent by the application, and the total send time. We also assume that the overhead involved in load balancing is negligible in the long time scale (of minutes) considered here, because calculation times are significantly higher than the overhead.

To start, we explain how we compute an estimation of the lowest possible running time under a DLB strategy, denoted by D^*. To this end, we use the measured calculation times in the original run to estimate the lowest possible calculation time with the optimal DLB strategy. Let $nolb\_rows_j$ be the number of rows assigned to processor $j = 1, \ldots, P$ in the original run, and let $nolb\_calc_{i,j}$ be the measured calculation time on processor j for the ith iteration, $i = 1, \ldots, I$. Then, $nolb\_rows_j / nolb\_calc_{i,j}$ is an approximation of the number of rows that can be executed during iteration i on processor j per time unit. The total processing rate for iteration i is therefore $\sum_j nolb\_rows_j / nolb\_calc_{i,j}$, and the time it takes under a perfect DLB strategy to do iteration i is therefore

$$D^*\_calc_i = \frac{\sum_{j=1}^{P} nolb\_rows_j}{\sum_{j=1}^{P} nolb\_rows_j / nolb\_calc_{i,j}} = \frac{(\sum_{j=1}^{P} nolb\_rows_j)(\prod_{j=1}^{P} nolb\_calc_{i,j})}{\sum_{j=1}^{P} (nolb\_rows_j \prod_{k \neq j} nolb\_calc_{i,k})} \quad . \tag{2}$$

Note that $D^*\_calc_i$ is the estimated calculation time for iteration i with the optimal dynamic load balancing strategy, assuming all processor speeds are known in advance, and that $D^*\_calc_i$ is the same for all processors. In this calculation we assumed that the overhead in realizing the dynamic load balancing is negligible.

Now we focus on the calculation of an estimation of the lowest possible running time under SLB, denoted by S^*. With respect to the dynamic situation, we compute the average processing rate over the whole run, and not the rate per iteration. This rate is given by

$$\frac{nolb\_rows_j}{\frac{1}{I}\sum_{i=1}^{I} nolb\_calc_{i,j}} \quad .$$

Thus the number of rows $S^*\_rows_j$ that have to be assigned to processor k under the SLB strategy is equal to

$$S^*\_rows_j = \sum_{k=1}^{P} nolb\_rows_k \frac{\frac{nolb\_rows_j}{\frac{1}{I}\sum_{i=1}^{I} nolb\_calc_{i,j}}}{\sum_{k=1}^{P} \frac{nolb\_rows_k}{\frac{1}{I}\sum_{i=1}^{I} nolb\_calc_{i,k}}} \quad , \tag{3}$$

and we estimate that iteration i on processor j takes

$$S^*\_calc_{i,j} = \frac{S^*\_rows_j}{nolb\_rows_j} nolb\_calc_{i,j} \quad . \tag{4}$$

time units.

To calculate the running times of the Static and Dynamic Load Balancing strategies we first derive the number of rows, $S(n)\_rows_j$ and $D(\alpha, T)\_rows_{i,j}$ respectively, each processor j receives from the strategy in each iteration i. For this step we used the methods described in the previous subsection: for calculating $S(n)\_rows_j$ we used the first n iterations and for $D(\alpha, T)\_rows_{i,j}$ Exponential Smoothing. Next, with the following formulas we compute the new calculation times of the strategies for each processor j in iteration i:

$$S(n)\_calc_{i,j} = \frac{S(n)\_rows_j}{nolb\_rows_j} nolb\_calc_{i,j} \ , \tag{5}$$

$$D(\alpha,T)\_calc_{i,j} = \frac{D(\alpha,T)\_rows_{i,j}}{nolb\_rows_j} nolb\_calc_{i,j} \ . \tag{6}$$

Above, we explained how we calculated the new calculation times for each strategy in each iteration. Finally, we put those new calculation times and the send times of the original run in a plain model to derive the new wait times and the estimated running times of the different strategies.

5 Performance Comparison: Experimental Results

To compare the performance under different load balancing strategies, we have estimated the running times under a variety of static and dynamic load balancing schemes. We define the speedups of $S(n)$, $D(\alpha,T)$, S^* and D^* as the number of times those strategies are faster than the run without load balancing:

$$\text{speedup } S(n) := \frac{S(0)}{S(n)} \ , \tag{7}$$

$$\text{speedup } D(\alpha,T) := \frac{S(0)}{D(\alpha,T)} \ , \tag{8}$$

$$\text{speedup } S^* := \frac{S(0)}{S^*} \ , \tag{9}$$

$$\text{speedup } D^* := \frac{S(0)}{D^*} \ . \tag{10}$$

Table 2 shows the speedups that can be made by load balancing on the basis of ES predictions, compared to the case with no load balancing, for a variety of load balancing strategies. Based on extensive experimentation with the value of α, we found that a suitable value of α is 0.5.

The results shown in Table 2 lead to a number of interesting observations. First, we observe that there is a high potential speedup by properly reacting to fluctuations of processing speeds by dynamic load balancing. The potential speedup is shown by the speedup of D^* in Table 2; in the optimal dynamic load balancing case it is possible to obtain a speedup of 2.5 minus the overhead for the running times, and in 20% of the iterations even more than 3.6. Second, we observe that despite the inaccuracy in the predictions of the calculation times the speedup factor by applying dynamic load balancing is still close to the "theoretical" optimum. Even load balancing every 200 iterations, which relatively causes almost no extra overhead compared to the total running time, leads to an average speedup of 2.0 compared to the case of no load balancing. Third, we also observe that even if a better static load balancing scheme is used as a benchmark, the speedup factor realized by implementing a dynamic load balancing scheme is still significant.

Table 2. Relative improvements compared of different load balancing strategies (compared to no load balancing)

LB strategy	*Mean speedup of 10 runs*
$S(0)$	1.0
$S(1)$	1.2
$S(10)$	1.2
$S(20)$	1.3
S^*	1.9
$D(0.5, 1)$	2.5
$D(0.5, 2)$	2.5
$D(0.5, 3)$	2.4
$D(0.5, 4)$	2.4
$D(0.5, 5)$	2.4
$D(0.5, 10)$	2.3
$D(0.5, 20)$	2.3
$D(0.5, 30)$	2.3
$D(0.5, 40)$	2.3
$D(0.5, 50)$	2.2
$D(0.5, 100)$	2.2
$D(0.5, 200)$	2.0
$D(0.5, 300)$	2.0
$D(0.5, 400)$	1.8
$D(0.5, 500)$	1.7
D^*	2.5

6 Conclusions and Further Research

The results presented in this paper raise a number of challenges for further research. First, the results demonstrate the importance of effectively reacting to randomness in a grid environment. The development of robust grid applications is a challenging topic for further research. Second, in the results presented here we have focused on the fluctuations in processing speeds. However, in data-intensive grid applications the fluctuations in the available amount of network capacity may be even more important than fluctuations in processor speed. To this end, extensive experiments need to be performed to control changing network capacities. Third, more research has to be done on the aspect of selecting the best predicting methods for processor speeds. With those methods general dynamic load balancing algorithms for regularly used parallel applications have to be developed. Finally, in this paper we focus on the SOR application, which has a relatively simple linear structure (see Figure 1). One may suspect that even larger improvements of the running times may be obtained for more complex computation-intensive applications with more complex structures, which is an interesting topic for further research.

Acknowledgements

We are indebted to Mathijs den Burger, Thilo Kielmann, Henri Bal, Jason Maassen and Rob van Nieuwpoort for their useful comments.

References

1. (http://www.planet-lab.org)
2. Banicescu, I., Velusamy, V.: Load balancing highly irregular computations with the adaptive factoring. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS). (2002)
3. Attiya, H.: Two phase algorithm for load balancing in heterogeneous distributed systems. In: Proceeding of the 12th Euromicro conference on parallel, distributed and network-based processing. (2004)
4. Shirazi, B.A., Hurson, A.R., Kavi, K.M.: Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE CS Press (1995)
5. Zaki, M.J., Li, W., Parthasarathy, S.: Customized dynamic load balancing for a network of workstations. Journal of Parallel and Distributed Computing **43** (1997) 156–162
6. Nemeth, Z., Gombas, G., Balaton, Z.: Performance evaluation on grids: Directions, issues and open problems. In: Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing. (2004)
7. Banicescu, I., Liu, Z.: Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes. In: Proceedings of the High Performance Computing Symposium (HPC). (2000) 122–129
8. Cariño, R.L., Banicescu, I.: A load balancing tool for distributed parallel loops. In: International Workshop on Challenges of Large Applications in Distributed Environments. (2003) 39–46
9. Evans, D.J.: Parallel SOR iterative methods. Parallel Computing **1** (1984) 3–18
10. Hageman, L.A., Young, D.M.: Applied Iterative Methods. Academic Press (1981)
11. Wolski, R., Spring, N.T., Hayes, J.: Predicting the CPU availability of time-shared unix systems on the computational grid. Cluster Computing **3** (2000) 293–301
12. Berman, F.D., Wolski, R., Figueira, S., Schopf, J., Shao, G.: Application-level scheduling on distributed heterogeneous networks. In: Proceedings of the 1996 ACM/IEEE conference on Supercomputing, ACM Press (1996) 39
13. Wolski, R.: Forecasting network performance to support dynamic scheduling using the Network Weather Service. In: HPDC. (1997) 316–325
14. Shum, K.H.: Adaptive distributed computing through competition. In: Proceedings of the International Conference on Configurable Distributed Systems, IEEE Computer Society (May 1996) 200–227