Dynamic Load Balancing Experiments in a Grid*

Menno Dobber[a], Ger Koole[a], Rob van der Mei[a,b]

[a] *Vrije Universiteit Amsterdam, The Netherlands*
[b] *CWI Amsterdam, The Netherlands*
{*amdobber,koole,mei*}*@few.vu.nl*

Abstract

Connected world-widely distributed computers and data systems establish a global source of processing power and data, called a grid. Key properties of a grid are the fact that computers providing processing power may connect and disconnect at any time, and that demands for processing power may highly fluctuate over time. This has raised the need for the development of applications that are robust against changing circumstances. In [4] the impact of fluctuations in processing speeds on running times has been investigated, and it was found that dynamic load balancing methods provide a promising means to deal with the ever-changing environment in the grid. In this paper we demonstrate with extensive experiments in a real grid environment, Planetlab, that dynamic load balancing based on predictions via Exponential Smoothing indeed lead to significant reductions in running times of parallel applications in a randomly changing grid environment.

1. Introduction

Although grids are the successors of distributed computing environments (DCEs), these two environments are fundamentally different. A property of a DCE environment is the predictability: resources are homogeneous and reservations have to be made to use the nodes, which leads to a guaranteed amount of processing capacities. Unlike the DCEs, a grid environment is extremely *unpredictable*: processor capacities are different and usually unknown, computers may connect and disconnect at any time, and their speeds may change over time. For these reasons, it is a challenge to develop parallel programs that are *robust* against changes in the environment and as such are suitable for execution in a grid environment. Over the years, much research has been done on grid computing. Initially, in the absence of publicly available grid environments, the research was mainly theoretically oriented. Recently, a variety of grid test beds have been developed (e.g., Planetlab [1]). This enables us to perform extensive experiments with grid applications, to investigate how well grid applications perform in practice, and how they can be improved. In this paper, we provide an experimental analysis of the performance of grid applications, and assess the actual improvements that can be obtained by implementing dynamic load balancing (DLB) schemes.

Variations in the available resources (e.g., computing power, bandwidth) may have a dramatic impact on the running times of parallel applications. Over the past few decades, performance of parallel applications has received much attention in the research community. Due to the difficulty of analyzing realistic variations, most of the fluctuations were imitated and therefore controllable (see for example [3]), while performance experiments were performed in a controllable DCE. However, the variations in grid environments are not manageable, which limits the applicability of these results in a real grid environment. Alternatively, mathematicians typically build stochastic models to describe the performance of resources, the network and dependencies related to runs of parallel programs. They create algorithms to decrease running times, and analyze these algorithms mathematically [2, 8, 9]. For some situations such a mathematical approach is useful. However, in practice a lar-

*Accepted at CCGrid2005

ge number of simplifying assumptions are needed to performance a mathematical analysis, which limits its applicability. As another alternative, DCE experts produce strategies that are valuable for computational grids, connected clusters of computational nodes. However, as a result of the difference between fluctuations for grid environments and computational grids, the effectiveness of these strategies in a grid environment is questionable [7]. These observations stress the importance for an *integrated* analysis of grid applications, combining the three above-mentioned approaches, to analyse properties, dependencies and distributions within a grid environment, and to implement the ideas in a real grid environment and verify how the methods perform in practice.

The impact of fluctuations in processing speeds on running times in a grid environment has been investigated in [4]. In that paper, Exponential Smoothing (ES) was shown to be a good predictor for processing power. Moreover, DLB methods based on this ES-predictor were found to be a promising means to improve performance. In this paper, we present the results of extensive load balancing experiments on a real grid test bed, called Planetlab [1]. The experiments were performed with the classical Successive Over Relaxation (SOR) application, which is particularly suitable for running in a parallel computing environment. The results demonstrate that DLB methods consistently lead to a significant reduction in running times.

This paper is organized as follows. In Section 2 we describe the SOR application and the grid test bed Planetlab on which we perform our experiments, and in Section 3 the use of ES is discussed. In Section 4 the experimental results are discussed in detail. Finally, in Section 5 we summarize the conclusions and address a number of topics for further research.

2. Experimental Setup

To carry out experiments with parallel applications in a realistic setting, the test bed must have the following key characteristics of a grid environment: (1) processor capacities often differ, (2) processor loads change over time, (3) processors are geographically distributed, and (4) network conditions are highly unpredictable. We chose Planetlab [1], a commonly used grid test bed environment that fulfils those conditions. At the time of our experiments, version 2.0 was installed on the nodes. Planetlab is an open, processor-shared globally distributed network for developing and testing planetary-scale network services.

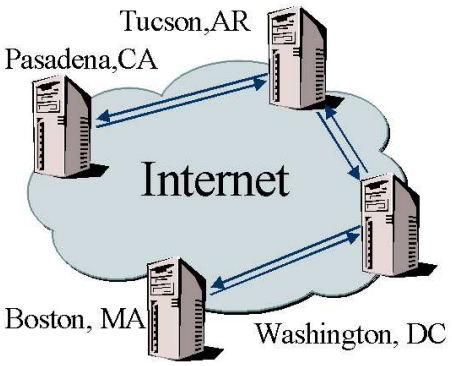

Figure 1. Nodes on Planetlab, used for experiments

Ideally, experiments should be performed both with a small number of nodes and with a very large number of nodes. To obtain statistically significant results, the experimental results need to be reproducible. In practice, however, the most commonly used grid test beds are not yet mature enough, and the availability of many nodes is limited. For this reason, we have chosen to conduct our experiments with four sites. On the one hand, the number of four sites is large enough to demonstrate a significant speedup factor by DLB. On the other hand, this number is small enough to reproduce experiments with the same set of nodes within a reasonable time frame. We used two sets of four nodes of Planetlab to conduct our experiments. Set 1 consists of the nodes Pasadena (CA), Tucson (AR), Washington (DC) and Boston (MA), and set 2 consists of Vancouver (BC), San Diego (CA), Salt Lake City (UT) and Chicago (IL). The nodes were connected by the Internet via a linear structure, as shown by Figure 1.

To demonstrate the improvements that can be made by DLB, the parallel application must have dependencies between its successive iterations, which is common for parallel applications. We have conducted our experiments with the Successive Over Relaxation (SOR) application. SOR is an iterative method that is valuable in solving Laplace equati-

ons [5]. Our implementation of SOR performs calculations with a two-dimensional discrete state space $M \times N$, a grid of points. Logically, each point in the grid has at most four neighbors. In each iteration each point takes 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 [6]. The point 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 point 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. This update takes time, referred to as the calculation time. Before a processor starts the update of a certain color, it exchanges the border points of the opposite color with its neighbors. This amount of time is referred to as the send time. The total duration of an iteration is called the iteration time.

We have conducted experiments in two parts. In the first part, we used Red/Black SOR with a grid size of 5000 × 1000. Interrupted runs were omitted. One run consists of 2000 iterations. To increase the running times such that parallelisation improves performance we repeated each iteration 50 times. This corresponds to a grid size of $25 \cdot 10^4 \times 10^3$. The default load balancing scheme is referred to as Equal Load Balancing (ELB). ELB assumes no prior knowledge of processor speeds of the nodes, and consequently balances the load equally among the different nodes. To compare the effectiveness of our DLB implementation (see Section 3) to the ELB, we have performed extensive experiments. Ideally, experiments with and without DLB should be performed simultaneously. Unfortunately, in practice performing experiments simultaneously is not possible because of interference. Therefore, to make a fair comparison we alternatingly ran the two implementations under comparable circumstances: each day at 09:00 CET we started one of the two implementations, and the next day we started the other one. To obtain statistically relevant results, we ran the two versions of SOR on set 1 (see above) of Planetlab sites both as many as 30 times. In the second part, we performed experiments to investigate the dependence between the problem sizes and the speedup gained by implementing DLB. Therefore, we ran the two versions of the Red/Black SOR for grid sizes of 2500 × 1000, 5000 × 1000, 7500 × 1000, and 10000 × 1000. Those runs consisted of 1000, 500, 375, and 250 iterations, respectively. We repeated each iteration 50 times. We ran each grid size for both versions seven times on set 2 of Planetlab sites.

3. Implementation of Dynamic Load Balancing

In DLB schemes from time to time decisions are made to update the balancing of the loads on the basis of predictions of the processing speeds. We used the Exponential Smoothing (ES) technique to obtain these predictions, because in [4] the ES technique with parameter $\alpha = 0.5$ was found to be a good predictor of processor performance. ES appears to be a simple and usable method in load balancing strategies. On the one hand ES filters outliers in the data, and on the other hand adapts the predictor quickly to long-term changes. Denote by y_n the realization of the n-th iteration step, and let $\hat{y}_n$ denote the forecast of y_n. The ES recursive formula we use to predict is:

$$\hat{y}_n = 0.5y_{n-1} + 0.5\hat{y}_{n-1}. \quad (1)$$

To investigate whether DLB based on ES is indeed an effective means to react on fluctuations in load or performance of processors we have implemented it in our SOR application. Our implementation of the load balancing step is as follows. At the end of each iteration the processors predict their processing speed for the next iteration. After every N iterations the processors send their prediction to processor 0, the DLB scheduler. Subsequently, this processor calculates the "optimal" load distribution given those predictions and sends relevant information to each processor. The load distribution is optimal when all processors finish their calculation exactly at the same time. Therefore, it is "optimal" when the number of rows assigned to each processor is proportional to its predicted processor speed. Finally, all processors redistribute the rows. The total load balancing step takes around one third of

the total time of one iteration (i.e., calculation and sending time).

Load balancing each single iteration is rarely a good strategy. On the one hand, the running time of a parallel application directly depends on the overhead of DLB, and therefore it is better to increase the number of iterations between two load balancing steps. On the other hand, less load balancing leads to an imbalance of the load for the processors for sustained periods of time, due to significant changes in processing speeds. In [4] we present the theoretical speedups in running times when using load balancing compared to equal load balancing (ELB), given that the application load balances every N iterations, but without taking into account the overhead. Based on those speedups and the load balancing overhead addressed above, a suitable value of N was found to be 10.

4. Experimental results

To investigate the speedup that can be obtained by implementing DLB compared to the default ELB we have performed numerous experiments. The results of these experiments are outlined in this section. First, to assess the potential benefits that can be obtained by using DLB in Section 4.1 we analyze the stochastic behavior of the calculation times at different time scales at the different nodes. Moreover, we evaluate the effectiveness of the ES technique to predict the calculation times over these time scales. Second, in Section 4.2 we discuss the experimental results for the DLB and ELB.

4.1. Stochastic behavior of the calculation times

Figures 2 to 5 show the calculation time for the successive iterations for a selection of two sites from both sets used during the experiments. The results lead to a number of interesting observations. First, for each of the nodes we observe fluctuations at different times scales. More precisely, we observe fluctuations at a short time scale of a few iterations (typically in the range 1 to 10 iterations), which roughly corresponds to several minutes. These short-term fluctuations are bursty and rather unpredictable. In addition, we observe fluctu-

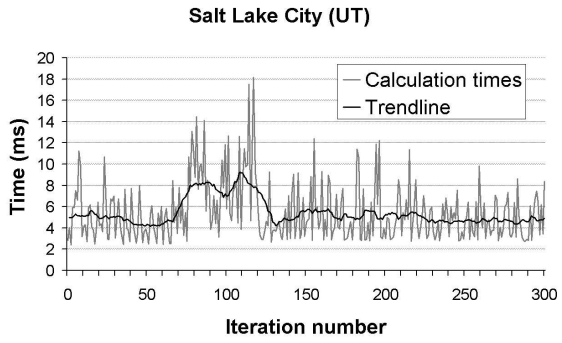

Figure 2. Calculation times in Salt Lake City (UT)

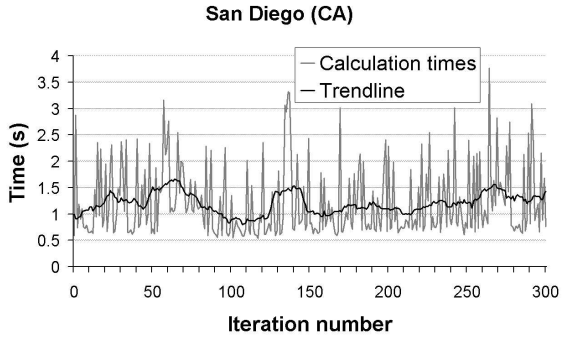

Figure 3. Calculation times in San Diego (CA)

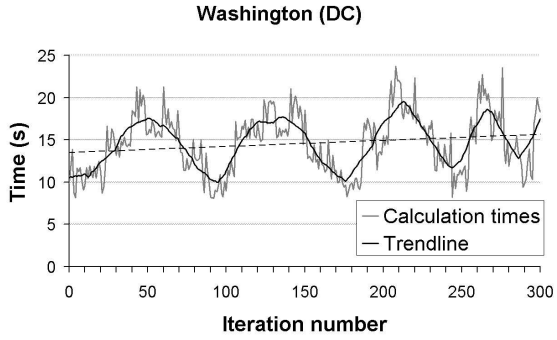

Figure 4. Calculation times in Washington (DC)

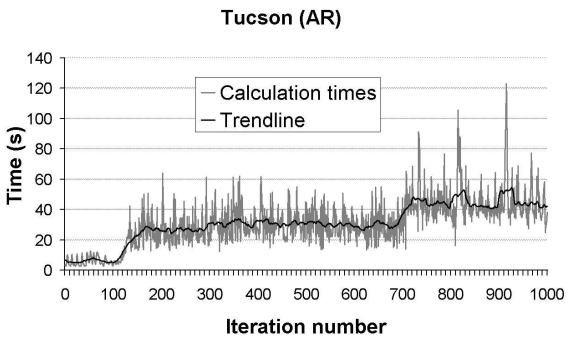

Figure 5. Long- and short-term calculation-time fluctuations in Tucson (AR)

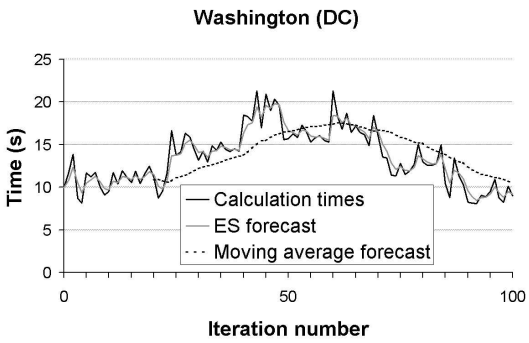

Figure 6. ES predicting calculation times in Washington (DC)

ations on a longer time scale of several tens of iterations, which is roughly on the order of tens of minutes. Moreover, the results in Figure 5 suggest fluctuations at an even longer time scale of several hundreds of iterations, which corresponds to several hours in the time domain. Second, comparing the results depicted in Figures 2 to 5 we observe a strong heterogeneity between the different patterns, with significant differences in the burstiness on the short time scale and the fluctuations on the longer time scales. For example, the short-term behavior in Figures 2 and 3 is much more bursty than the short-term behavior in Figure 4. Moreover, for the longer-term fluctuations we observe a heterogeneity of patterns, including periodically changing behavior (see Figure 4) and randomly changing behavior (see Figures 2, 3 and 5).

To cope with this strong heterogeneity and randomness, efficient and robust DLB techniques should be based on well-performing prediction methods. In [4] we argued that the prediction technique based on ES (see Section 3 for details) seems appropriate. To investigate whether this is indeed the case, Figure 6 (which is based on the same set of data used in Figure 4) shows (1) the measured calculation times, (2) the forecasts based on ES, and (3) the moving average over the last 20 iterations. Figure 6 shows that the moving average predicts the fluctuations over the longer times scales very well, but fails to provide accurate predictions of the fluctuations on the shorter time scale. Moreover, we observe that the ES technique does capture the fluctuations on *both* the longer *and* the shorter time scale rather well. Extensive ES-based test runs have shown that ES consistently outperforms the moving average prediction technique, and as such seems to be an excellent basis for the development of robust DLB schemes in an ever-changing and heterogeneous grid environment. For this reason, we have implemented DLB on the basis of ES as the prediction technique.

4.2. Experiments with DLB and ELB

In this subsection we demonstrate the effectiveness of implementing DLB based on ES. To this end, we have performed 30 runs with the original ELB SOR implementation and 30 runs of the DLB implementation in Planetlab. To make a fair comparison, the runs were alternatingly performed with ELB and DLB. The odd run numbers correspond to DLB-based experiments, and the even run numbers are based on ELB. Figure 7 shows the running times for these experiments. The results plotted in Figure 7 show that the DLB-based experiments are significantly faster than their ELB-counterparts, consistently over all experiments (except for a single outlier in run 20). Interestingly, the DLB strongly outperforms ELB independent of the actual running times. The average speedup factor by using DLB instead of ELB was found to be roughly a factor of 1.8. This confirms the predictions on the basis of a theoretical analysis addressed at the end of Section 3.

To analyze the speedup between DLB and ELB in more detail, Figure 8 shows the evolu-

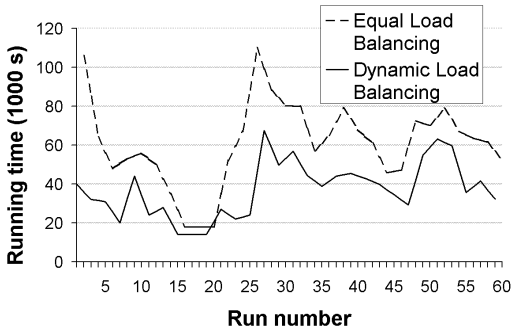

Figure 7. Running times for SOR based on DLB compared to ELB

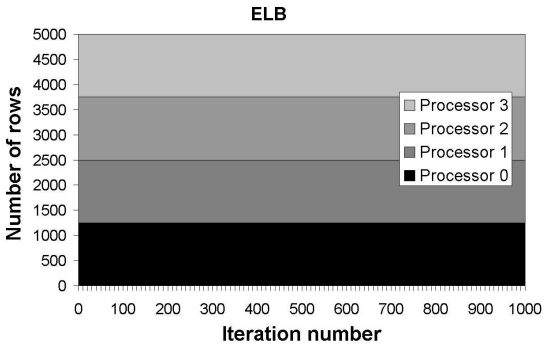

Figure 9. Number of rows for each processor in a ELB run

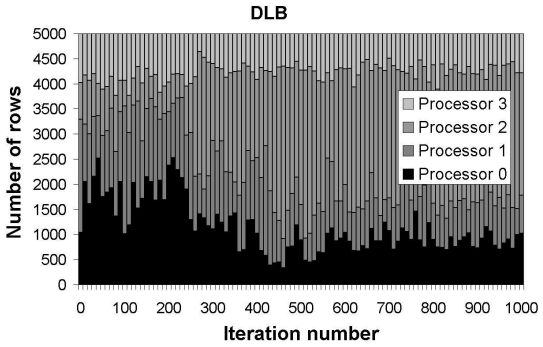

Figure 8. Number of rows for each processor in a DLB run

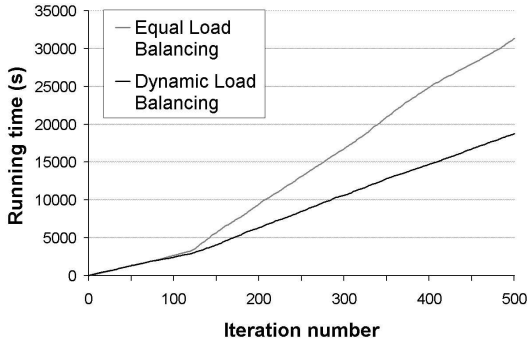

Figure 10. Cumulative running time as a function of the iteration number

tion of the load distribution over the different nodes for both the DLB and the ELB scheme. More precisely, a representative development of the number of rows assigned to the nodes is shown. For the DLB case we observe both short- and long-term changes in the number of rows during a run, which are caused by dynamic reactions on the short- and long-term changes in calculation times. The results also show the drawback of implementing Static Load Balancing (SLB) schemes, where the load is balanced statically on the basis of the first M iterations. A key problem is to find a suitable value for M, which is based on the following trade-off. If M is too large, then the benefit of SLB is marginal by definition. If M is too small, then the estimates of the processing speeds of the nodes, and hence of the "optimal" load distribution, are unreliable. For example, the results in Figure 8 show that the optimal load distribution on the basis of DLB is roughly $12\%, 18\%, 54\%$ and 16%, for processor 0 to 3, respectively. However, if SLB were used with $M \leq 250$ then the SLB weights would be roughly $20\%, 16\%, 34\%$ and 30%, respectively.

The next question is how the speedup based on DLB compared to ELB evolves over time. To this end, Figure 10 shows the cumulative running time as a function of the number of iterations for DLB and ELB, respectively, for the same experiment as in Figure 8. Figure 10 shows that at the beginning of the run DLB is not faster than the original implementation ELB. However, after about 120 iterations, the ELB run tends to slow down significantly, whereas the DLB slows down only marginally. This observation can be explained from Figure 8 as follows. During the first (say) 120 iterations, processors 0 and 1 were relatively fast compared to processors 2 and 3. However, around iteration 120 for some unknown reason processors 0, 1 and 3 were slowing down possibly caused by background load, whereas the

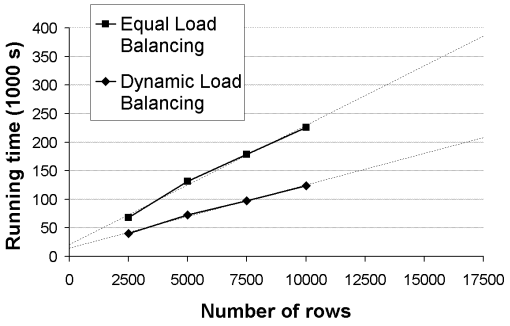

Figure 11. Running times as a function of the number of rows

processing speed of processor 2 did not change significantly. Consequently, the DLB scheme dynamically assigned additional rows to processor 2, while the static ELB scheme (see Figure 9) did not. In this way, the DLB scheme was found to properly react to changes in the effective processor speeds, and as such outperformed ELB signifantly.

Another interesting question is how the running times achieved by implementing DLB depends on the problem size. To this end, we have performed experiments with DLB and ELB and with different problem sizes with 1000 columns and N_{row} rows. The experiments have been repeated seven times in order to obtain reliable estimates. Figure 11 shows the average running time as a function of N_{row}, for $N = 2500$, 5000, 7500 and 10000. Confidence intervals are not presented here for ease of the discussion. This figure shows that the running time increases nearly linearly in the number of rows, for both ELB and DLB. More precisely, based on a simple least-square estimation method we obtain the following approximate expression for the running times RT_{ELB} and RT_{DLB} (in seconds) as a function of the number of rows:

$$RT_{ELB} = 20.8 * N_{row} + 21138, \quad (2)$$

$$RT_{DLB} = 11.1 * N_{row} + 14363. \quad (3)$$

The offset for ELB consists of send and wait times, which are independent of the number of rows. The offset for DLB also consists of send times and wait times, but in the DLB case the wait times are smaller than in the ELB case, because DLB is able to react to temporary imbalance causing larger wait times. In addition, the DLB-offset contains the overhead involved performing load balancing actions. Table 1 shows the speedup factor for different values of N_{row}.

N_{row}	*Problem size*	*Average speedup*
2500	2500×1000	1.71
5000	5000×1000	1.82
7500	7500×1000	1.82
10000	10000×1000	1.83

Table 1. Speedup factors for different problem sizes

Table 1 demonstrates that, because of the above-mentioned differences in the offsets for ELB and DLB, the speedup depends on the problem size. More precisely, it follows directly from (2) and (3) that in the current experimental setting the speedup factor converges to the following constant when N_{row} grows to infinity:

$$\lim_{N_{row}\to\infty} \frac{RT_{ELB}}{RT_{DLB}} = \lim_{N_{row}\to\infty} \frac{20.8N_{row}+21138}{11.1N_{row}+14363} = \frac{20.8}{11.1} = 1.87.$$

Acknowledgments: The authors would like to thank Henri Bal, Thilo Kielmann and Mathijs den Burger for their useful comments.

5. Conclusions

We have investigated the impact of implementing DLB schemes on the running times in a grid environment. Extensive experimentation in the testbed environment PlanetLab have led to the following conclusions. (1) A significant speedup factor of on average 1.8 can be consistently achieved by implementing DLB instead of the default ELB scheme. (2) Using DLB based on ES-predictions of the running times provides an effective means to react to changes in the performance of the resources used by a parallel application. (3) The effective calculation times for the different processors at grid nodes fluctuate over different time scales. On the short time scale the calculation time may be highly bursty. On the longer time scales the calculation time may follow periodic or random patterns. (4) The relation between the running time and the problem size is approximately linear.

Finally, we address a number of challenges for further research. First, an interesting and important question in running parallel applications in a grid environment is "When do we need to balance loads?". In this paper the DLB-actions were performed each $N = 10$ iterations, partly based on the theoretical model discussed in [4]. However, due to the random nature of the grid environment, one may expect that more efficient load balancing schemes can be achieved by allowing load balancing actions to be performed at any moment, according to some dynamic algorithm that optimally balances the "cost" of load balancing actions and the benefits in quickly reacting to load changes. Second, in this paper we have performed experiments with the SOR application. SOR has a specific linear structure (see Figure 1). The question arises to what extend the results presented in this paper are applicable to other parallel applications, especially with a non-linear structure. In depth-analysis of parallel applications with a non-linear structure is a challenging topic for further research. Third, to develop optimal load balancing schemes advanced and accurate predictions of the calculation times are needed. To this end, the development of stochastic models (for the measurements shown in Figures 2 to 5), emcompassing the effect of fluctuation over different time scales (e.g., based on Fourier analysis, or the theory of multi-fractals) may be extremely useful, and an interesting topic for further research.

Referenties

[1] http://www.planet-lab.org.

[2] H. Attiya. Two phase algorithm for load balancing in heterogeneous distributed systems. In *Proceeding of the 12th Euromicro conference on parallel, distributed and network-based processing*, page 434. IEEE, 2004.

[3] I. Banicescu and V. Velusamy. Load balancing highly irregular computations with the adaptive factoring. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 195. IEEE Computer Society, 2002.

[4] A. M. Dobber, G. M. Koole, and R. D. van der Mei. Dynamic load balancing for a grid application. In *Proceedings of HiPC 2004*, pages 342–352. Vrije Universiteit, Springer-Verslag, December 2004.

[5] D. J. Evans. Parallel SOR iterative methods. *Parallel Computing*, 1:3–18, 1984.

[6] L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, 1981.

[7] Z. Nemeth, G. Gombas, and Z. Balaton. Performance evaluation on grids: Directions, issues and open problems. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2004.

[8] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE CS Press, 1995.

[9] M. J. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing*, 43(2):156–162, 1997.