Load Management for Load Balancing on Heterogeneous Platforms: A Comparison of Traditional and Neural Network Based Approaches

Bettina Schnor, Stefan Petri, Horst Langendörfer

Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig,
Bültenweg 74/75, D-38106 Braunschweig, Germany
schnor@ibr.cs.tu-bs.de

Abstract. In this paper we compare simple load metrics with neural networks which have been trained to predict the expected delay of an application from the sampled load informations. The results show that the proposed load metric performs well in heterogeneous environments. Further, neural networks can improve the performance of load balancing facilities.

1 Introduction and Definition of Load Metrics

Clusters of workstations are an increasingly popular platform for parallel computing. Some of the main differences to 'real' parallel computers, as far as resource management is concerned, are (configurational) heterogeneous nodes and (interactive) users running their applications concurrently. The observed workload is also heterogeneous and consists of jobs with highly varying service demands, long running batch jobs, and interactive applications with a high percentage of I/O-operations.

Load metrics for homogeneous systems have been examined by e.g. Ferrai and Zhou [1] and by Kunz [3]. Here, we present and compare two load metrics which aim to predict the delay of an application under the given load situation.

Definition of Delay Factors To minimize the response times, we want to assign an arriving job to the processor with most free processing capacity. In a heterogeneous system, this does not only depend on the current load of the machines, but also on the speed characteristics of the machines.
The *delay factor* of an application on machine M_i under load $load_i$ is defined as

$$delay(M_i) := \frac{runtime(M_i, load_i)}{runtime(M_{slow}, 0)},$$

where M_{slow} denotes the machine architecture with the lowest processing capacity in the system. When there is no other load in the system, this delay is equal to the *speed factor* α_i of machine M_i:

$$\alpha_i := \frac{runtime(M_i, 0)}{runtime(M_{slow}, 0)} .$$

However, this definition of the delay factors is of no use for computing them. While $runtime(M_{slow}, 0)$ is the result of one run for each application, $runtime(M_i, load_i)$ has to be calculated for every application, every machine,

and every load, which is impossible. Instead, we approximate the delay from the current load as

$$delay(M_i) := \alpha_i \, (1 + load_i) \ ,$$

where $load_i$ is an approximation for the current load.

Our benchmark for calculating the speed factors is LaTeX, one of the most popular applications in our computing department.

The simplest approximation for *load* is the CPU run queue length. Since the UNIX system statistics are gathered only every 5 seconds, we have decided to use the CPU run queue length smoothed over 30 seconds. The quality of this so-called YALB-delay[1] is evaluated in section 2.

Definition of Neuro–Delay Our aim is to investigate, whether neural networks can learn to predict the expected delay more accurately.

The analytic relation between resource statistic parameters and the resulting delay is unknown. The advantage of using a neural net is that the net can learn the relation between load statistics and delay during the training phase.

Mehra used neural networks to learn the relation between the resource statistics and the load index [4]. However, Mehra's method is not suited for real world use, because of the kernel modifications, and because the machines are needed exclusively for about ten days to sample the training patterns after changes in the configuration.

We trained a neural net for each of the 5 architectures listed in table 1. We measured about 1500 benchmark runs to get training and test patterns for the nets. A pattern consists of the resource statistic parameters, which are sampled before a benchmark is started, and the execution time. Our example applications were LaTeX, gcc, and the simulation program simuLa. The choice of the benchmarks was motivated by our system accounting.

	LaTeX	gcc	Mandelbrot	PovRay	dhrystone	MIPS
Sun SLC	1.00	1.00	1.00	1.00	1.00	1.00
Sun IPC	0.83	0.75	0.55	0.73	0.84	0.79
Sun ELC	0.70	0.69	0.34	0.52	0.56	0.53
Sun Classic	0.72	0.56	0.36	0.46	0.36	-
Sun SS2	0.53	0.45	0.26	0.42	0.46	0.44

Table 1. Speed factors of different applications.

We used Backpropagation nets with 3 layers [2]. At the input layer the load patterns are presented to the neural net. The output is the expected delay. During the training phase, the net compares its own calculated delay with the measured delay of the benchmark and tries to minimize its error.

Since the networks learn to predict the delay compared to a reference machine (Sun SLC), the trained networks can be used in a heterogeneous environment.

[1] The name is motivated by the use of the delay factor in our experimental YALB (Yet Another Load Balancing) System.

It should be mentioned that this approach avoids the need to calculate speed characteristics for the machines.

In order to reduce the training effort, we first did a correlation analysis of the sampled *vmstat* parameter values, and eliminated the insignificant ones. The significant parameters for all machines turned out to be the run queue length, system call rate, percentage of time spent in system mode, context switch rate, device interrupts per second, and the number of pages on the free list. Additionally, we used an average of the CPU run queue over 1 minute to smooth out inaccuracies of the UNIX statistics.

Earlier experiences [6] have shown that nets which do not predict the exact value, but classify the input patterns, are easier to train. For classification, each output node represents a class of delay factors. The center of a class is its representative.

When a load pattern is presented to the net, the two classes with the highest outputs out_1 and out_2 are determined. Let r_1 and r_2 be the corresponding class representatives. The Neuro-delay is then defined as the weighted mean: Neuro-delay $:= \frac{out_1 \cdot r_1 + out_2 \cdot r_2}{out_1 + out_2}$.

During our first training phase the network failure could not be minimized sufficiently. The neural nets seemed to be incapable to manage their task. We chose the percentage of user and system time of the applications as additional input parameters to give hints about the different runtime behavior of the applications. These values can be predicted by the user quite easily.

The results of the trained networks are presented in the next section.

2 Comparison of YALB– and Neuro–Delay

Both metrics predict the delay of an application on a machine compared to a reference machine. If they are used within a load balancing system, not the exact value of the delay is important, but the correct ranking of the machines.

2.1 Rank–Correlation–Coefficient

A suitable load metric ranks the machines in the same order as they would be ranked by the as yet unknown completion times of the application.

If we want to compare the ranking of the load metric with the *true ranking* due to completion times, we have to measure the completion time of an application on every machine. We used artificial load to guarantee the same load situation for each of the n measurements.

The different rankings are compared by Spearman's rank–correlation coefficient [5]:

$$r := 1 - \frac{6 \cdot \sum_i d_i^2}{n(n^2-1)},$$

where n is the number of ranked test cases equal to the number of machines in the system, and d_i is the difference between the two different ranks of the i^{th} test case.

Correlation for Training Patterns Here, we report the results for the training patterns used in section 1. For each tested application there are 107 samples (about 20 samples from each of the 5 architectures). Each sample is a triple consisting of the measured execution time, the YALB–delay, and the Neuro–delay (both calculated before the application was started).

	LaTeX	gcc	simuLan
Neuro	0.93	0.90	0.95
Yalb	0.91	0.90	0.93

	LaTeX	gcc	simuLan
Neuro	0.87	0.81	0.92
Yalb	0.87	0.65	0.83

Table 2. Correlation for Training Patterns (left), and the same for medium and high load only (right).

The correlation for all three benchmarks are very high for both metrics (see table 2 (left)). This means that the neural networks are trained well to predict the expected delay. On the other hand, the simple YALB–delay has also good ranking qualities.

Correlation under Medium and High Load In this experiment, we wanted to test the metrics under medium and high load. Therefore, we removed all samples from hosts which had none or only one job running concurrently to the benchmark. This results in 41 patterns for the gcc and 84 patterns each for the LaTeX and simuLan application.

The correlation coefficients decrease for both metrics (see table 2 (right)). The Neuro-delay behaves a little better than the YALB-delay.

2.2 Simulation of a Load Balancing System

The simulated load balancing system determines the fastest available machine in the system based upon the current delay values. Then an application is started on that machine, and the runtime is measured. This experiment is repeated 1000 times and all run times are summed up.

We re-used the samples from the rank correlation test for the simulation experiment. To simulate a system with n machines, we randomly choose n samples from this set. These n samples represent the load situations on n different machines. The machine with the smallest delay value is chosen for execution.

Then we compare the *measured* delays of all samples to determine the optimal placement of the application, i.e., the machine with the measured shortest runtime of the benchmark under the given load situation. This gives us the possibility not only to compare YALB– and Neuro–delay, but also to evaluate how far away the results are from the possible optimal placement.

The YALB-*slowdown* of a benchmark is defined as

$$\text{YALB-}slowdown := \frac{\sum runtime(M_{\text{YALB}}, load_{M_{\text{YALB}}}) - \sum runtime(M_{opt}, load_{M_{opt}})}{\sum runtime(M_{opt}, load_{M_{opt}})},$$

where M_{YALB} is the machine with the shortest expected YALB-delay, and M_{opt} is the machine with the measured shortest delay. The Neuro-*slowdown* is defined analogous.

Results for Training Patterns In case of the LaTeX application the YALB-

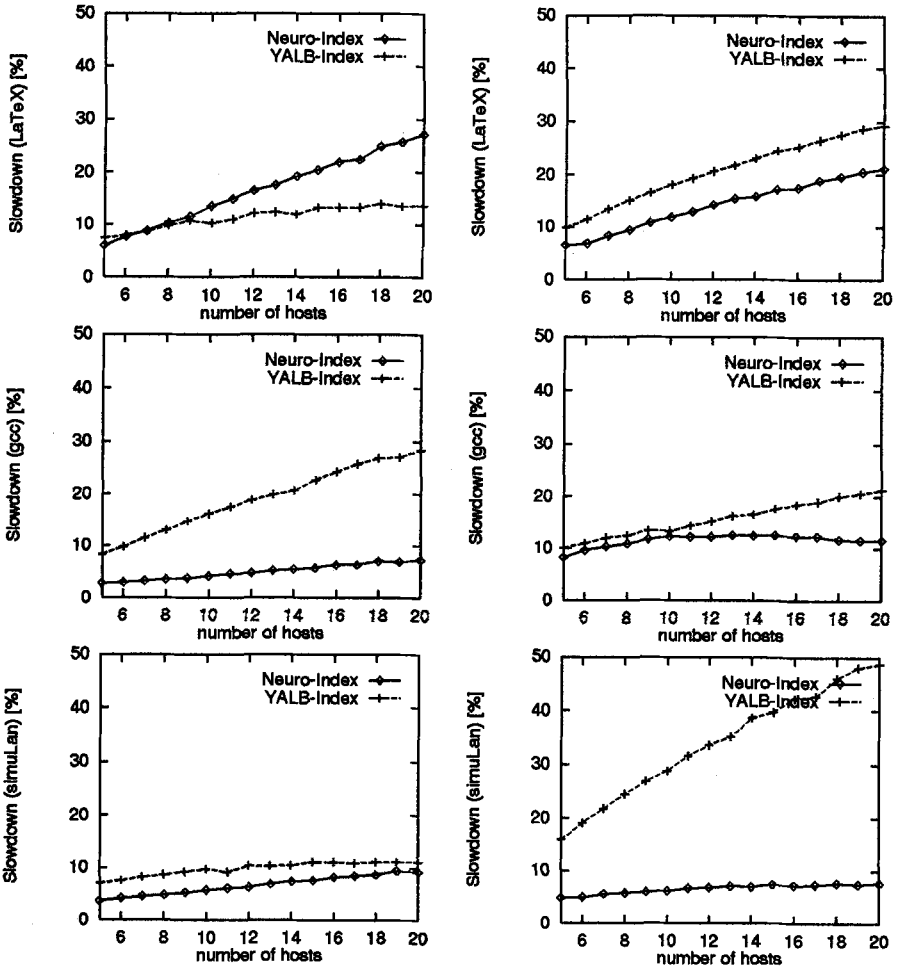

Fig. 1. Slowdown for example applications (left), and the same under medium and high load only (right).

delay performs better, i.e., it yields a lower slowdown. For gcc and simuLan the Neuro–delay shows very good results (see figure 1 (left)).

The slowdowns tend to get a little bit worse with increasing system size, because the probability to choose a sample of a lowly loaded machine increases, too, so that the optimal $runtime(M_{opt}, load_{M_{opt}})$ decreases. When the metric favors a wrong sample as the fastest one, this mistake will be weighted stronger when a shorter optimal runtime is possible.

Results under Medium and High Load Again, for this experiment the samples of lowly loaded hosts are removed. In this situation, the YALB-slowdown increases (see figure 1 (right)). The Neuro-slowdown is lower in most test cases and stays stable also for bigger systems.

Results for Unknown Applications Our first experiences have shown a poor performance of the Neuro–Delay with unknown applications. Therefore, we trained even simpler networks with modified input parameter sets to also handle I/O intensive applications and applications with a higher percentage of floating point operations. The improved results of these experiments will be presented elsewhere [7].

3 Conclusions

We compared two approaches for load metrics in heterogeneous systems: a simple load metric which uses only the current run queue length and the speed characteristics of the machines, and a metric which is calculated by neural networks.

While the first is easy to calculate and performs satisfactory in heterogeneous systems, the neural network approach yields better results, especially under medium and high load and when the system size increases.

We want to thank Detlef Nauck of our Softcomputing department. Without him and his support in the field of neural networks this work would not have been done.

Further information about our research activities can be found on our web page `http://www.ibr.cs.tu-bs.de/projects/load/` .

References

1. D. Ferrari and S. Zhou. An empirical investigation of load indices for load balancing applications. In *Performance '87*, pages 515–528. Elsevier Science Publishers, 1988.
2. S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillian College Publishing Company, New York, 1994.
3. T. Kunz. The Influence of Different Workload Descriptions on a Heuristic Load Balancing System. *IEEE Transactions on Software Engineering*, 17(7):725–730, July 1991.
4. Pankaj Mehra. *Automated Learning of Load-Balancing Strategies for a Distributed Computer System.* PhD thesis, University of Illinois at Urbana-Champaign, 1993. Available on ftp.ibr.cs.tu-bs.de.
5. I. Miller and J.E. Freund. *Probability and Statistics for Engineers.* Prentice-Hall, Englewood Cliffs, 2nd edition, 1977.
6. B. Schnor, H. Langendörfer, and S. Petri. Einsatz neuronaler Netze zur Lastbalancierung in Workstationclustern. In *Proc. Praxisorientierte Parallelverarbeitung*, pages 154–165, Braunschweig, October 1994.
7. B. Schnor, S. Petri, and H. Langendörfer. Using neural networks for prediction of load indices in heterogeneous computing environments. In Preparation, 1996.