Adaptive Load Balancing of Parallel Applications with Reinforcement Learning on Heterogeneous Networks

Johan PARENT
COMO, VUB
Brussels, Belgium
Email: johan@info.vub.ac.be

Katja Verbeeck
COMO, VUB
Brussels, Belgium
Email: kaverbee@vub.ac.be

And

Jan LEMEIRE
PADX, VUB
Brussels, Belgium
Email: jlemeire@info.vub.ac.be

Published in Proc. of Int. Symposium DCABES 2002, Wuxi, China, Dec 16th – 20th, 2002

ABSTRACT
We report on the improvements that can be achieved by applying machine learning techniques, in particular reinforcement learning, for the dynamic load balancing of parallel applications. The applications being considered here are coarse grain data intensive applications. Such applications put high pressure on the interconnect of the hardware. Synchronization and load balancing in complex, heterogeneous networks need fast, flexible, adaptive load balancing algorithms. Using reinforcement learning it is possible to improve upon the classic job farming approach.

Keywords: Parallel processing, Adaptive load balancing, reinforcement learning, heterogeneous network, intelligent agents, data intensive applications.

1. INTRODUCTION

Load balancing is crucial for parallel applications since it ensures a good use of the capacity of the parallel processing units. Here we look at applications which puts high demands on the parallel interconnect in terms of throughput. Examples of such applications are compression applications which both process important amounts of data and require a lot of computations. Data intensive applications [2] require a lot of communication and are therefore dreaded for most parallel architectures. The problem is exacerbated when working with heterogeneous parallel hardware. This is the case in our experiment using a heterogeneous cluster of PCs to execute parallel application with a master-slave software architecture. Adaptive load balancing is indispensable if system performance is unpredictable and no prior knowledge is available [1].

In the multi-agent community, adaptive load balancing is an interesting testbed for multi-agent learning algorithms, likewise for multi-agent reinforcement algorithms as in [8, 10]. However the interpretations and models of load balancing there are not always in the view of real parrallel applications. We report on the results of adaptive agents in the farming scheme.
The bottleneck for parallelizing data intensive applications is the link to the master. The presented results show that using reinforcement learning it is possible to reduce the strain on the communication hardware. This can be achieved by individually adapting the amount of data (block size) requested by each slave. The learning scheme proves to be better than the relatively efficient sequential job farming scheme.
This document is structured as follows. Section 2 introduces reinforcement learning. Section 3 gives an overview of the existing load balancing strategies. Section 4 presents the experimental setup and section 5 reports the experimental results. The last section concludes with results and future work.

2. REINFORCEMENT LEARNING

Reinforcement learning is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment. A model of reinforcement learning consists of a discrete set of environment states, a discrete set of agent actions and a set of scalar reinforcement signals. On each step of interaction the agent receives reinforcement and some indication of the current state of the environment, and chooses an action.
The agent's job is to find a policy, i.e. a mapping from states to actions, which maximizes some long-run measure of reinforcement. These rewards

can take place arbitrarily distant in the future. To obtain a high overall reward, an agent has to prefer actions that it has learned in the past and found to be good, i.e. exploitation, however discovering such actions is only possible by trying out alternative actions, i.e. exploration. Neither exploitation, nor exploration can be pursued exclusively.

Common reinforcement learning methods, which can be found in [6, 12] are structured around estimating value functions. A value of a state or state-action[1] pair, is the total amount of reward an agent can expect to accumulate over the future, starting from that state. One way to find the optimal policy is to find the optimal value function. If a perfect model of the environment as a Markov decision process is known, the optimal value function can be learned with an algorithm called value iteration. An adaptive version of this algorithm exists for situations were a model of the environment is not known in advance.

For instance the Q-learning algorithm, which is an adaptive value iteration method [6, 12] bootstraps its estimate for the state-action value $Q_{t+1}(s,a)$ at time *t+1* upon its estimate for $Q_t(s^{'},a^{'})$ with *s'* the state where the learner arrives after taking action *a* in state *s*:

$$Q_{t+1}(s,a) = (1-\alpha).Q_t(s,a) + \alpha.(r + \gamma.\max_{a^{'}} Q_t(s^{'},a^{'})) \quad (1)$$

With α the usual step size parameter, γ a discount factor and *r* the immediate reinforcement.

In our load-balancing problem setting processors are of the receiver-initiated type and can thus be viewed as agents, which we will give extra learning abilities. Each processor will be an independent Q-learning agent, which tries to learn an optimal chunk size of data to ask the master, so that the blocking time for others is minimized.

3. LOAD BALANCING

Load balancing is assigning to each processor work proportional to its performance, minimizing the execution time of the program. But processor heterogeneity and performance fluctuations make static load balancing insufficient [1]. We investigate dynamic, local, distributed load balancing strategies [13], which are based on heuristics, since finding the optimal solution has shown to be NP-complete in general [9]. Following the agent philosophy, the request assignment strategy is a receiver-initiated algorithm [5], in which the "consumers" of workload look for producers [11]. The goal is a fast adaptive system that optimizes computation and synchronization.

4. EXPERIMENTS

Problem description

In situations were the communication time is not negligible, as is the case for data intensive applications, faster processing units can incur serious penalties due to slower units. A data request issued by a slow unit can stall a faster unit when using farming. This of course results in a reduction of the parallelism.

This phenomenon is bound to occur when slave request identical amounts of data from the master. And this independently of the actual amount (we here neglect the communication delay, which is acceptable given sufficiently big requests).

In order to improve upon the job-farming scheme when working with heterogeneous hardware, the slaves have to request different amount of data from the master (server). Indeed their respective consumption of communication bandwidth should be proportional to their processing power. Slower processing units should avoid obstructing faster ones by requesting less data from the master.

Computation model

The initial computation model is sequential job farming. In this master-slave architecture the slaves (one per processing unit) request a chunk of a certain size from the master. As soon as the data has been processed the result is transferred to the master and the slave sends a new request (figure 1).

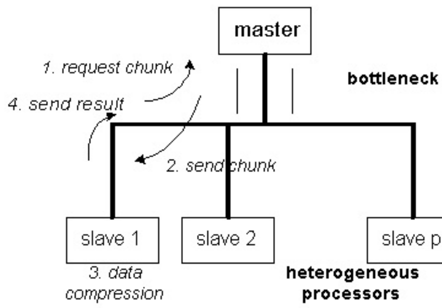

Fig 1: Model.

This scheme has the advantage of being both simple and efficient. Indeed, in the case of heterogeneous hardware the load (amount of processed data) will depend on the processing speed of the different processing units. Faster processing units will more

[1] In control problems approximating action/ value functions is more interesting because then there is no need for knowing the environments transition dynamics.

frequently request data and thus be able to process more data.
The bottleneck of data intensive applications with a master-slave architecture is the link connecting the slaves to the master. In the presented experiments all the slaves share a single link to their master (through an ethernet switch). In this scenario the applications performance will be influenced by the efficient use of the shared link to the master. Indeed, the fact that the application has a coarse granularity only insures that the computation communication ratio is positive. But it does not preclude a low parallel efficiency even when using job farming.

Experimental setup
To assess the presented algorithm for coarse grain data intensive applications on heterogeneous parallel hardware a synthetics approach has been used. An application has been written using the PVM [3] message-passing library to experiment with the different dimensions of the problem. The application has been designed not to perform any real computation, but instead it replaces the computation and communication phases by delays with equivalent duration[2].

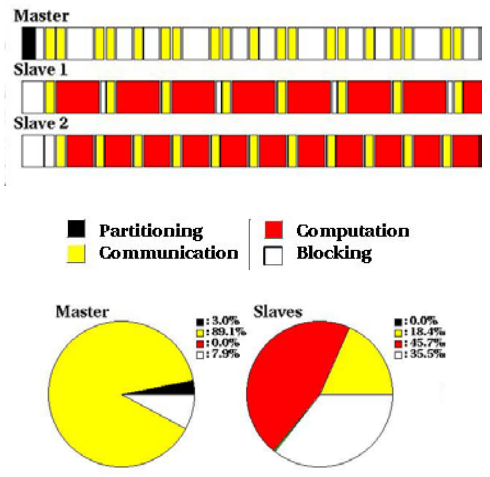

Fig 2: Computation bottleneck

The advantage is the possibility of configuring all the parameters during an experiment, both the granularity of the hardware and the software can be chosen. Choosing the ratio between bandwidth of the network (MB/sec) and the execution speed of the processors (MB/sec) can set the hardware granularity. The software granularity is ratio of the required communication (MB) and processing (MB) and can thus be chosen as well. This gives more control when running the experiments without losing the essential behavior of such an application.

We will investigate the non-trivial case where the total task computation time is comparable with the total communication time through the bottleneck. This is achieved by setting the average granularity equal to the number of processors.
When the total computation power is lower than the master's communication bandwidth, there is no bottleneck, the master will be able to serve the slaves constantly and these will work at 100% efficiency. This can be seen in the experiment of figure 2, where the total computation power is 0.45 of the total communication bandwidth. But for data intensive application, this won't be the case; moreover, more processors can be added. On the other hand, when the total computation power is higher, the communication at the master will serve as the bottleneck, reaching 100% efficiency. But the efficiency of the slaves will drop, so the surplus of slave processors should better be used for other computations (as in the Grid philosophy). This can be seen in figure 3, where the total computation power is 2.45 of the total communication bandwidth.

Learning to request data
The algorithm is based upon the concept of a Stochastic Learning Automaton, such an automaton serves the purpose of finding optimal actions out of a set of allowable actions [7]. Unlike a reinforcemtent learner model, a stochastic learning automaton only considers one state and uses a $\gamma = 0$.

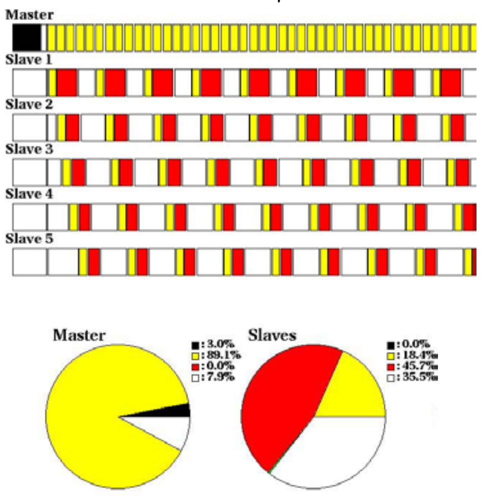

Fig 3: Communication bottleneck

In our experiment, the slaves will learn to request a chunk size that minimizes its blocking time. To that end each slave has a stochastic learning automaton which uses Eq.(1), as shown in Fig 4. In the presented results the block size is a multiple of a given initial block size. Here the multiples are 1, 2 and 3 times the initial block size
The feedback *r* provided to the learner is the waiting time, which is the time a slave has to wait before the master acknowledges its request for data. The inverse of the waiting time (1/ waiting time) is used to update the Q-values using Eq(1). Less interesting

[2] The experimental application can easily be turned into a real application by replacing the delay producing code with real code.

actions (i.e. multiples of initial block size) will incure higher waiting times and thus will have lower Q-values associated with them.

The stochastic learning automaton choses an action probabilistically using the Q-values. Which means the actions with a low Q-value have a lower chance of being chosen. In order to get reliable Q-values the stochastic learning automaton present in each of the slaves does not choose a new actions for each request. Instead, each action is performed T times (T=5 during the experiments).

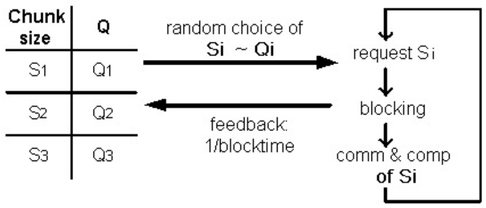

Fig 4: The Reinforcement Learner.

We will compare our adaptive algorithm with a static load-balancing scheme where fixed amounts of data are requested.

5. EXPERIMENTS

Figure 5 shows the time course of a typical experiment, with the computation, communication and blocking phases (data size = 800MB, communication speed = 40MB/s, average chunk size = 1MB, processors=4, average granularity = #processors).

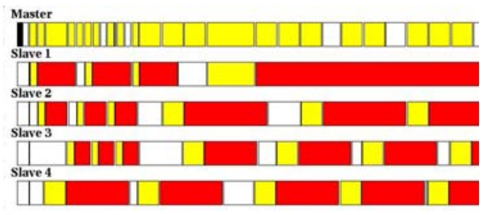

Fig 5: experiment.

Performance results

The global goal is minimizing the total computation time, as shown in table 1 (same parameter values as in the previous experiments, we vary the number of processors and keep the average granularity = #processors). In these first experiments, we get an improvement of about 40% of the learner upon the static load balancer.

#Slaves	Static LB	Adaptive LB	Perf. Gain
4	32.1s	22.6s	-42%
8	32.0s	23.0s	-39%
10	32.2s	23.1s	-39%

Table 1: Total Computation Time.

Overhead Analysis

Three overheads are responsible for the total computation time (table 2, same experiment as for Fig 5):

- The masters blocking time, which stands for inefficient use of the communication bottleneck at the master.
- The total blocking time of the slaves, due to inefficient synchronization (since the total computation time equals the communication time).
- The total computation time of the slaves, which decreases with better load distribution (better use of the faster processors of the heterogeneous network).

	Static LB	Adaptive LB	Perf Gain
Total Computation time	32.1s	22.6s	-42%
Master blocking time	8.2s	5.3s	-35%
Total slave blocking time	80.1s	49.5s	-62%
Total slave computation time	63.1s	54.0s	-16%

Table 2: Overhead Analysis.

The results show that the 3 overheads have decreased, due to better load balancing.
In the next section we will investigate which parameters influence the performance.

Performance Analysis

As can be seen in table 1, the number of slaves doesn't influence the performance gain. On the other hand, the learning algorithm needs a training period. We investigated this by varying the number of requests of the experiments. The results can be seen in figure 6, where we plot the performance gain in function of the number of requests. This parameter was set by varying the blocksize and the number of processors.

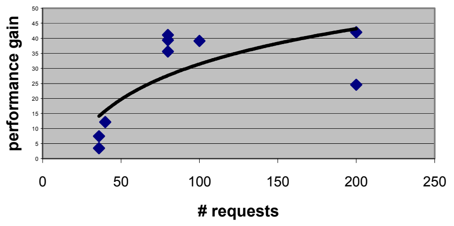

Fig 6: Training Period.

6. CONCLUSIONS

Complex, heterogeneous system controlled for optimal use by learning automata is promising. We implemented a reinforcement learner for distributed load balancing of data intensive applications. The first performance results show considerable improvements upon a static load balancer. The

algorithm works locally on the slaves (receiver-initiated), thus acting like intelligent agents.
Until now the learning automaton is only fed by local information, which we believe, is too less for fast, highly efficient parallel processing. We will have to extend the feedback by exchanging information between the agents, as explained in the next section.

7. FURTHER WORK

Feeding the reinforcement learner with only local information leads to egoistic behavior or non-cooperative load balancing [4]. Improved performance will be reached by exchanging information between the slaves (or agents), i.e. feedback needed for better synchronization and load distribution.
Another important aspect in heterogeneous networks is the computational contribution of the fast processors, where computation time is of higher value than equal time on slower processors. To take this into account, we first express the processor power relative to a base one [12], which we expressed in our setup by the hardware granularity. Then the processor waiting time, used as feedback for the reinforcement learner, has to be multiplied by this factor. This scales local time with respect to the processor power and will result in a better tuning.

8. REFERENCES

[1] I. Banicescu and V. Velusamy, “Load Balancing Highly Irregular Computations with the Adaptive Factoring”, Proceedings of the 16th International Parallel & Distributed Processing Symposium, IEEE, Los Alamitos, California, 2002.

[2] M. D. Beynon, T. Kurc et al. “Efficient Manipulation of Large Datasets on Heterogeneous Storage Systems”, Proceedings of the 16th International Parallel & Distributed Processing Symposium, IEEE, Los Alamitos, California, 2002.

[3] A. Geist, A. Beguelin et al., “PVM: Parallel Virtual Machine”, the MIT press, 1994.
www.netlib.org/pvm3/book/pvm-book.html
www.epm.ornl.gov/pvm/ (pvm homepage)

[4] D. Grosu and A.T. Chronopoulos, “A Game-Theoretic Model and Algorithm for Load Balancing in Distributed Systems”, Proceedings of the 16th International Parallel & Distributed Processing Symposium, IEEE, Los Alamitos, California, 2002.

[5] D. Gupta and P. Bepari, "Load sharing in distributed systems", In Proceedings of the National Workshop on Distributed Computing, January 1999.

[6] Kaelbling L.P., Litmann M.L., Moore A.W.,: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4 (1996) p 237-285.

[7] T. Kunz, “The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme”, IEEE Transactions on Software Engineering, Vol. 17, No. 7, July 1991, pp. 725-730.

[8] Nowé, A., Verbeeck, K., “Distributed Reinforcement learning, Loadbased Routing a case study”, Proceedings of the Neural, Symbolic and Reinforcement Methods for sequence Learning Workshop at ijcai99, 1999.

[9] C.C. Price and S. Krishnaprasad, “Software allocation models for distributed systems”, in Proceedings of the 5th International Conference on Distributed Computing, pages 40-47, 1984.

[10] Schaerf A., Shoham Y., Tennenholtz M., “Adaptive Load Balancing: A Study in Multi-Agent Learning”, Journal of Artificial Intelligence Research (1995) 475-500.

[11] T. Schnekenburger and G. Rackl, “Implementing Dynamic Load Distribution Strategies with Orbix”, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, Nevada, 1997.

[12] Sutton, R.S., Barto, A.G., “Reinforcement Learning: An introduction”, Cambridge, MA: MIT Press (1998).

[13] M.J. Zaki, Wei Li; S. Parthasarathy, “Customized dynamic load balancing for a network of workstations”, Proceedings of the High Performance Distributed Computing (HPDC'96), IEEE, 1996.