Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 7 (11): 1504-1510, 2007
ISSN 1812-5654

Q-Learning Scheduler and Load Balancer for Heterogeneous Systems

Faiza Samreen and M. Sikandar Hayat Khiyal
Department of Computer Science, International Islamic University, Islamabad, Pakistan

Abstract: Distributed computing is a viable and cost-effective alternative to the traditional model of computing. Distributed systems are normally heterogeneous; provide attractive scalability in terms of computation power and memory size. Generally, in such systems no processor should remain idle while others are overloaded. Large degrees of heterogeneity add additional complexity to the scheduling problem. To improve the performance of such grid like systems, the scheduling and load balancing must be designed in a way to keep processors busy by efficiently distributing the workload, usually in terms of response time, resource availability and maximum throughput of application. Dynamic load balancing is NP complete. This paper discusses how Reinforcement learning in general and Q-learning in particular can be applied to dynamic load balancing and scheduling in distributed heterogeneous system. We consider a grid like environment consisting of multi-nodes. Experimental results suggest that Q-learning improves the quality of load balancing in large scale heterogeneous systems.

Key words: Distributed computing, load balancing, reinforcement learning, Q-learning

INTRODUCTION

Distributed heterogeneous systems emerged as a viable alternative to dedicated parallel computing (Keane, 2004). A distributed system is made up of a set of sites cooperating with each other for resource sharing. The information exchange medium among the sites is a communication network. When the processing power varies from one site to another, a distributed system seems to be heterogeneous in nature (Karatza and Hilzer, 2002). Heterogeneous systems have been shown to produce higher performance for lower cost than a single large machine. The factors of performance degradation during parallel execution are: the frequent communication among processes; the overhead incurred during communication; the synchronizations during computations; the infeasible scheduling decisions and the load imbalance among processors (Dhandayuthapani *et al*., 2005). In short we can say that, Load balancing and Scheduling are crucial factors for grid like distributed heterogeneous systems (Radulescu and van Gemund, 2000).

Scheduling is all about keeping processors busy by efficiently distributing the workload. Load balancing attempts to ensure that the workload on each host is within a balance criterion of the workload present on every other host in the system. Dynamic load balancing assumes no prior knowledge of the tasks at compile-time. Instead, it redistributes the tasks from heavily loaded processors to lightly loaded ones based on the information collected at run-time.

Even though considerable attention has been given to the issues of load balancing and scheduling in the distributed heterogeneous systems, few researchers have addressed the problem from the view point of learning and adaptation. It has been shown by the communities of Multi-Agents Systems (MAS) and distributed Artificial Intelligence (AI) that groups of autonomous learning agents can successfully solve the issues regarding different load balancing and resource allocation problems (Weiss and Schen, 1996; Stone and Veloso, 1997; Weiss, 1998; Kaya and Arslan, 2001). Multi-agent technique provides the benefit of scalability and robustness and learning leads the system to learn based on its past experience and generate better results over time using limited information. To solve these core issues like learning, planning and decision making Reinforcement Learning (RL) is the best approach and active area of AI.

Reinforcement learning: Reinforcement Learning (RL) is an active area of research in AI because of its widespread applicability in both accessible and inaccessible environments. The model of the reinforcement learning problem is based on the theory of Markov Decision Processes (MDP) (Stone and Veloso, 1997).

In RL, an agent learns by interacting with its environment and tries to maximize its long term return by performing actions and receiving rewards as shown in Fig. 1. This area of machine learning learns the behavior of dynamic environment through trial and error. The trial and error learning feature and the concept of reward makes the reinforcement learning distinct from other learning techniques.

Corresponding Author: Faiza Samreen, Department of Computer Science, International Islamic University, Islamabad, Pakistan

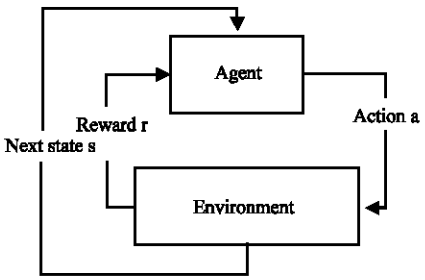

Fig. 1: Reinforcement learning model

Q-learning: The Q-learning is a recent form of Reinforcement Learning. It works by maintaining an estimate of the Q-function and adjusting Q-values based on actions taken and reward received (Kaelbling *et al*., 1996) (Sutton and Barto, 1998). It is adaptive version of Reinforcement Learning and does not need model of its environment. Q-learning gradually reinforces those actions that contribute to positive rewards by increasing the associated Q-values. Q-value can be calculated by Eq. 1.

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\left(r + \gamma \max Q_t(s',a') - Q_t(s,a)\right) \quad (1)$$

$Q_{t+1}(s,a)$ denotes the state-action value of the next possible state at time t+1, r the immediate reinforcement and α is the learning rate of the agent. One expects to start with a high learning rate, which allows fast changes and lowers the learning rate as time progresses. Action a must be chosen which maximizes, Q(s,a). When in each state the best-rewarded action is chosen according to the stored Q-values, this is known as greedy-method. γ is discount factor. The closer γ is to 1 the greater the weight is given to future reinforcements.

Motivation behind using this technique is that, Q-Learning does converge to the optimal Q-function (Even-Dar and Monsour, 2003). It uses the observed information to approximate the optimal function, from which one can construct the optimal policy.

Related work: Extensive research has been done in developing scheduling algorithms for load balancing of parallel and distributed systems. These algorithms are broadly classified as non-adaptive and adaptive algorithms.

Guided Self Scheduling (GSS) (Polychronopoulos and Kuck, 1987) and factoring (FAC) (Hummel *et al*., 1993) are examples of non-adaptive scheduling algorithms. GSS addresses the problem of uneven starting time of the processor and is applicable to constant length and variable length iterates executions (Polychronopoulos and Kuck, 1987). In FAC, iterates are scheduled in batches, where the size of a batch is a fixed ratio of the unscheduled iterates and the batch is divided into P chunks (Hummel *et al*., 1993).

Banicescu *et al*. (2000) proposed Adaptive Weighted Factoring (AWF) algorithm which was applicable to time stepping applications, it uses equal processor weights in the initial computation and adapts the weight after every time step. Adaptive Factoring (AF) (Banicescu and Liu, 2000) dynamically estimated the mean and standard deviation of the iterate execution times during runtime.

The scheduling problem is known to be NP-complete. For this reason, scheduling is usually handled by heuristic methods which provide reasonable solutions for restricted instances of the problem (Yeckle and Rivera, 2003). Most research on scheduling has dealt with the problem when the tasks, inter-processor communication costs and precedence relations are fully known. Now we will converge specifically towards multi-agent RL techniques. Majercik and Littman (1997) evaluated, how the load balancing problem can be formulated as a Markov Decision Process (MDP) and described some preliminary attempts to solve this MDP using guided on-line Q-learning and a linear value function approximator tested over small range of value runs. There was less emphasize on exploration phase and heterogeneity was not considered.

Zomaya *et al*. (1998) proposed five Reinforcement Based Schedulers (RBSs) which were: 1) Random RBS 2) Queue Balancing RBS 3) Queue Minimizing RBS 4) Load Based RBS 5) Throughput based RBS. The load-based and throughput-based RBSs were not effective in performing dynamic scheduling. The random scheduler and the queue-balancing RBS proved to be capable of providing good results in all situations. The queue balancing RBS had the advantage of being able to schedule for a longer period before any queue overflow took place. Random scheduler was Capable of extremely efficient dynamic scheduling when the processors are relatively fast. Under more difficult conditions, its performance is significantly and disproportionately reduced.

Parent *et al*. (2002) implemented a reinforcement learner for distributed load balancing of data intensive applications in heterogeneous environment. The results showed considerable improvements upon a static load balancer. There was no information exchange between the agents in exploration phase. Later Parent *et al*. (2004) improved the application as a framework of multi-agent reinforcement learning for solving communication overhead. This algorithm was receiver initiated and works locally on the slaves.

Galstyan *et al.* (2004) proposed, Minimalist decentralized algorithm for resource allocation in a simplified Grid-like environment. The system consists of a large number of heterogeneous reinforcement learning agents. This technique neglected the need for co-allocation of different resources. Present work is the enhancement of this technique. Verbeeck *et al.* (2005) described how multi-agent reinforcement learning algorithms can practically be applied to common interest problem and conflicting interest problem. They proposed a new algorithm called Exploring Selfish Reinforcement Learning (ESRL) based on 2 phases, exploration and synchronization phase.

Problem description: The aim of this research is to solve scheduling and load balancing problem and extension of Galstyan *et al.* (2004) work by handling co-allocation. There are some other challenges and Issues which are considered by this research.

- Some existing scheduling middle-wares are not efficient as they assume knowledge of all the jobs in a heterogeneous environment.
- Process redistribution cost and reassignment time is high in case of non-adaptive algorithms.
- Complex nature of the application causes unrealistic assumptions about node heterogeneity and workload.
- Allocating a large number of independent tasks to a heterogeneous computing platform is still a hindrance. This is due to the different speeds of computation and communication of resources.
- A further challenge to load balancing lies in the lack of accurate resource status information at the global scale. Redistribution of tasks from heavily loaded processors to lightly loaded ones in dynamic load balancing needs quick information collection at run-time in order to use it for rectification of scheduling technique.

The goal of this study is to apply Multi-Agent Reinforcement Learning technique to the problem of scheduling and Load Balancing in the grid like environment and dynamically distribute the workload over all available resources in order to get maximum throughput.

MATERIALS AND METHODS

The key features of our proposed solution are: Support for a wide range of parallel applications; use of advance Q-Learning techniques on architectural design and development; multiple reward calculation; and QL-analysis, learning and prediction*. The architecture diagram of our proposed system is shown in Fig. 2.

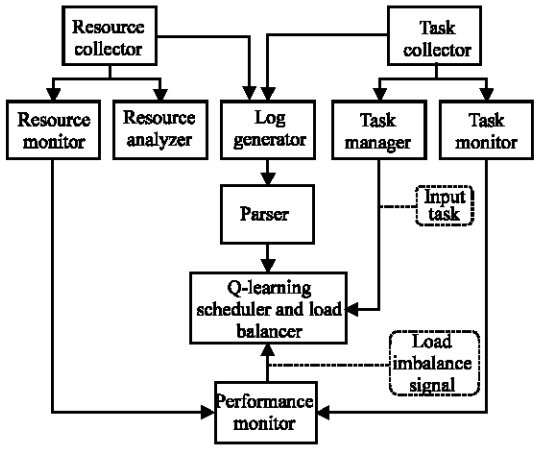

Fig. 2: Architecture diagram of proposed system

Modules description: The Resource Collector directly communicates to the Linux kernel in order to gather the resource information in the grid. The Resource Analyzer displays the load statistics. Tasks that are submitted from outside the boundary will be buffered by the Task Collector. The Task Manager handles user requests for task execution and communication with the grid.

The Log Generator saves the collected information of each grid node and executed tasks information. The Performance Monitor monitors the resource and task information and signals for load imbalance and task completion to the Q-Learning Load Balancer in the form of RL (Reinforcement learning) Signal (described after sub-module description). It is also responsible for backup in case of system failure.

Before scheduling the tasks, the QL Scheduler and Load balancer dynamically gets a list of available resources from the global directory entity. In Q-Learning, the states and the possible actions in a given state are discrete and finite in number. A detailed view of QL Scheduler and Load balancer is shown in Fig. 3.

Sub-module description of QL scheduler and load balancer:

- QL Analyzer receives the list of executable tasks from Task Manager and list of available resources from Resource Collector. It analyzes the submission time and size of input task and forwards this information to State Action Pair Selector.
- The State Action Pair Selector searches the nearest matched states of current input and gets its action set A_t for each agent from Log History. This information provides efficiency and performance of that resource in the past. (Nearest matched states will be searched on the basis of submission time and size of task)

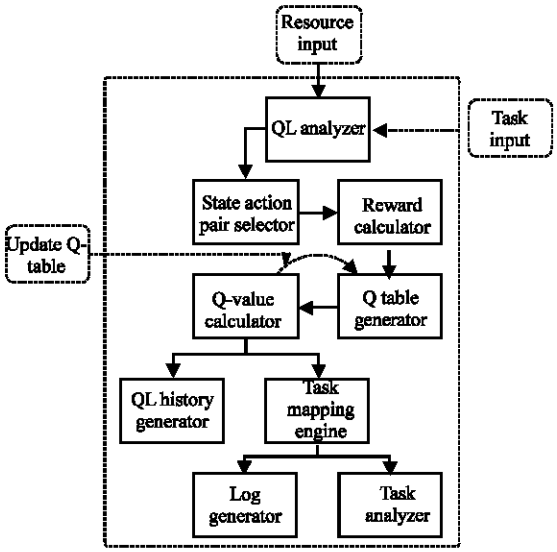

Fig. 3: Detailed View of Q-Learning Scheduler and Load Balancer

- Reward Calculator calculates reward by considering five vectors as reward parameters using Eq. 2. These parameters are: CPU Speed, Load, Memory, Status and Task Execution Time. The reward will be calculated for each resource on the basis of information collected by State Action Pair Selector.

$$f(R) = a(\text{CPU speed}) + b(\text{load}) + c(\text{memory}) + d(\text{status}) + [e(Tw) + (1-e)Tx] \quad (2)$$

$$\text{CPU speed} = \text{max. CPU speed/current speed} \quad (3)$$

$$\text{Load} = [\text{current CPU utility/total possible utility}]^* 100 \quad (4)$$

$$\text{Status} = \text{Average uptime of node} \quad (5)$$

$$\text{Memory} = \text{Average used memory} \quad (6)$$

Where Tw is the task wait time and Tx is the task execution time. a, b, c, d, e are constants determining the weight of each contribution from history performance.

- Q-Table Generator generates Q-Table and Reward-Table and places reward information in Reward-Table.
- The Q-Value Calculator follows the Q-Learning algorithm to calculate Q-value for each node and update these Q-Values in Q-Table. γ value is zero and epsilon greedy policy is used in our proposed approach. Algorithm is given below:
  - Initialize Q (s,a) arbitrarily
  - Repeat (for n episodes) Where $0.3<\alpha>0.1$
  - Initialize s
  - Repeat for each step of episode (Learning)
  - Take action a, observe reward r, move next state s'
  - $Q_{i,t+1} \leftarrow Q_{i,t} + \alpha\,(r-Q_{i,t})$ I = 1-n Resources
  - s = s'
  - Until Learning is done

- QL History Generator stores state action pairs (s_t, a_t) along with Q-values (calculated by Q-Value Calculator) as episodic memory into QL History.
- Task Mapping Engine, Co-allocation is done by the Task Mapping Engine; it calculates the average distribution of tasks and distributes them on selected resources. Average distribution of tasks for Resource $R_{1...n}$ (Avg. Dist. Ri) will be calculated by dividing Q-Value of that particular Resource with Cumulative Q-Value (CQV) using Eq. 7.

$$\text{Avg. Dist. Ri} = \text{Q-Value of Ri/CQV} \quad (7)$$

CQV is calculated by Eq. 8.

$$CQV = \sum_{i=1}^{n} QRi \quad (8)$$

Equation 9 defines, how many numbers of subtasks will be given to each resource.

$$\text{No. of subtasks for Ri} = \text{AvgRi} * \beta \quad (9)$$

β is a constant for determining number of sub jobs calculated by averaging over all submitted sub jobs from history.

- Task Analyzer shows the distribution and run time performance of tasks on grid resources. Out put will be displayed after successful execution.
- Finally, the Log Generator generates log of successfully executed tasks.

Reinforcement learning signals:

Task completion signal: After successful execution of task, Performance Monitor signals the Reward Calculator (sub-module of QL Scheduler and Load balancer) in the form of task completion time. After receiving RL signal Reward Calculator calculates reward and update Q-value in Q-Table.

Load imbalance signal: Performance Monitor keeps track of maximum load on each resource in the form of Threshold value. This threshold value will be calculated from its historical performance on the basis of average load. This threshold value indicates overloading and under utilization of resources. On finding load imbalance, Performance Monitor signals QL Load Balancer to start its working and remapping the subtasks on under utilized resources.

RESULTS AND DISCUSSION

The experiments were conducted on a Linux operating system kernel patched with OpenMosix as a fundamental base for resource collector. For comparison purpose we are using Guided Self Scheduling (GSS) and Factoring (FAC) as non-adaptive algorithms and Adaptive Factoring (AF) and Adaptive Weighted Factoring (AWF) as adaptive algorithms.

The experiments to verify and validate the proposed algorithm are divided into two categories. The first category of e experiments is based on learning with varying effect of load and resources. The second level of experiments describes the load and resource effect on Q-Scheduling and Other Scheduling (Adaptive and Non-Adaptive). Experiments were conducted for a different number of processors, episodes and task input sizes. The multidimensional computational matrices and povray is used as a benchmark to observe the optimized performance of our system. The cost is used as a performance metric to assess the performance of our Q-Learning based grid application. Cost is calculated by multiplying number of processors P with parallel execution time Tp.

Starting with the first category, Table 1-2 and Fig. 4 show the execution time comparison of different number of episodes and processors. We can see from tables that execution time is decreasing when the number of episodes increasing. This allows the system to learn better from more experiences. The results obtained from these comparisons highlight the achievement of the goal of this research work, that of attaining performance improvements by increasing Learning.

For second category of experiments Fig. 5-7 show the cost comparison for 500, 5000 and 10000 episodes respectively. It can be seen from these graphs that the proposed approach performs better than the non-adaptive techniques such as GSS and FAC and even against the advanced adaptive techniques such as AF and AWF. Consistent cost improvement can be observed for increasing number of processors. For Q-learning, there is

Table 1: Execution time for 10000 episodes vs. 6000 episodes with 30 input task and increasing number of processors

	Execution time (sec)	
No. of processors	10000 Episodes	6000 Episodes
4	162.55	215.16
8	55.67	79.99
12	46.183	64.58
16	22.811	39.12

Table 2: Execution time for 5000 episodes vs. 200 episodes with 60 input task and increasing number of processors

	Execution time (sec)	
No. of processors	5000 Episodes	200 Episodes
8	121.64	136.21
12	43.88	53.59
16	37.74	46.34
24	22.41	31.53
32	13.71	20.69

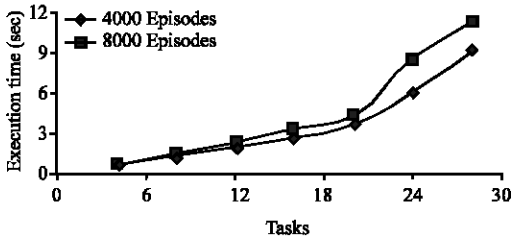

Fig. 4: Execution time for 8000 episodes vs. 4000 episodes with 30 input task and increasing number of processors

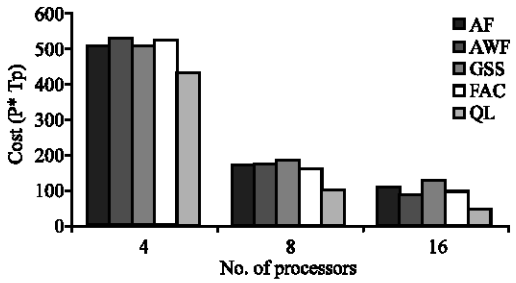

Fig. 5: Cost comparison of QL Scheduling vs. Other Scheduling with increasing number of processors for 500 Episodes

a significant drop in the cost when processors are increased from 2-8. However, Tp does not significantly change as processors are further increased from 12-32. This validates the hypothesis that the proposed approach provides better optimal scheduling solutions when compared with other adaptive and non-adaptive algorithms.

Present proposed technique also handles load distribution overhead which is the major cause of performance degradation in traditional dynamic schedulers.

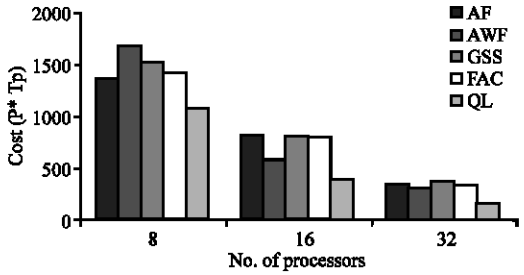

Fig. 6: Cost comparison of QL Scheduling vs. Other Scheduling with increasing number of processors for 5000 Episodes

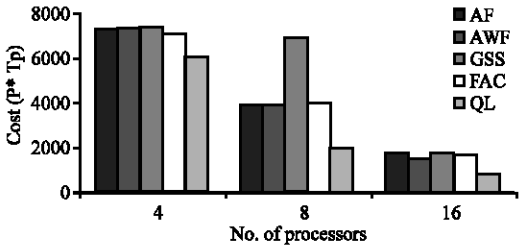

Fig. 7: Cost comparison of QL Scheduling vs. Other Scheduling with increasing number of processors for 10000 Episodes

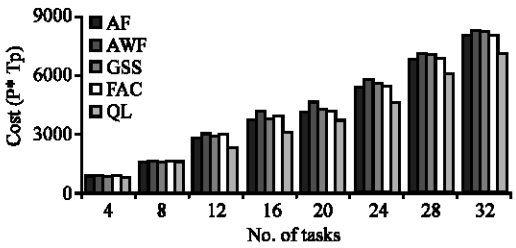

Fig. 8: Cost comparison of Q Scheduling vs. Other Scheduling with increasing number of tasks for 500 Episodes and 8 processors

Figure 8 shows the cost comparison with increasing number of tasks for 8 processors and 500 episodes. Again this graph shows the better performance of QL scheduler with other scheduling techniques. Results of Fig. 8 highlight the achievement of attaining maximum throughput using Q-Learning while increasing number of tasks. By outperforming the Other Scheduling, the QL-Scheduling achieves the design goal of dynamic scheduling, cost minimization and efficient utilization of resources.

CONCLUSION AND FUTURE WORK

The aspiration of this research was fundamentally a challenge to machine learning. Computer systems can optimize their own performance by learning from experience without human assistance. To repeatedly adjust in response to a dynamic environment, they will need the adaptability that only machine learning can offer. In this regard, the use of Reinforcement Learning is more precise and potentially computationally cheaper than other approaches. This research has shown the performance of QL Scheduler and Load Balancer on distributed heterogeneous systems. Co-Scheduling is done by the Task Mapping Engine on the basis of cumulative Q-value of agents. Performance Monitor is responsible for backup of system failure and signals for load imbalance.

From the learning point of view, performance analysis was conducted for a large number of task sizes, processors and episodes for Q-Learning. The optimality and scalability of QL-Scheduling was analyzed by testing it against adaptive and non-adaptive Scheduling for a varying number of tasks and processors. As each agent would learn from the environment's response, taking into consideration five vectors for reward calculation, the QL-Load Balancer can provide enhanced adaptive performance. Ultimately, the outcome indicates an appreciable and substantial improvement in performance on an application built using this approach.

In future we will enhance this technique using SARSA algorithm, another recent form of Reinforcement Learning. We will try to merge our methodology with Verbeeck *et al*. (2005) proposed algorithm.

REFERENCES

Banicescu, I. and Z. Liu, 2000. Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes. In Proceedings of High Performance Computing Symposium, pp: 122-129.

Banicescu, I., V. Velusamy and J. Devaprasad, 2000. On the scalability of adaptive weighted factoring. J. Networks, Software Tools Applic., 6: 213-226.

Dhandayuthapani, S., L. Banicescu, R.L. Carino, E. Hansen, J.P. Pabico and M. Rashid, 2005. Automatic selection of loop scheduling algorithms using reinforcement learning. In Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (HPDC- Stone, P. and M. Veloso, July 2000.

Even-Dar, E. and Y. Mansour, 2003. Learning rates for q-learning. J. Machine Learning Res., 5: 1-25.

Galstyan, A., K. Czajkowski and K. Lerman, 2004. Resource Allocation in the Grid Using Reinforcement Learning. In: Proceedings of the 3rd International Conference on Autonomous Agents and Multi-Agent Systems 2005, July 19-23, 2004, AAMAS'04, New York, USA.

Hummel, S.F., E. Schonberg and L.E. Flynn, 1993. Factoring: A method for scheduling parallel loops. Communications of the ACM., 35: 90-101.

Kaelbling, L.P., M.L. Littman and A.P. Moore, 1996. Reinforcement learning: A survey. J. Artif. Intelli. Res., 6: 237-285.

Kaya, M. and A. Arslan, 2001. Parallel and Distributed Multi-agent Reinforcement Learning. In: Proceedings of Eighth International Conference on Parallel and Distributed Systems (ICPADS'01).

Karatza, H.D and R.C. Hilzer, 2002. Load Sharing in Heterogeneous Distributed Systems. In: Proceedings of the Winter Simulation Conference, ACM, IEEE, SCS, San Diego, California, pp: 489-496.

Keane, T.M., 2004. A general-purpose heterogeneous distributed computing system, M.Sc. Thesis, National University of Ireland, Maynooth, Co. Kildare, Ireland.

Majercik, S.M. and M.L. Littman, 1997. Reinforcement learning for selfish load balancing in a distributed memory environment. In: Proceedings of the International Conference on Information Sciences, pp: 262-265.

Polychronopoulos, C. and D. Kuck, 1987. Guided self-scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. IEEE Trans. Comput., C-36: 1425-1439.

Parent, J., K. Verbeeck and J. Lemeire, 2002. Adaptive Load Balancing of Parallel Applications with Reinforcement Learning on Heterogeneous Networks. In: Proceedings of International Symposium DCABES Dec 16th-20th, 2002, Wuxi, China.

Parent, J., K. Verbeeck, A. Nowe and K. Steenhaut, 2004. Adaptive load balancing of parallel applications with multi-agent Reinforcement Learning on Heterogeneous Networks. Submitted for Scientific Programming Journal Special Issue on Distributed Computing and Computation, IOS Press.

Radulescu, A. and A.J.C. van Gemund, 2000. Fast and effective scheduling in heterogeneous systems. In: Proceedings HCW 2000, Cancun, May 2000, IEEE CS, pp: 229-238.

Stone, P. and M. Veloso, 1997. Multi-agent Systems: A Survey from a Machine Learning Perspective. Autonomous Robots, 8: 345-383. Formerly citable as Carnegie Mellon University CS technical report number CMU-CS-97-193.

Sutton, R.S. and A.G. Barto, 1998. Reinforcement Learning: An Introduction. The MIT Press, Cambridge, Massachusetts. (CLADE 2005), Research Triangle Park, North Carolina, USA. pp: 87-94. IEEE Computer Society Press, 24 July 2005.

Verbeeck, K.A., Now´e, K. Tuyls, 2005. Coordinated Exploration in Multi-Agent Reinforcement Learning: An Application to Load-balancing. In Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems 2005 (AAMAS'05). Utrecht, Netherlands.

Weiss, G. and S. Schen, 1996. Adaptation and Learning in Multi- agent systems: Some remarks and a bibliography. Adaptation and Learning in Multi-Agent Systems. Lecture Notes in Artificial Intelligence, Springer-Verlag, pp: 1042.

Weiss, G., 1998. A multi-agent perspective of parallel and distributed machine intelligence. In: Proceedings of 2nd International Conference on Autonomous Agents, ACM Press, pp: 226-230.

Yeckle, J. and W. Rivera, 2003. Mapping and characterization of applications in heterogeneous distributed systems. Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI2003).

Zomaya, A.Y., M. Clements and S. Olariu, March 1998. A Framework for reinforcement-based scheduling in parallel processor systems. IEEE Transactions On Parallel And Distributed Systems, Vol. 9, No. 3.