http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (7): 1005-1012, 2007
ISSN 1812-5638

A Novel ANN-Based Load Balancing Technique for Heterogeneous Environment

Muniza Salim, Ammara Manzoor and Khalid Rashid
Department of Computer Science, Faculty of Basic and Applied Science,
International Islamic University, Islamabad, Pakistan

Abstract: Grid computing is an emerging computing paradigm and is distinguished from distributed computing by its efficient and optimal utilization of heterogeneous, loosely coupled resources tied to work load management. However, complexity incurred in efficient management of heterogeneous, geographically distributed and dynamically available resources has become one of the most challenging issues in grid computing. A lot of parameters have to be taken into consideration to efficiently utilize the grid resources. Many heuristics has been proposed in the literature to address this complex problem. Present research aims to solve load balancing decisions using Artificial Neural Networks (ANN). Since ANN are best at identifying patterns or trends in data, their ability to learn by examples makes them very flexible and powerful. In this research we have developed and evaluated a completely new scheduling-cum-load balancing module for a scaleable grid. Experimental results suggest that once trained, ANN outperforms other heuristic approaches for large tasks. However for small tasks, ANN suffers from extensive overheads.

Key words: Grid computing, load balancing, Artificial Neural Networks

INTRODUCTION

Advances in networking infrastructure and ever changing computational requirements have led to the development of grid infrastructure that provides a way to meet the needs of these demanding applications. Grid provides predictable, consistent and uniform access to geographically distributed resources like computers, data repositories, scientific instruments and advanced display devices (Foster and Kesselman, 1999).

Grids are often considered as the successors of distributed computing. Grid computing is the more general case of distributed computing which enables collaborative multi-site computation (Foster *et al*., 2001). Grid integrates and coordinates resources and users that live within different control domain as they lack central control (Foster, 2002). Moreover the Computational capabilities of a Grid vary significantly over time as the resources are added and removed dynamically. OGSA (Open Grid Services Architecture) which defines uniform exposed services semantics (the grid service), defines standard mechanism for creating, naming and discovering transient grid service instances, provides transparency for service instances and supports integration with underlying native platform facilities (Foster *et al*., 2002).

The objective of load balancing is to minimize the response time and execution time of a program by trying to equally spreading the load on processors and maximizing their utilization. The goal of grid task scheduling is to achieve high system throughput and to meet the application needs within available computing resources.

Kohonen network or Self-Organizing Map (SOM) is a type of neural network that works with unsupervised training. Its takes input data and trains itself. Kohonen based his neural network on the associative neural properties of the brain (Kohonen, 2001). Self-Organizing Map is called a topology-preserving map because there is a topological structure imposed on the nodes in the network. Self-organizing behavior is the spontaneous formation of well-organized structures, patterns or behaviors, from random initial conditions.

In this study we address load balancing heuristic based on Self-organizing neural network which allows adaptive and reliable management and scheduling of tasks. Load balancing is accomplished by scheduling of the tasks.

The basic description of problem domain is as follows: Resource management and scheduling in Grid computing environment is a very complex task. Thousands of jobs are submitted by users without knowing where they will execute. When a job arrives, the decision maker must decide where it should be served on a given set of processors in order to maximize or minimize the given performance measure. Load balancing can be static or dynamic. Static load balancing determines the

Corresponding Author: Ammarra Manzoor, Faculty of Basic and Applied Science, International Islamic University, Block 2, Sector H-10, Islamabad, Pakistan

task and data distribution at compile time. It is useful only to problems that have predictable resource requirements and load variations. Although it can solve many problems i.e., those caused by processor heterogeneity and non uniform loops for most regular applications, the transient external load due to multiple users on a network of workstations necessities a dynamic approach to load balancing. With dynamic load balancing work is assigned to nodes at run time and information about the status of the node and application can be used to optimize the task.

Traditional load balancing algorithms make several simplifying assumptions; (I) completion time of a job is not affected by the loads on resources other than the CPU; (ii) some moving average of CPU queue length is a significant determinant of completion time and (iii) simple decision rules such as always sending a job to the least loaded site, can be determined a priori and perform well in reality. Load balancing for heterogeneous applications is difficult because different tasks have different costs and data dependencies between the tasks can be very complex. It creates new challenge including dynamic and unpredictable behavior, multiple administrative domains. Due to highly heterogeneous and complex computing environments, effective load balancing is a very difficult problem even though if dynamic load balancing is used, load balancing is very challenging to achieve. Load imbalance is another serious problem in which load from heavily loaded nodes is migrated to lightly loaded ones by the use of efficient load balancing. Traditional approaches to the load balancing are either overly conservative or not portable as they require a human designer to specify a formula for computing load, the load index as a function of the current and recent utilization levels of various resources. They also require manual setting of all policy parameters and are not efficient as they are not adaptive to the current state of grid. They are also sensitive to installation-specific characteristics of hardware devices as well as to the prevalent load patterns. Load average value cannot guarantee that if a site with lower load average value is given a job, it will complete earlier than another with higher value. They also ignore resources other than the CPU. All these and other drawbacks make it necessary to make use of artificial intelligence and machine learning in load balancing.

The literature surveyed for analysis of problem domain is as follows: Load balancing architecture that can deal with applications with heterogeneous tasks is presented by Nishikawa and Steenkiste (1993). However it lacks in several areas e.g., the architecture has to be evaluated using more complex applications and larger systems. Additionally Interface of the current system is at low level and user has to specify large no of functions.

Mehra and Wah (1993) demonstrated automated learning of meaningful load-index functions from workload data. The approach uses comparator neural networks, one per site, which learns to predict the relative speedup of an incoming job using only the resource utilization patterns, observed prior to the job's arrival. The learning algorithm overcomes the lack of job-specific information by learning to compare the relative speedups of different sites with respect to the same job, rather than attempting to predict absolute speedups.

Siegall and Steenkiste (1994) presented an architecture for a system that supports the automatic generation of parallel programs with dynamic load balancing. The measurements demonstrate that load balancing overhead can be kept low by proper adjustment of load balancing parameters and that the load balancer can rapidly adjust the work distribution in a heterogeneous environment. The results also showed that techniques that overlap load balancing with computation are effective in reducing load balancing overhead. However the nature of algorithm is best suited for cluster computing rather than load-balancing.

Zaki *et al.* (1997) examined the behavior of global vs. local and centralized vs. distributed load balancing strategies. Different schemes were analyzed for various applications under varying program and system parameters. A hybrid compile-time and runtime modeling and decision process is presented which customizes the best scheme along with automatic generation of parallel code with calls to runtime library for load balancing. However the method requires use of compilers specific to certain machines and further suffers from the case that nodes themselves cannot act as server thereby reducing scalability.

Parallel incremental scheduling, a new approach for load balancing is presented by Wu (1995). It combines the advantages of static scheduling and dynamic scheduling, adapts to dynamic problems and produces high quality load balance. However the presence of central control makes it unsuitable for grid environments.

Bevilacqua (1999) introduces a method to obtain efficient load balancing for data parallel applications through dynamic data assignment and a simple priority mechanism on a heterogeneous cluster of workstations assuming no prior knowledge about the workload. This strategy reduces the overhead due to communication so that it could be successfully employed in other dynamic load balancing approaches. While this algorithm is suitable for data centric applications/grids but performance detoriate rapidly for computation centric applications/grids.

Ichikawa and Yamashita (2000) describes static load balancing scheme for partial differential equation solvers in a distributed computing environment. This method considers both computing and communication time to minimize the total execution time with automatic data partitioning and processor allocation. However, large scale and non-embarrassingly parallel applications are not discussed. The taxonomy for describing and classifying growing number of different load balancing techniques is presented by Osman and Ammar (2002). The potential benefit of sharing jobs between independent sites in a grid computing environment is discussed by Ernemann (2002). In this study the usage of multi-site applications leads to even better results under the assumption of a limited increase on job execution time due to communication overhead. The results showed that the collaboration between sites by exchanging jobs even without multi-site execution significantly improves the average weighted response time. Schopf (2002) presented a general architecture for scheduling on grid. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed and information about the systems is often limited or dated. Furthermore, the idea has not been put to practical use and experimental results limits to well behaved, embarrassingly parallel examples.

Karatza (2003) presented load sharing and job scheduling in a Network of Workstations (NOW). Along with traditional methods of load sharing and job scheduling, it also examines methods referred to as epoch load sharing and epoch scheduling, respectively. Again the algorithm is not suitable for highly flexible organization such as grid.

Load sharing policies in a heterogeneous distributed system is investigated by Karatza and Hilzer (2002), where half of the total processors have double the speed of the others. Processor performance is examined and compared under a variety of workloads. A priori knowledge of job execution time is not considered in this paper which leads to high overheads for estimating job times. Further for quite a long initial time the algorithm gives erroraneous result. Yu *et al.* (2005) presented a prediction-based scheduling algorithm that predicts task execution time and then allocates tasks among workers according to the predictions. This leads to high overheads for estimating job times. Further for quite a long initial time the algorithm gives erroraneous results.

Cao *et al.* (2003) presented an agent based grid management infrastructure is coupled with a performance-driven task scheduler that has been developed for local grid load balancing. The agents require extensive interference at the client end thus reducing the effectiveness. The impact of fluctuations in the processing speed on the performance of grid applications is presented by Dobber *et al.* (2004). Experiments shows that burstiness in the processing speed has a dramatic impact on running times which heightens the need for dynamic load balancing schemes to realize good performance. Another aspect on which more research has to be done on the aspect of selecting the best predicting methods for processor speeds. Hamidzadeh *et al.* (1995) described a general model for describing and evaluating heterogeneous systems that considers the degree of uniformity in the processing elements and the communication channels as a measure of the heterogeneity in the system. The performance of a class of centralized scheduling algorithms referred to as SASH in computing environments with different degrees of heterogeneity is investigated. This approach was compared over DSS in highly heterogeneous system and the work demonstrated that, even with a small no of processors, performing a sophisticated scheduling technique on dedicated processor can produce substantial improvements in total execution times. El-Abd and El-Bendery (1997) proposed a novel neural based solution to the problem of dynamic load balancing in homogeneous distributed systems. The Winner-Take-All (WTA) neural network model is used for implementing the selection and location policies of a typical dynamic load balancing algorithm. Again the process is well suited if initial estimations are highly accurate which in uncentralized environment like grid computing is not possible.

Labonte and Quintin (1999) presented the implementation of self-organizing map neural networks on distributed parallel computers consisting of identical and of disparate workstations. The implementation is able to reduce the computational time required by SOMs to a fraction of the time required by a single computer. Issues related to dynamic scheduling have not been discussed. Atun and Gursy (2000) described a new implementation of Kohonen Self-Organizing Map for static load balancing problem and examines variations of the algorithm. The algorithm preserves neighborhood relations. Load balancing is incorporated into SOM algorithm. Because of the high degree of retention required in building neighborhood trees it is difficult to use the algorithm for larger No. of nodes.

Gursoy and Ammar (2000) implemented static load balancing algorithm based on Self-Organizing Maps (SOM) for a class of parallel computations where the communication pattern exhibits spatial locality. The communication overhead can be reduced if the physically nearby and heavily communicating tasks are mapped to the same processor or to the same group of processors.

However issues related to dynamic load balancing has been overlooked. Maris described back propagation neural network based dynamic load balancing algorithm which distributes remote procedure calls among cluster servers, based on statistics about execution time of remote procedure calls. The implementation was used in cluster of Enterprise Java Beans containers, although the described approach can be used with any type of distributed components that uses synchronous remote invocation protocol, for example, WEB services. Neural network based approach was compared to traditional static algorithms of EJB load balancing. In test cases the presented algorithm produced up to 176% performance increase compared with Round-Robin load balancing policy. However this algorithm is much slower than static load balancing algorithms. Neural network causes significant overhead.

MATERIALS AND METHODS

The methodology comes as a solution to the problems mentioned in problem domain. Load balancing is very difficult and challenging task to achieve. We consider load balancing in the following manner: Assume a set of n parallel heterogeneous nodes (N = 1,2,3…n) and a set of m independent jobs (J = 1,2,3...m); the jobs arrive one by one, where each job has an associated load vector and has to be assigned to exactly one of the nodes, thereby increasing the load on that node. The main objective of load distribution is the division of workload amongst available group of nodes in such a way so that overall completion time of the parallel program is minimized. The workload of a node consists of the combined demands of resources from all of the local processes. The task of load balancing can be viewed as a strategy-learning task, which can be decomposed into learning of load indices and policies. The load indices are used by the policies to make decisions to balance the load. It is hard to find an optimal solution to achieve load balancing for a specific application due to dynamic nature and non-reliability of computational environment.

Initially each node has a list of jobs to be executed and the time when they have to start. Based on the job's characteristics as well as information about load history at various sites, it is determined where to execute each incoming job. When job reallocation is required, the appropriate jobs will be selected from the job queue on a node and transferred to another node. A job is migrated from one node to another; so its net effects are reduced load on resources local to the originating node and increased load on resources local to the remote node.

We incorporated machine learning and artificial intelligence as a vehicle for automation of load balancing. They have an ability to learn how to do tasks based on the data given for training or initial experience Since Neural Nets are best at identifying patterns or trends in data. Their ability to learn by examples makes them very flexible and powerful. Either humans or other computer techniques can use neural networks with their remarkable ability to derive meaning from complicated or imprecise data, to extract pattern and detect trends that are too complex to be noticed. A trained neural network can be thought of as an expert in the category of information it has been given to analyze.

Neural networks with their remarkable ability to derive meaning from complicated or imprecise data can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an expert in the category of information it has been given to analyze (Stergiou, 1996).

The technique of Neural Networks which we adopted for load balancing in Grid is Kohonen Maps. The Kohonen Map is a Self-Organizing Map of Neural Network meaning that no expected output exists to judge by and the network finds and reinforces patterns on its own.

Functional modules: Functional modules of the system are as follows:

Resource collector: Resource Collector is the most significant module of our grid. As it acts as a daemon and without this daemon the working of the system is impossible. Its functionalities are as follows:

- Gathers resource information in grid.
- It creates and save the log file on the basis of accumulated information from each node present in grid.
- Start or stop the Resource Monitor and Analyzer, Task Controller and KNN Load Balancer daemon.
- It parses the generated log file according to the conditions.

Resource monitor: It checks the resource information accumulated by Resource Collector, displays it, arrange this information in user readable form. The information is updated after the specified time period given by user.

Resource analyzer: It display load statistics, read the log file generated by Resource Collector and generate

dynamic graphs for load, memory, time and status. The graphs are updated after regular time period.

KNN load balancer: This is another important module of our grid. The features of KNN Load Balancer are as follows:

- Read the parsed log file with respect to the parameters required by KNN-Learning. In short it collects the offline information of resources.
- Initialize learning rate and neighborhood parameters.
- Initialize neurons (nodes) with random values.
- Euclidean distance of input neurons is calculated to find the winner. Weights of the neuron are updated and the process is repeated to find another winner.
- Perform task rendering and provide output results.
- Forward the task and optimal nodes information to Task-Mapping Engine, Process migration is also performed from this engine.

Task controller: It has three sub modules.

Task collector: It acknowledges tasks from external environment and keep them in a queue and writes task information in log file. It parses the log file to get required task information.

Task administrator: It gives interface to user for communication with our grid. Handles user requests for load task, edit task, save task, create task and remove task.

Task monitor

- It examines all tasks present in queue and gives the status for each individual task, whether it is running, finished, or about to finish.
- It matches task completion time with its approximate weighted factor in order to get the efficiency.

Performance monitor

- It extracts resource information from resource monitor and task information from task monitor.
  - In case of load imbalance pass the current extracted information to KNN Load Balancer in order to balance the load on grid (Fig. 1).

ALGORITHM

Step 1: Initialize size of network, no of iterations.

Step 2: Set initial and final values of learning rate α and neighborhood radius θ, respectively.

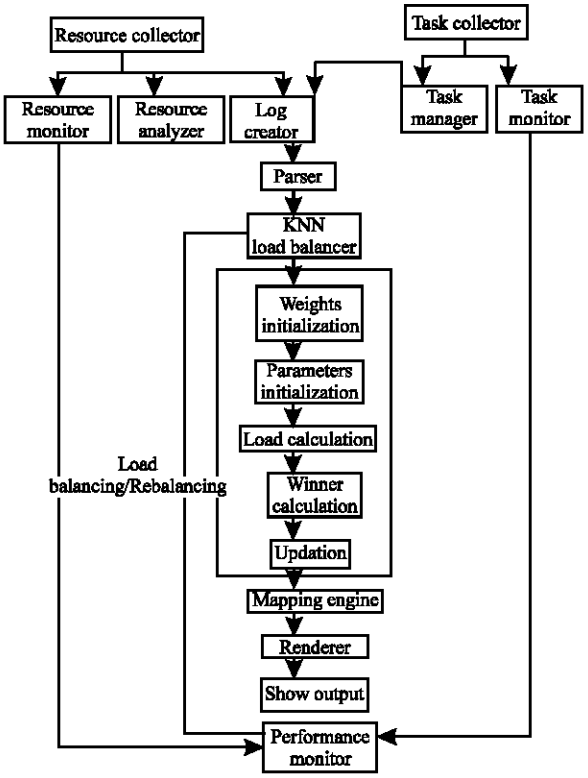

Fig. 1: Architecture diagram

Step 3: Initialize the weight vectors w_{ij} of the nodes randomly.

Step 4: Calculate the load of each node by using the formula

Load = current CPU utilization/total possible CPU utilization

Step 5: While $(t \neq t_{max})$ do

- Select lightly loaded node from the nodes.
- Select random input vector X from the input space S.
- Compute the Euclidean distance of each neuron j by using the formula

$$D(j) = \sum_i (W_{ij} - X_i)2$$

- Find Index j such that D(j) is minimum.
- Update weight vectors of nodes by using the formula

$$W_{ij}\ (new) = W_{ij}\ (old) + \alpha\ [X_i - W_{ij}\ (old)]$$

Step 6: Update learning rate.

Step 7: Reduce neighborhood radius.

Step 8: Update load of each node.

Step 9: Repeat from step 5 until max iterations is not reached.

EXPERIMENTAL SETUP

Our system consists of 100 nodes with Linux Operating system on them interconnected via Ethernet LAN. As a fundamental base we have adapted openmosix as the process/task monitoring and management tool. We benchmark two computationally intensive applications which are matrix multiplication and PovRay. For the sake of comparison we compare our results with the case when dynamic scheduler is in place. We also compare our results with static scheduler.

RESULTS

In the Fig. 2 KNN is compared with other dynamic algorithms and we see that execution time starts decreasing, as the No. of processors increases thus improving the performance as compared to others.

In Fig. 3 initially we see that execution time of KNN is higher but as No. of processors increases the difference begins to fall off. And its performance is better as compared to other dynamic algorithms. This is because KNN consumes lot of time for training.

In Fig. 4 and 5 the general trend is same but in Fig. 4 we see that KNN takes less time in the beginning as it is trained but as no. of tasks increases, KNN takes more time to complete the job as the no. of processors is also fixed. But the performance of KNN is still better than other dynamic algorithms. KNN incurs some delay as No. of tasks increases due to context switching between the processors.

Figure 6 and 7 show that the general trend is same as before except we see that KNN outperforms static scheduler after some time. We also see that the difference

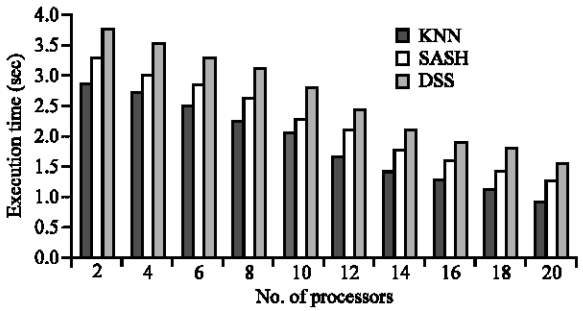

Fig. 2: KNN (with training) vs. other dynamic algorithms

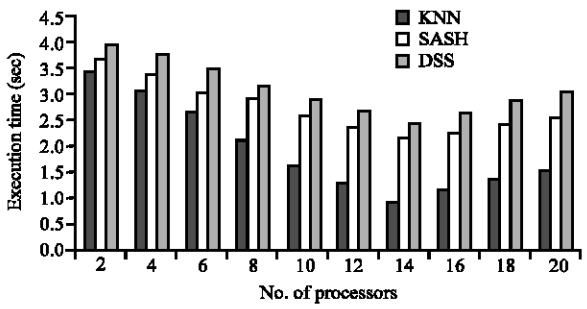

Fig. 3: KNN (no training) vs. other dynamic algorithms

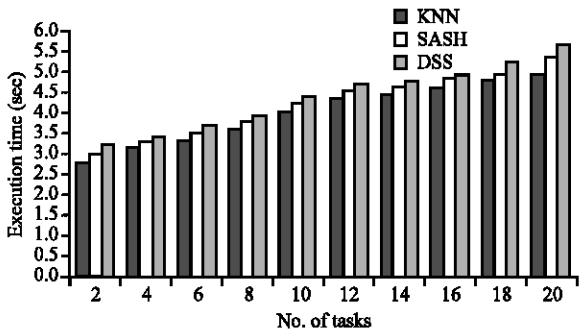

Fig. 4: KNN (with training) vs. Other dynamic algorithms w.r.t No. of tasks

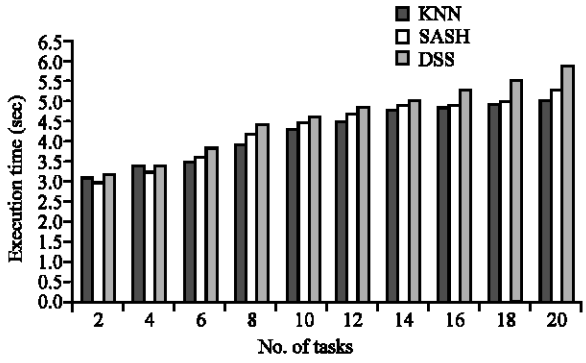

Fig. 5: KNN (no training) vs. other dynamic algorithms w.r.t No. of tasks

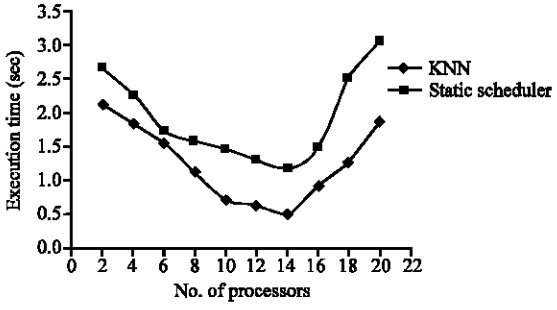

Fig. 6: KNN (with training) vs. static scheduler

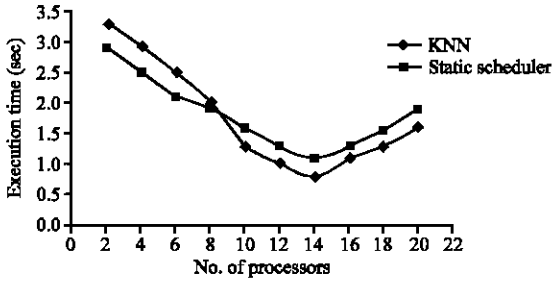

Fig. 7: KNN (no training) vs. static scheduler

is much smaller in the beginning because static scheduler also takes some time for their own calculation.

CONCLUSIONS

In this study we presented a load balancing technique based on artificial neural network for heterogeneous environment for computationally intensive applications. Resource management and scheduling in heterogeneous environment is challenging task to achieve. Load balancing is done in such a way so as to minimize job execution time. We incorporated SOM also known as Kohonen Networks a technique of ANN for automation of load balancing. As the results shows the performance of Kohonen networks with respect to other dynamic algorithms and the case where static scheduler is in place. The results showed that the performance was analyzed for large No. of tasks, processors in response to execution time. And we can see the fluctuation in execution time with varying No. of processors, task sizes, No. of tasks etc. As Kohonen takes much time for training but once it is trained it gives better performance in comparison with other dynamic and Static Schedulers.

REFERENCES

Atun, M. and A. Gürsoy, 2000. A new load-balancing algorithm using self-organizing maps. Euro-Par, 2000. Parallel Processing. 6th Intl. Euro-Par Conf., Germany.

Bevilacqua, A., 1999. A dynamic load balancing method on a heterogeneous cluster of workstations. Informatica (Slovenia), 23: 49-56.

Cao, J., D.P. Spooner, S.A. Jarvis and G.R. Nudd, 2003. Agent-based grid load balancing using performance-driven task scheduling. Proceeding of 17th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003); Nice, France.

Dobber, M., G. Koole and R. van der Mei, 2004. Dynamic load balancing for grid applications. 11th International Conference Proceedings HiPC; Bangalore, India.

El-Abd, A.E. and M.I. EI-Bendary, 1997. A neural network approach for dynamic load balancing. In: Homogeneous Distributed Systems. 30th Hawaii International Conference on System Sciences (HICSS) Vol. 1: Software Technology and Architecture.

Ernemann, C., V. Hamscher, U. Schwiegelshohn and R. Yahyapour, 2002. On advantages of grid computing for parallel job scheduling. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02).

Foster, I. and C. Kesselman, 1999. The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers.

Foster, I., C. Kesselman and S. Tuecke, 2001. The anatomy of the grid-enabling scalable virtual organizations. Int. J. High Performance Computing Appl., 15: 200-222.

Foster, I., 2002. What is the Grid? A Three Point Checklist. GRIDToday, pp: 100-136.

Foster, I., C. Kesselman, J.M. Nick and S. Tuecke, 2002. The physiology of the grid: An open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum.

Gürsoy, A. and M. Atun, 2000. Neighborhood preserving load balancing: A self-organizing approach. Euro-Par Parallel Processing, LNCS 1900, pp: 324-341.

Hamidzadeh, B., D.J. Lilja and Y. Atif, 1995. Dynamic scheduling techniques for heterogeneous computing systems. Concurrency: Practice and Experience, 7: 633-652.

Ichikawa, S. and S. Yamashita, 2000. Static load balancing of parallel PDE solver for distributed computing environment. Proceeding ISCA 13th Internaroinal Conf. Parallel and Distributed Computing Systems (PDCS2000), pp: 399-405.

Karatza, H.D. and R.C. Hilzer, 2002. Load sharing in heterogeneous distributed systems. Proceedings of the 2002. Winter Simulation Conference (WSC'02). Vol. 1.

Karatza, H.D., 2003. A comparison of load sharing and job scheduling in a network of workstations. Int. J. Simulation: Systems, Science Technology, UK Simulation Society, 4: 4-11.

Kohonen, T., 2001. Self-Organization and Associative Memory. 3rd Extended Edn., Springer-Verlag, Berlin Heidelberg.

Labonte, G. and M. Quintin, 1999. Network parallel computing for SOM neural networks. In: The Proceedings of the High Performance Computing Symposium, 1999.

Mehra, P. and B. Wah, 1993. Automated learning of workload measures for loadbalancing on a distributed system. Proceeding of the International Conference on Parallel Processing, CRC Press, 3: 263-270.

Nishikawa, H. and P. Steenkiste, 1993. A general architecture for load balancing in a distributed memory environment. Proceeding of 13th Int. Conf. Dist. Computing Systems, IEEE, Pittsburgh, pp: 47-54.

Osman, A. and H. Ammar, 2002. Dynamic loadbalancing strategies for parallel computers. Scientific, 11: 110-120.

Schopf, J.M., 2002. A general Architecture for scheduling on the grid. Special Issue of JPDc on grid computing, April, 2002.

Siegell, B.S. and P. Steenkiste, 1994. Automatic generation of parallel programs with dynamic load balancing. Proceeding of the 3rd IEEE International Symposium on High Performance Distributed Computing. San Francisco, CA.

Stergiou, C., 1996. What is a Neural Network. http://wwwhomes.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/cs11/article1.html.

Wu, M.Y., 1995. Parallel incremental scheduling. Parallel Processing Lett., 5: 659-670.

Yu, C., P. Chen and S. Wang, 2005. Adaptive task scheduling algorithms for master-worker applications in grid computing. 11th Workshop on Compiler Techniques for High-Performance Computing.

Zaki, M.J., W. Li and S. Parthasarathy, 1997. Customized dynamic load balancing for a network of workstations. Proceeding 5th IEEE International Symposium on High Performance Distributed Computing, Syracuse, New York.