Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp55-60)

A Neural Network Realization of Scheduling in Grid Computing Environment

MOHAMMAD KALANTARI, KAZEM AKBARI
Computer Engineering Department
Amirkabir University of Technology (Tehran Polytechnic)
TEHRAN-IRAN

Abstract: - The Computational Grids provide a promising platform for efficient execution of computational and data intensive applications. Scheduling in such environments is challenging because target resources are heterogeneous and their load and availability varies dynamically. In this paper, we propose a mathematical neural network based scheduling solution for grid computing environment. Using mathematical method guarantees rapid convergenc that is essential for such environments with proliferation of resources.

Key-Words: - Scheduling, Grid Computing, Neural Networks.

1 Introduction

Grid scheduling is intrinsically more complicated than local scheduling of resources, because it must manipulate large-scale resources across management boundaries. In such a dynamic distributed computing environment, resource availability varies dramatically, so scheduling becomes quite challenging. There have been extensive research activities on scheduling problems in distributed systems that must be extended for the purpose of grid computing environment [1,2,3].

Most problems in scheduling area are NP-Complete. This fact implies that an optimal solution for a large scheduling problem is quite time-consuming. Therefore, some researchers translated the job-scheduling problem into a format of linear programming or K-out-of-N rule and mapped it into an appropriate neural network structure to obtain a reasonable solution [4,5,6].

Neural networks for combinatorial optimization problems were first introduced by Hopfield and Tank in 1985 [7]. They used the predefined energy function *E* which follows the quadratic form:

$$E = \sum_{i=1}^{N}\sum_{j=1}^{N} W_{ij} V_i V_j + \sum_{i=1}^{N} V_i I_i \qquad (1)$$

where W_{ij} is the strength of a synaptic link between the *i*th and the *j*th neuron where the condition of $W_{ij} = W_{ji}$ must be always satisfied. Note that I_i is constant bias of the *i*th neuron. Hopfield gives the motion equation of the *i*th neuron:

$$\frac{dU_i}{dt} = -\frac{U_i}{\tau} - \frac{\partial E}{\partial V_i} \qquad (2)$$

where the output follows the continuous, nondecreasing, and differentiable function called sigmoid function:

$$V_i = f(U_i) = \frac{1}{2}\left(\tanh(\lambda_0 U_i) + 1\right) \qquad (3)$$

where λ_0 is constant and is called *gain* which determines the slope of the sigmoid function.

Since Wilson and Pawley strongly criticized the neural network methods (specifically Hopfield model) for optimization problems [8], and in addition, after the publication of discouraging report of Paielli [9] regarding the drawbacks of Hopfield nets (e.g., convergence to local minima, limited capacity of network, and disability for solving hard-learning problems), it has been widely believed that the neural network methods are not suitable for optimization problems. But Takefuji and others in their continuous and unfailing efforts have demonstrated the capability of the artificial neural networks (i.e., Hopfield-like method) for solving optimization problems, over the best known algorithms and methods. They found that the use of decay term $(-U_i/\tau)$ in Eq.(2) increases the computational energy function *E* under some conditions instead of decreasing it [10].

In his method, Takefuji exploits the topology of Hopfield net in conjunction with both mathematical method of *McCulloch-Pitts* (with or without

hysteresis) and Maximum (winner-take-all) function to tackle and solve the problems [10]. His works cover a wide variety of professional fields including game theory, computer science, graph theory, molecular biology, VLSI computer aided design, communication, and computer networks.

In [1] Yueh-Min Huang et al. represented a Hopfield neural network based solution to scheduling multiprocessor job with resource and timing constrains. In their model, they assume that all resources are homogeneous and available for scheduling at time 0, and during scheduling, no resources are added to or deleted from system, which is rational for such a system. However in Grid computing environments, resources are heterogeneous and can be added or deleted dynamically. This work aims to overcome these new constraints by using mathematical neural model rather than Hopfield neural method.

The rest of this paper is organized as follows. Section 2 contains an overview of the mathematical neural network model. In section 3 our grid computing environment is described. In section 4 the scheduling problem is described in detail and mapped onto a neural network, followed by simulation results in section 5. Finally, we will summarize the outcomes and future work in section 6.

2 Mathematical Neural Network Model

The mathematical model of the artificial neural network consists of two components; neurons and synaptic links. The output signal transmitted from a neuron propagates to other neurons through the synaptic links. The state of the input signal of a neuron is determined by the linear sum of weighted input signals from the other neurons where the respective weight is strength of the synaptic links. Every artificial neuron has the input U and the output V. The output of the ith neuron is given by $V_i = f(U_i)$ where f is called the neuron's input/output function. The interconnections between the ith neuron and other neurons are determined by the motion equation. The change of the input state of the ith neuron is given by the partial derivations of the computational energy function E with respect to the output of the ith neuron where E follows an n-variable function: $E(V_1,V_2,...,V_n)$. The motion equation of the ith neuron is given by:

$$\frac{dU_i}{dt} = -\frac{\partial E(V_1,V_2,...,V_n)}{\partial V_i} = -\frac{dE}{dV_i} \qquad (4)$$

In general, the goal of neural computation is to optimize the fabricated computational energy function. The energy function not only determines how many neurons should be used in the system but also it specifies the strength of synaptic links between neurons. Indeed, energy function is constructed from information in the given problem, considering the required constraints and/or cost function. Practically, it is usually easier to calculate the motion equation (partial differential of the energy function) than the energy function itself. The superiority of the motion equation over energy function can be articulated as follows: its simplicity (step by step computations and ease of formulation), binary behavior, ease of application, and the flexibility in which all constraints can be incorporated. It also resolves the deficiencies of the Hopfield net which was discussed earlier. The energy function, however, can be defined:

$$E = \int dE = -\int \frac{dU_i}{dt} dV_i \qquad (5)$$

In order to numerically solve the partial differential equation or the differential equation to determine the value of motion equation, the first order *Euler method* is widely used where it is the simplest among the existing numerical methods. Therefore, based upon the first order Euler method the value of $U(t+1)$ is determined as below:

$$U(t+1) = U(t) + \Delta U(t).\Delta t \qquad (6)$$

where $\Delta U(t)$ is given by Eq. (4) and termination condition is given by:

$$\Delta U(t) = 0 \qquad (7)$$

The condition of the Eq. (7) implies that the constraints are all satisfied.

3 System Model

The Grid computing environment in this work (shown in Fig.1) consists of n sites, $S_1, S_2, ...S_n$; where a site $S_i (1 \le i \le n)$ consists of a number of heterogeneous computational resources. Within each local site, there is a forecasting system such as PACE (Performance Analysis and Characterization Environment) toolkit [11,12] to predict the Job's execution time on the candidate resources prior to run time. Selection of candidate resources is

accomplished via a selector component in each site. Candidate resources are those which are available at the time or within a predetermined time. To avoid the race condition we can reserve the candidate resources. The scheduling mechanism in our work is done in batch-mode, in which arriving jobs are collected at prescheduled times and are scheduled as a meta-task when a scheduling event is triggered. The following two steps are accomplished for scheduling; first scheduler sends the characteristics of jobs to each site; and then in each site, selector and predictor components return whatever required for scheduling to the scheduler (i.e. a list of candidate resources and the execution time of each job on each resource). Scheduler uses a neural network based scheduling mechanism (see section 4) along with the information for the scheduling purposes. For the sake of simplicity, we ignore the communication overhead in this work.

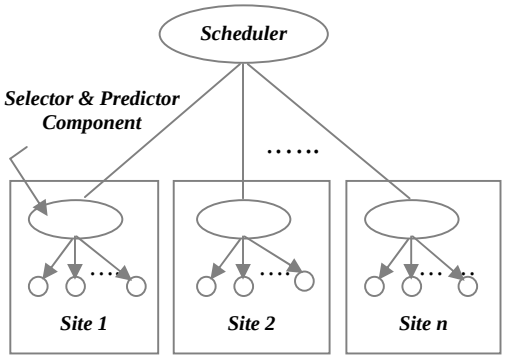

Fig.1 System Model

4 Problem Formulation With Neural Network Method

Our scheduling approach considers N jobs to be run on some machines (resources) and the following assumptions are made regarding the problem domain. First, the execution time of each job on each machine is predetermined (see section 3). Second, a job can not be assigned to different machines, implying that no job migration is allowed between machines. Third, resources are added to or deleted from environment dynamically and the time of addition or deletion is estimated by predictor component. The constraints imposed to our model are a deadline(d_i) for each job *i* and a processing time along with available resources on systems.

Therefore, scheduling parameters comprise the listed jobs, the required machines, and time variables as depicted in Fig.2, where the "*x*" axis denotes the "*job*" variable within a range from 1 to N (the total number of jobs to be scheduled) and the "*y*" axis represents the "*machine*" variable within a range from 1 to M (the total number of machines) and the "*z*" axis is for the "*time*" variable and *k* represents a specific time which should be less than or equal to T, the amount of total scheduling time. Thus, a state variable V_{ijk} is defined to indicate whether or not the job *i* is executed on machine *j* at a certain time *k*. In case of $V_{ijk}=1$, it denotes that the job *i* is run on machine *j* at the time *k*; otherwise, $V_{ijk}=0$. It should be noted that, each V_{ijk} corresponds to a neuron of the neural network.

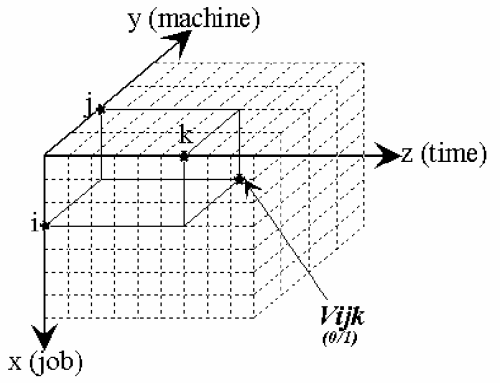

Fig.2 Three-dimensional modeling of problem

Mapping constraints into the motion equation is as follows:

$$\frac{dU_{ijk}}{dt} = -A*\sum_{\substack{i_1=1 \\ i_1\neq i}}^{N} V_{ijk}V_{i_1jk} \qquad \text{-1}^{st}\ \textit{inhibitory factor}$$

$$-B*\sum_{\substack{j_1=1 \\ j_1\neq j}}^{M}\sum_{k_1=1}^{T} V_{ijk}V_{ij_1k_1} \qquad \text{-2}^{nd}\ \textit{inhibitory factor}$$

$$-C*L\left(2*\left(\sum_{k_1=1}^{T} V_{ijk_1} - P_{ij}\right)\right) \qquad \text{-3}^{rd}\ \textit{inhibitory or 1}^{st}\ \textit{excitatory factor.}$$

$$-D*K\left(2*\left(\sum_{i_1=1}^{N} V_{i_1jk} - 1\right)\right) \qquad \text{-4}^{th}\ \textit{inhibitory factor}$$

$$-E*H\left(k-\left(d_i-1\right)\right) \qquad \text{-5}^{th}\ \textit{inhibitory factor}$$

$$-F * H(P_{ij} - d_i) \qquad \text{-}6^{th} \text{ inhibitory factor} \tag{8}$$

Moreover, the effects of overall inhibitory and excitatory factors in the motion equation can be summarized as :

- The first inhibitory factor states that a machine *j* can only execute one job *i* at a certain time *k.*
- The second inhibitory factor states that no migration is allowed, on the other word, job *i* can only be executed on machine *j* or machine j_1 at any time.
- The third inhibitory factor or the first excitatory factor states that the time spent by job *i* should be equal to P_{ij}, where P_{ij} is the estimated execution time of job *i* on machine *j*.
- The fourth inhibitory factor states that only one job can be executed on a specified machine and at a certain time.
- The fifth and sixth inhibitory factors state that no violation of deadline is allowed.

The functions *L*, *K* and *H* is defined as follow:

$$L(\alpha) = \begin{Bmatrix} 1 & \alpha > 0 \\ 0 & \alpha = 0 \\ -1 & \alpha < 0 \end{Bmatrix}, \quad K(\alpha) = \begin{Bmatrix} \alpha & \alpha \geq 0 \\ 0 & \alpha < 0 \end{Bmatrix},$$

$$H(\alpha) = \begin{Bmatrix} 1 & \alpha > 0 \\ 0 & \alpha \leq 0 \end{Bmatrix}$$

we have used *hysteresis McCulloch-Pitts* neuron model where the input/output function of the *i*th hysteresis neuron is given by:

$$V_i = f(U_i) = \begin{Bmatrix} 1 & if \quad U_i > UTP \\ 0 & if \quad U_i < LTP \\ unchenged & otherwise \end{Bmatrix} \tag{9}$$

where *UTP and LTP* are upper trip point and lower trip point respectively, as well as a modified form of *Maximum Neuron Model*. Maximum neuron model is composed of *M* cluster where each cluster consists of n neurons. In this model "winner-take-all" function is embedded. This implies that one and only one neuron out of n neurons in every cluster with a maximum value is encouraged to be fired. The corresponding input/output function of the *i*th neuron in the *m*th cluster is given by:

$$V_{mi} = \begin{Bmatrix} 1 & if \quad U_{mi} = \max\{U_{m1},...,U_{mn}\} \quad and \\ & U_{mi} \geq U_{mj} \quad for \quad i > j \\ 0 & otherwise \end{Bmatrix} \tag{10}$$

In the maximum neuron model it is always guaranteed to generate satisfactory solutions [10]. The advantages of the maximum neural model can be summarized [10]:

- Every local minimum is one of the acceptable solutions, while other existing neural models cannot guarantee that.
- Tuning of coefficient parameters for the activation function is not required.
- The termination condition of the equilibrium state is clearly defined by a simple mathematical formula.

It should be noted that there are some important points in our method, which can be summarized in the following:

- Although it is possible to find the exact form of energy function *E*, but we do not need it in the process because of the motion equation $dU/dt = -dE/dV$. When we use dU/dt to find a new state of the system, the energy function will be employed implicitly.
- The termination condition and the maximum neuron model together cause the convergence of system in the global minimum.
- By using the motion equation method, in fact we are employing a sparsely connected network whenever we need that, as opposed to the Hopfield's fully connected networks. Therefore, for the big problems there is no need to employ a huge number of synaptic connections which would cause a spurious convergence.

The corresponding algorithm is given in Fig.3.

5 Simulation Results

The neural network based scheduling algorithm described in the previous sections has been implemented in C and executed on computer with Pentium IV processor and 512 mega byte RAM. For the evaluation of the system's performance, different examples of all sizes: small, medium and large were run for 10 times. Some of the experiments and the overall results are listed below.

The input data for the first, the second, and the third experiments and their corresponding results have been shown in Fig.4, 5 and 6 respectively.
In the first example, some machines are not available at time 0. We compute the maximum execution time of each job on each machine and use these values as inputs to the algorithm.

Algorithm:
begin
initialize U_{ijk}s randomly.
while *(a set of conflicts is not empty)* ***do***
begin
for *i:=1 to N* ***do***
for *j:=1 to M* ***do***
for *k:=1 to T* ***do***
begin
$U_{ijk} = U_{ijk} + \Delta U_{ijk}$
$V_{ijk} = f(U_{ijk})$
end

for *i:=1 to N* ***do***
begin
1. find the maximum U_{ijk}
2. determine a segment (j) in which the maximum U_{ijk} exists. If there is tie remove it randomly.
3. The output of the other segment's neurons is set to zero.
end
end
end.

Fig.3 The Scheduling Algorithm

	m_0	m_1	m_2
Job_0	7	10	6
Job_1	5	2	2
Job_2	10	12	20
Job_3	1	4	12
Job_4	7	11	2
Job_5	1	6	10
Job_6	6	3	10

maximum finishing time of each job on each machine	m_0	m_1	m_2
Job_0	10	12	6
Job_1	8	4	2
Job_2	13	14	20
Job_3	4	6	12
Job_4	10	13	2
Job_5	4	8	10
Job_6	9	5	10

	m_0	m_1	m_2
time of availability	3	2	0

	Job_0	Job_1	Job_2	Job_3
deadline	16	2	16	16
	Job_4	Job_5	Job_6	
deadline	16	16	16	

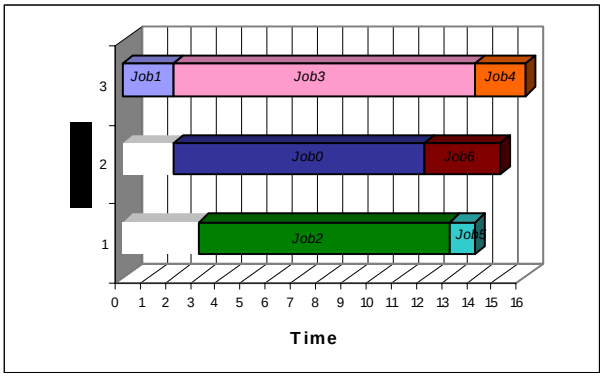

Fig.4 The first experiment

However if the deadlines of all jobs are set to 15 (The minimum makespan), our algorithm can't be converged.
In Fig.6 no constraints on availability imposed but the problem size has been increased and the deadlines of all jobs have been set to 8.

	m_0	m_1	m_2
Job_0	5	4	8
Job_1	20	5	3
Job_2	6	10	4
Job_3	10	4	2
Job_4	20	6	5

	m_0	m_1	m_2
time of availability	0	0	0

	Job_0	Job_1	Job_2	Job_3	Job_4
deadline	6	10	7	10	10

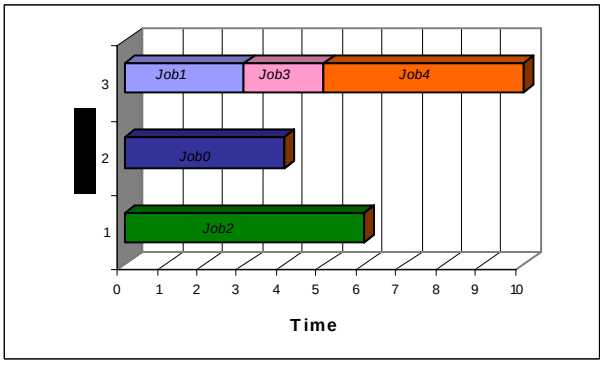

Fig.5 The second experiment

	m_0	m_1	m_2	m_3	m_4	m_5
Job_0	10	9	11	12	2	12
Job_1	4	8	12	16	1	66
Job_2	8	10	13	11	4	50
Job_3	20	12	14	10	8	4
Job_4	40	14	16	9	14	3
Job_5	12	4	15	14	80	1
Job_6	6	3	22	17	44	14
Job_7	2	12	34	18	52	17
Job_8	3	16	4	10	61	12
Job_9	1	14	4	4	12	19
Job_{10}	10	20	8	3	44	20

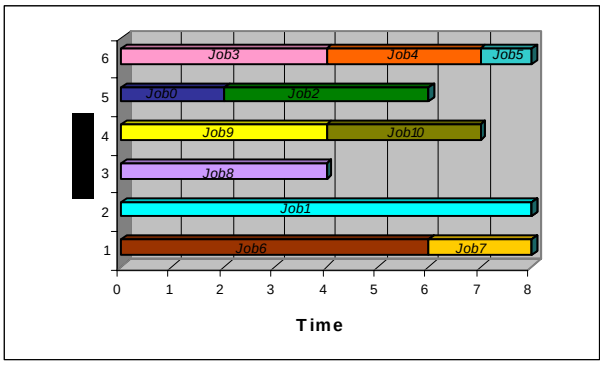

Fig.6 The third experiment

Table 1 shows the average number of iterations and execution time of algorithm for different examples of different sizes. The number of iteration and the execution time of program decrease or increase smoothly when the problem size increases, as shown in Table 1.

No.	*# of Job*	*# of machines*	*Avg # of Iteration*	*Avg Conv. Time(sec.)*
1	7	3	280	<1
2	5	3	192	<1
3	11	5	65	<1
4	20	15	852	6
5	100	50	1152	11

Table 1: The overall results

6 Conclusion and Future Work

We have presented a neuro-based scheduling solution for grid computing environment. As mentioned before, if all machines are available at time 0, our algorithm always gives the scheduling solution with respect to the given deadlines. Anyway, rapid convergence is an important characteristic of the proposed solution.

In the future, a neuro-based scheduling algorithm which includes communication overhead will be presented.

References:

[1] Y. M. Huang and R. M. Chen, Scheduling multiprocessor job with resources and timing constraints using neural networks. *IEEE Transaction on system, man, and cybernetics*, Vol. 29, No. 4, August 1999.

[2] T. D. Braun, Howard Jay Siegel, Noah Beck, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *Journal of Parallel and Distributed Computing*, 2001.

[3] N. Fujimoto and K. Hagihara, A comparison among grid scheduling algorithms for independent coarse-grained tasks. *IEEE Int. Symp. on Applications and the Internet Workshops (SAINTW'04)*, 2004.

[4] Y. P. S. Foo and Y. Takefuji, Integer linear programming neural networks for job-shop scheduling, *IEEE Int. Conf. Neural Networks*, 1991, pp. 1361-1366.

[5] C. Y. Chang and M. D. Jeng, Experimental study of a neural model for scheduling job shop, *IEEE Int. Conf. System, Man, Cybernetics*, 1995, Vol. 1, pp. 536-540.

[6] J. M. Gallone, F. Charpillet, and F. Alexandre, Anytime scheduling with neural networks, *Proc. 1NRIA/IEEE Symp.* 1995.

[7] J. J. Hopfield and D. W. Tank, Neural computation of decision in optimization, *Journal of Biological Cybernetics*, Vol. 52, 1985, pp. 141-152.

[8] G. V. Wilson and G. S. Pawley, On stability of the traveling salesman problem algorithm of Hopfield and Tank, *Biological Cybernetics*, Vol 58, 1988, pp. 63-70.

[9] R. A. Paielli, Simulation tests of the optimization method of Hopfield and Tank using neural networks, *NASA Technical Memorandom 101047*, 1988.

[10] Yoshiyasu Takefuji, *Neural Network Parallel Computing*, Kluwer Academic Publishers, 1992.

[11] G. R. Nudd, D. J. Kerbyson, E. Papaefastathiou, J. S. C. Perry and D. V. Wilcox, PACE: A toolset for the performance prediction of parallel and distributed systems, *In International journal of High Performance Computing*, 1999.

[12] L. He, S. A. Jarvis, D. P. Spooner, X. Chen and G. R. Nudd, Dynamic scheduling of parallel jobs with Qos demands in multiclusters and Grids, *In Proc. of 5th IEEE/ACM Int. Workshop on Grid Computing*, 2004.