An Adaptive Load Balancing Algorithm with Use of Cellular Automata for Computational Grid Systems

Laleh Rostami Hosoori and Amir Masoud Rahmani

Department of Computer Engineering
Islamic Azad University, Science and Research Branch
Tehran, Iran
l.rostami@srbiau.ac.ir, rahmani@srbiau.ac.ir

Abstract. Load balancing algorithms play a challenging, complicated, and important role in the performance of computational Grid systems. In this paper, we present a decentralized adaptive load balancing algorithm with use of cellular automata, named LBA_CA. Each computing node in the Grid system is modeled as a cell of proposed cellular automata and can be in four states. Cellular automata (abbreviated to CA) are used for designing a load balancing algorithm for computational Grids because of its distributed and dynamic manner. In addition, such natural properties of CA make LBA_CA an appropriate local load balancing algorithm for each cluster of computational Grids. Due to resource heterogeneity and communication overheads exist in computational Grid systems; we take account of several issues in LBA_CA such as processing power of computing nodes and communication latency. The main goal of our algorithm is to reduce the average response time of arrival jobs. The performance of our algorithm is evaluated in terms of several metrics including the average response time of jobs, processor utilization, percent of executed jobs, and average Off time in relation to considerable variations in transition time, service time, and number of jobs.

Keywords: Load balancing; Cellular Automata; Computational Grid systems; Distributed systems.

1 Introduction

The computational Grid is a promising platform that provides large resources for distributed algorithmic processing [1]. Computational Grid environments promise to support resource sharing and coordinated problem solving in dynamic multi-institutional Virtual Organizations [2]. End users and applications see this environment as a big virtual computing system. The systems connected together by a grid might be distributed globally, running on multiple hardware platforms, under different operating systems and owned by different organizations. However, computational Grids have different constraints and requirements than those of traditional high-performance computing systems, such as heterogeneous computing resources and considerable communication delays [3]. In distributed systems, every node has different processing speed and system resources, so in order to enhance the

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6852, Part I, pp. 419–430, 2011.

utilization of each node and minimize the average response time of jobs, load balancing will play a critical role [4].

In general, load balancing algorithms can be classified as centralized or decentralized. In the centralized algorithms (e.g., [5]), one computing node makes all the load-balancing decisions. In the decentralized algorithms (e.g., [6]), all computing nodes are involved in the load-balancing decisions.

Moreover, Load balancing algorithms are classified as static, dynamic, adaptive, or hybrid, based on the used information. Static load balancing algorithms (e.g., [7]) assume that all required information about computing nodes, jobs, and communication network that influence load-balancing decisions are determined in advance, on the other hand, dynamic load balancing algorithms (e.g., [8] and [9]) attempt to gather the current state information to make more informative load-balancing decisions. Adaptive algorithms are a special type of dynamic algorithms where the parameters of the algorithm and/or the scheduling policy themselves are changed based on the global state of the system [10]. Finally hybrid algorithms (e.g.,[11]) attempt to combine the merits of static and dynamic load balancing algorithms in order to minimize their weak points.

A cellular automaton is a discrete dynamical system that consists of a regular network of definite state automata (cells) that change their states depending on the states of their neighbors, according to a local update rule. All cells change their state simultaneously, using the same update rule. The process is repeated at discrete time steps. It turns out that amazingly simple update rules may produce extremely complex dynamics when applied in this fashion. A well known example is the *Game-of-life* by John Conway [12]. For this reason, CA is often used to model real-world phenomena. CA is also considered as a model of highly parallel and distributed computations in multiprocessor and distributed systems [13]. They were used to find solutions of such problems as scheduling and resource management [14].

This paper presents a load balancing algorithm with using CA. The remainder of the paper is organized as follows: In section 2, an introduction to cellular automata is addresses. Section 3 describes our proposed load balancing algorithm in detail. Section 4 discusses our simulation and results of evaluation. Finally, section 8 concludes this paper.

2 Cellular Automata

Cellular automata are a class of discrete dynamical systems, consisting of an array of nodes (cells) of n-dimension. Each cell can be in one of k different states at a given time t [15]. At each discrete time, each cell may change its state, in a way determined by the local transition rules of the particular CA. The transition rules describe precisely how a given cell should change states, depending on its current state and the states of its neighbors. The neighborhood of a given cell must be specified explicitly.

More precisely, CA consists of 4 major components. At first all cells in the CA constitute the *cellular space*, which may be of any dimension, and is of infinite extent. For example, one-dimensional CA can be visualized as having a cell at each integral point on the real number line [15]. A two-dimensional CA has cells at all points in the plane that has only integral coordinates. Minutely *state set* is a finite set whose

elements are all the possible distinct states of the cells. The state of cell i at discrete time t is denoted by $S_i(t)$. Then *neighbourhood* is defined as the neighbours of each cell. $V_i(t)$ denotes the neighborhood of cell i at time t. Finally, each cell transforms from its current state to a new state (at the next time) based on its current state and the states of its neighbors, according to the transition rules. f denotes the *transition rule* of cell i in (1):

$$S_i(t + 1) = f(S_i(t), V_i(t)) . \quad (1)$$

3 The Proposed Load Balancing Algorithm

We assume that the computational Grid environment consists of three major components including processors, communication network, and jobs. Our Grid system consists of M heterogeneous processors, P_1; P_2;...; P_M. It is assumed each processor P_i has several different attributes. Firstly *processing power (W_i)* is the ratio of the processing power of the processor P_i to the processing power of the reference processor P_{ref} (with a relative W_{ref} equal to 1) in the system. In this paper, we elect the slowest processor as P_{ref}. Minute attribute is *job queue*. It is assumed that each processor has an infinite capacity job queue to store jobs waiting for execution. Finally *neighbours (N_i)* defined as a set of processors that are directly connected to the processor P_i.

The jobs are assumed to be computationally intensive, mutually independent, and can be executed at any processor. Several different attributes are assumed for each job J_j. First attribute is *arrival time (AT_j)* of job J_j. It is assumed that the arrival rate of jobs follows Poisson distribution with mean λ. In other words, the inter-arrival time is exponentially distributed with mean$1/\lambda$. It is assumed that *service time (ST_j)* of job J_j follows an exponential distribution with mean $1/\mu$. We assume that job J_j will be missed if it isn't executed until its *deadline time (DT_j)*. In addition, deadline times are distributed exponentially with a given mean.

One of the important parts of Grid computing system is communication network that connects processors to each other. The network topology is varying, since computational Grid systems are dynamic in nature. Thus, our model assumes no specific topology for the network and generates a random topology. Due to diverse network topologies in computational Grid systems, network heterogeneity also exists.

All items that each processor takes account of during the proposed load balancing algorithm are described below:

- *Transition Time (T_n):* The period of time, at the end of that each cell i of CA transforms from its current state $S_i(T_{n-1})$ to a new state (at the next time) $S_i(T_n)$, in a way determined by transition rules f.
- *$Q_i(t)$:* The number of jobs waiting in the processor P_i's job queue at time t
- *$\lambda_i(T_n)$:* The number of arrived jobs at the processor P_i during the interval $[T_{n-1}, T_n]$
- *$\mu_i(T_n)$:* The number of served jobs at the processor P_i during the interval $[T_{n-1}, T_n]$
- *$l_i(T_n)$:* $Q_i(T_n)$ divided by $\mu_i(T_n)$
- *W_i:* The processing power of P_i in relation to the reference processor P_{ref}
- *$l_N(T_n)$:* The average of normalized load in the neighbours N_i of the processor P_i during the interval $[T_{n-1}, T_n]$ as presented in (2):

$$l_N(T_n) = \frac{\sum_{i \in \text{Neighbourhood}} W_i * l_i(T_n)}{\sum_{i \in \text{Neighbourhood}} W_i} \tag{2}$$

- *$S_i(T_n)$:* The cell state of processor P_i during the interval $[T_{n-1}, T_n]$, denoted as below in (3):

$$S_i(T_n)=(S_{ix}(T_n), S_{iy}(T_n)) \ . \tag{3}$$

- *$Sender_p(T_n)$:* The percent of the processor P_i's neighbours N_i whose $S(T_n)$ are Sender, during the interval $[T_{n-1}, T_n]$ as presented in (4):

$$\text{Sender}_p(T_n) = \frac{\sum_{i \in \text{Neighourhood}} S_{ix}(T_n) * S_{iy}(T_n)}{\sum_{i \in \text{Neighourhood}} i} \ . \tag{4}$$

- *$Receiver_p(T_n)$:* The percent of the processor P_i's neighbours N_i whose $S(T_n)$ are Receiver, during the interval $[T_{n-1}, T_n]$ as presented in (5):

$$\text{Receiver}_p(T_n) = \frac{\sum_{i \in \text{Neighourhood}} \overline{S_{ix}}(T_n) * S_{iy}(T_n)}{\sum_{i \in \text{Neighourhood}} i} \ . \tag{5}$$

- *Threshold:* The load threshold (The default value is 1)
- *CL_j^k:* The communication latency between processor P_i and the processor P_k when the job J_j transferred.
- *$\tilde{l}_k(t)$:* The estimated added load on the processor P_k at time t, as presented in (6):

$$\tilde{l}_k(t) = \frac{\lambda_k(T_{n-1})}{\mu_k(T_{n-1})} * (t - T_{n-1}) \ . \tag{6}$$

- *ST_j^i:* The duration in which the job J_j will expectedly be served on the processor P_i, as denoted in (7):

$$ST_j^i = \frac{ST_j}{W_i} \ . \tag{7}$$

- *$ET_j^i(t)$:* When the job J_j will expectedly leave the system, if it is served on the processor P_i , as described in (8):

$$ET_j^i(t) = \frac{Q_i(t)}{\mu_i(T_{n-1})} + ST_j^i \ . \tag{8}$$

- *$ET_j^k(t)$:* When the job J_j will expectedly leave the system, if it is sent form the processor P_i to P_k to be served, as formulated in (9):

$$ET_j^k(t) = \max(l_k(T_{n-1}) + \tilde{l}_k(t), CL_j^k) + ST_j^k \ . \tag{9}$$

Our goal is to minimize the *Average Response Time (ART)* as calculated in (10):

$$\mathrm{ART} = \frac{1}{\mathrm{N}} \sum_{\mathrm{j=1}}^{\mathrm{N}} \mathrm{ExecutionTime}_{\mathrm{j}} - \mathrm{ArrivalTime}_{\mathrm{j}} \ . \tag{10}$$

Where N is the number of jobs executed by the system, $ExecutionTime_j$ is the time when the job J_j is completely executed, and $ArrivalTime_j$ is the time when the job J_j arrives. As obviously clear, ART takes account of the communication latency, waiting time in queues, and service time.

Our algorithm has proposed a CA with following features. All processors of an arbitrary network are cells of *cellular space*, in other words each processor is considered as a cell. *State set* of CA includes four states. The state of cell i at time t represented by the tuple $S_i(t)$ as in (11), where each element of the tuple can be a binary number (0 or 1):

$$S_i(t) = (S_{ix}(t), S_{iy}(t)) \ . \tag{11}$$

We assumed each cell can be in four states. At first, whenever a processor becomes overloaded in comparison with its neighbours, it is assumed as a *sender* processor. It means the processor will send all incoming jobs to its neighbours, which are in receiver state. This state is represented by tuple (1,1). Minutely whenever a processor becomes balanced in comparison with its neighbours, it is denoted as a *balanced* processor. It means the processor will run jobs, which exist in its queue and will accept new jobs as well. Meanwhile, the processor will refuse jobs sent form sender processors. This state is represented by tuple (1,0). Thirdly whenever a processor becomes under-loaded in comparison with its neighbours, it is assumed as a *receiver* processor. It means the processor will run jobs, which exist in its queue and will accept all new jobs and jobs sent form sender processors. This state is represented by tuple (0,1). Whenever a processor is idle for a while, it will be denoted as an *off* processor. It means the processor will go off. This state is represented by tuple (0,0).

The *neighbourhood* $V_i(t)$ of processor P_i at time t is the set of its neighbours N_i in the communication network. Finally the *transition rule* is the same for all processors as denoted by f in (12) and described in depth in Table 1.

$$S_i(T_n) = f(S_i(T_{n-1}), V_i(T_{n-1})) \ . \tag{12}$$

Table 1. Transition rule

$S_i(T_n)$	$f(S_i(T_{n-1}), V_i(T_{n-1}))$
Sender (1,1)	If $(l_i(T_n) - l_N(T_n)) >$ Threshold
Balanced (1,0)	If $\|l_i(T_n) - l_N(T_n)\| \le$ Threshold
Receiver (0,1)	If $(l_i(T_n) - l_N(T_n)) <$ Threshold or If $S_i(T_{n-1})$ == Off and $Sender_p(T_n) \ge Receiver_p(T_n)$
Off (0,0)	If $(l_i(T_n), \lambda_i(T_n), \mu_i(T_n)) == (0, 0, 0)$ If $S_i(T_{n-1})$ == Off: $S_i(T_n)$ = Off Else: $S_i(T_n)$ will be Off with probability 0.01

Our proposed model focuses on the dynamic and distributed properties of cellular automata, in order to offer an efficient load balancing algorithm. LBA_CA gathers

up-to-date information in each transition time. Nowadays, energy conservation becomes a hot discussion all over the world. So LBA_CA put some computing nodes in the *Off* state under special circumstances. This feature might be a step toward approaching energy conservation.

LBA_CA consists of two procedures. *Transition Procedure* indicates processes that each processor performs in each transition time T_n in order to run transition rule and update its state. *Main Procedure* is invoked whenever a new job J_j arrives to processor P_i at time t. If the processor P_i is in *Sender* or *Off* state, this procedure firstly will generate a set of processors which are suitable to execute the job J_j maybe including also the processor P_i. Afterwards it will assign different probability to each processor of the set. Then the job J_j will be sent to a processor with the highest probability. Otherwise, the job J_j will be executed by the processor P_i or kept in its queue.

```
Transition Procedure:
Each processor P_i runs the following statements
simultaneously at the Transition time (T_n):
 StateInfo = (λ_i(T_n), μ_i(T_n), l_i(T_n))
 Transfer StateInfo to neighbours N_i
 TransitionInfo = (l_N(T_n), Sender_p(T_n), Receiver_p(T_n))
 Call Transition rule

Main Procedure:
If S_i(T_n) == Sender OR S_i(T_n) == Off
  Calculate ET_j^i(t)
  If Receiver_p(t) ≤ 0.1
    pb = Generate a probability in range [0.9, 1]
  Else
    pb = Generate a probability in range [0, 1]
  Add (P_i, pb) to the set P_ET
  For each P_k in N_i
    If S_k(T_n) == Receiver
      Calculate ET_j^k(t)
      If ET_j^k(t) < ET_j^i(t)
        If Receiver_p(t) ≥ 0.9
          pb = Generate a probability in range [0.9, 1]
        Else
          pb = Generate a probability in range [0, 1]
        Add (P_k, pb) to the set P_ET
  Choose a processor as P_dest in P_ET with highest
  probability
  Send the job J_j to the processor P_dest
Else
  If P_i is Idle
    Execute J_j
  Else
    Put J_j into the job queue of P_i
```

4 Simulation and Results

In the simulation, our algorithm is compared with ELISA (Estimated Load Information Scheduling Algorithm). In ELISA [16], each processor exchanges the actual queue length, the actual service rate, and the estimated arrival rate at state exchange times. The queue lengths of its neighbours are estimated at estimation times using information received at state exchange times. The main reason that ELISA has been chosen for the comparison is it considers regular time intervals in order to make load-balancing decisions. Therefore, we can compare two algorithms more precisely by varying intervals. In this paper, four performance metrics are considered, addressed in detail as follow:

1. *Average Response Time (ART)*
2. *Processor Utilization (U)*: The utilization U_i of the processor P_i calculates as mentioned in (13):

$$U_i = \frac{Busy_i}{Busy_i + Idle_i} . \tag{13}$$

Where $Busy_i$ is the amount of time that the processor P_i remains busy and $Idle_i$ is the amount of time that the processor P_i remains idle.

3. *Percent of Executed Jobs (PEJ)*: Percent of jobs that their deadlines do not miss.
4. *Average Off Time (AOT)*: Let us begin by $offTime_i$ that is the amount of time that the processor P_i remains off. Next AOT is defined as presented in (14). It should be mentioned that this metric is unique to LBA_CA.

$$AOT = \frac{1}{M}\sum_{i=1}^{M} offTime_i . \tag{14}$$

4.1 Simulation Model

It is assumed that our simulated Grid system includes 60 heterogeneous computing nodes. Processing powers of nodes are assumed to follow a uniform distribution in range [1, 20]. Communication latency between each processing nodes is chosen from a lognormal distribution with a mean of 0.2 time units and a standard deviation 0.5. It is assumed that time unites are minutes.

In most cases, the number of executed jobs is 10000 unless otherwise stated explicitly. 300 extra jobs are executed at first that are considered as “warm-up jobs”. After warm-up jobs, performance metrics are traced. Jobs arrive at the Grid system according to a Poisson process with rate 1. Service times and deadline times of jobs are assumed to follow an exponential distribution with a mean of 50 and 300 minutes as default values, respectively. Other parameters in LBA_CA are the transition time (T_n) and threshold assumed 10 minutes and 1, as the default values, respectively.

4.1.1 Effect of Number of Jobs

In order to measure the effect of this factor, we increase the number of jobs from 10000 to 50000. At first, as can be seen from Fig. 1, the average response time of

LBA_CA is significantly lower than that of ELISA although it slightly increases due to sharp rises in number of jobs. By using LBA_CA, it clearly shows that jobs leave the system sooner than when ELISA runs due to better load-balancing decisions. The average improvement factor of LBA_CA in terms of the average response time under the effect of number of jobs is 28.7% over ELISA.

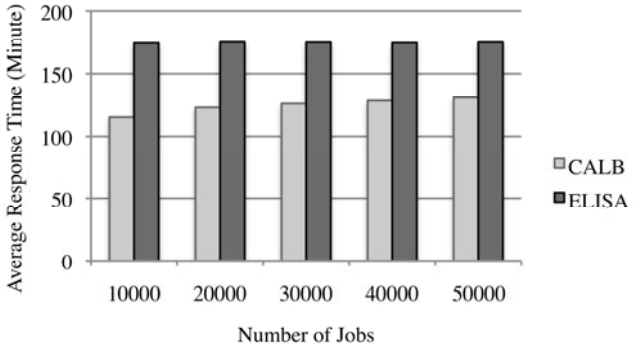

Fig. 1. Effect of number of jobs on average response time of the both algorithms

Afterwards, the amount of difference between the minimum and maximum of processor utilization in the Grid system partly states the quality of load-balancing services provided. As can be easily seen from Fig. 2, the minimum and maximum utilization of processors of LBA_CA are approximately 47.1% closer to the average utilization of the Grid system than those of ELISA. Therefore, this performance metric approves improvements in load-balancing decisions of LBA_CA as well as previous one.

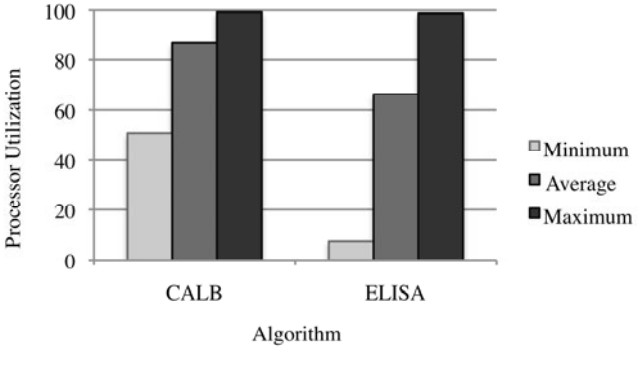

Fig. 2. Comparing processor utilization of the both algorithms

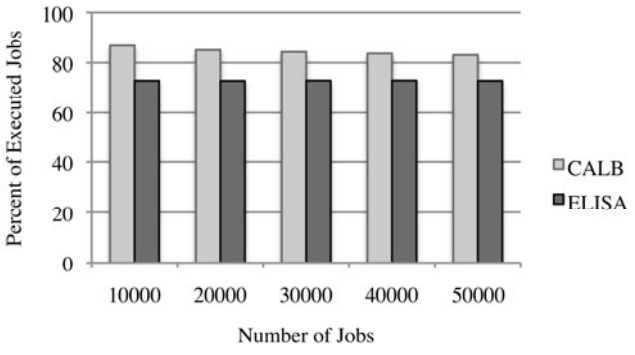

Fig. 3. Effect of number of jobs on percent of executed jobs of the both algorithms

After that Fig. 3 illustrates that the percent of executed jobs for LBA_CA gradually falls nearly from 87% to 83%. Meanwhile, the percent of executed jobs for ELISA remains fairly constant at about 73%. However, LBA_CA shows the average improvement factor 15.8% over ELISA, in terms of the percent of executed. The reason of this improvement is LBA_CA reduces the waiting time of jobs in comparison to ELISA through properly balancing the load across the processors.

Lastly, Fig. 4 wonderfully shows that the average off time has a linear growth with a slope about 0.0002. This metric indicates that some computing nodes usually exist in the Grid system do not take part in load-balancing process due to their inappropriate network situation such as link bandwidths. Therefore, it might be a good idea to turn off such computing nodes in order to conserving energy.

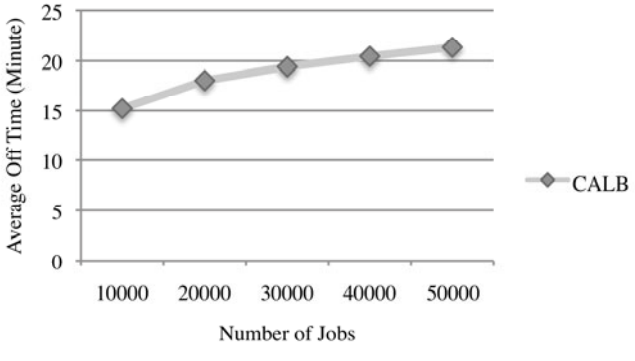

Fig. 4. Effect of number of jobs on average off time of LBA_CA

4.1.2 Effect of Service Time

In this part, it is assumed that the mean of service time varies from 12 to 150 minutes. In order to keep the percent of executed jobs acceptable, the mean of deadline times

for all jobs is fairly raised as the corresponding service time increases. Fig. 5 illustrates that changes in the service time cause dramatic increases in the average response time of the both algorithms. When the mean of service time is under or equal to 50 minutes, LBA_CA shows better results than ELISA. On the other hand, when it is above 50 minutes, ELISA indicates more acceptable results than LBA_CA. However, the average improvement factor of LBA_CA in terms of the average response time under the effect of service time is 13.8% over ELISA.

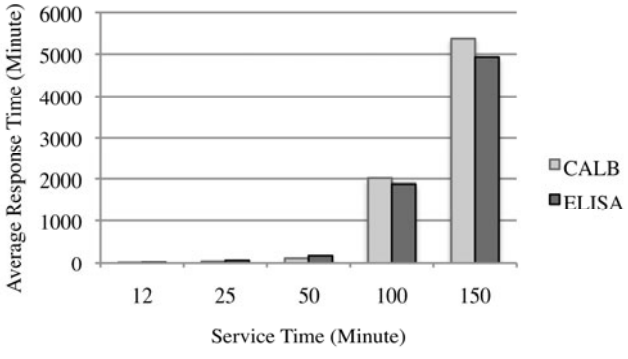

Fig. 5. Effect of service time on average response time of the both algorithms

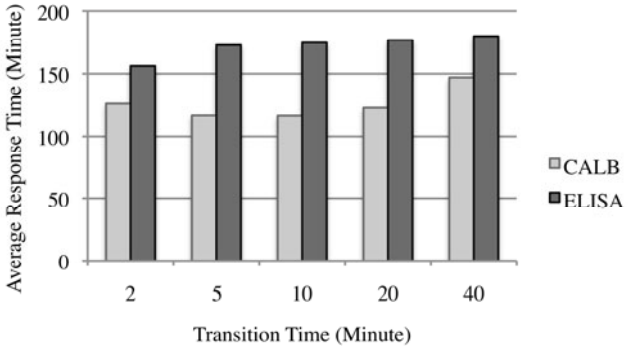

Fig. 6. Effect of transition time on average response time of the both algorithms

4.1.3 Effect of Transition Time

In order to find the most proper transition time (T_n) in terms of performance metrics for running transition rule, we experimented on LBA_CA with different transition times in the wide range of 2 to 40 minutes. It should be mentioned that the state exchange time and estimation time of ELISA are assumed equal to transition time and five times the length of it, respectively. Firstly, in Fig. 6, for transition times under or equal to 10 minutes, LBA_CA certainly shows a slight downward trend of this performance metric. But whenever the transition time rises more than 10 minutes, the average response time of LBA_CA gradually increases. On the other hand, this metric

of ELISA shows an increasing growth. However, LBA_CA indicates the better performance than ELISA due to the average improvement factor of 26.8%. Since LBA_CA reaches its minimum at the transition time of 10 minutes, it is concluded that the most proper time for the transition time is about 10 minutes.

5 Conclusion

Natural dynamic and distributed properties of CA convince us to use it in our proposed local load balancing algorithm and make LBA_CA partly practical for each cluster of computational Grid systems. As mentioned before, each computing node assumed as a cell of CA. Each cell of our proposed CA can be in four state including *Sender*, *Balanced*, *Receiver*, and *Off*. We address resource heterogeneity and communication overheads in LBA_CA through taking several parameters into account such as processing power of computing nodes and communication latency. Moreover, since energy conservation gets a high priority in each aspect of our lives; LBA_CA puts that node into Off state under special circumstances.

LBA_CA attempts to improve the average response time of jobs, although LBA_CA is evaluated in terms of some other performance metrics including processor utilization, the percent of executed jobs, and the average off time. In order to experiment LBA_CA, several items such as number of jobs, service time and transition time are varied in wide ranges of values and it is compared with ELISA. As addressed in depth, LBA_CA plainly performs far better than ELISA in all performance metrics measured by the average improvement factor between 10% and 47%.

Consequently, since LBA_CA gathers up-to-date state information in each transition time, communication overheads may partly increase, although LBA_CA takes account of communication latency for load-balancing decisions. Therefore, it might be a good idea to use estimation methods in some intervals of time instead of providing accurate information. However, it seems that CA is a proper instrument for designing load balancing algorithms and it can be applied to design more efficient load balancing algorithms in future.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
2. Foster, I.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In: Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L. (eds.) Euro-Par 2001. LNCS, vol. 2150, pp. 1–4. Springer, Heidelberg (2001)
3. Subrata, R., Zomaya, A.Y., Landfeldt, B.: Game-Theoretic Approach for Load Balancing in Computational Grids. IEEE Transactions on Parallel and Distributed Systems 19(1), 66–76 (2008)
4. Lu, K., Zomaya, A.Y.: A Hybrid Policy for Job Scheduling and Load Balancing in Heterogeneous Computational Grids. In: 6th International Symposium on Parallel and Distributed Computing, p. 19. IEEE Computer Society, Washington, D.C (2007)
5. Shivaratri, N., Krueger, P., Singhal, M.: Load Distributing for Locally Distributed Systems. Computer 25(12), 33–44 (1992)

6. Lu, K., Subrata, R., Zomaya, A.Y.: Towards Decentralized Load Balancing in a Computational Grid Environment. In: Chung, Y.-C., Moreira, J.E. (eds.) GPC 2006. LNCS, vol. 3947, pp. 466–477. Springer, Heidelberg (2006)
7. Penmatsa, S., Chronopoulos, A.T.: Game-theoretic static load balancing for distributed systems. Journal of Parallel and Distributed Computing (2010)
8. Nasir, H.J.A., Mahamud, K.R.K., Din, A.M.: Load Balancing Using Enhanced Ant Algorithm in Grid Computing. In: 2nd International Conference on Computational Intelligence, Modelling and Simulation, pp. 160–165. IEEE Computer Society Press, Washington, D.C (2010)
9. Zheng, Q., Tham, C.K., Veeravalli, B.: Dynamic Load Balancing and Pricing in Grid Computing with Communication Delay. Journal of Grid computing 6(3), 239–253 (2008)
10. Shah, R., Veeravalli, B., Misra, M.: On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Computational Grid Environments. IEEE Transactions on Parallel and Distributed Systems 18(12), 1675–1686 (2007)
11. Yan, K.Q., Wang, S.S., Wang, S.C., Chang, C.P.: Towards a hybrid load balancing policy in grid computing system. Journal Expert Systems with Applications 36(10), 12054–12064 (2009)
12. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for Your Mathematical Plays, vol. 2. Academic Press, New York (1982)
13. Gramb, T., Bornholdt, S., Grob, M., Mitchell, M., Pellizzari, T.: Computation in Cellular Automata: A Selected Review. In: Mitchell, M. (ed.) Non-Standard Computation: Molecular Computation - Cellular Automata - Evolutionary Algorithms - Quantum Computers, pp. 95–140. Wiley-VCH Verlag GmbH & Co, Weinheim (1998)
14. Swiecicka, A., Seredynski, F., Zomaya, A.Y.: Multiprocessor Scheduling and Rescheduling with Use of Cellular Automata and Artificial Immune System Support. IEEE Transactions on Parallel and Distributed Systems 17(3), 253–262 (2006)
15. Kari, J.: Theory of cellular automata: A survey. Theoretical Computer Science 334(1-3), 3–33 (2005)
16. Anand, L., Ghose, D., Mani, V.: ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing Systems. Computers & Mathematics with Applications 37(8), 57–85 (1999)