International Journal of Computational Intelligence Techniques, ISSN: 0976–0466 & E-ISSN: 0976–0474
Volume 1, Issue 1, 2010, PP-20-26

A novel load balancing algorithm for computational grid

Saravanakumar E. and Gomathy Prathima
Department of CSE, Adhiyamaan College of Engineering, Hosur, India
saraninfo@gmail.com, gomathy.prathima@gmail.com

Abstract-The Grid computing environment is a cooperation of distributed computer systems where user jobs can be executed on either local or remote computer. Many problems exist in grid environment. Not only the computational nodes are heterogeneous but also the underlying networks connecting them are heterogeneous. The network bandwidth varies and the network topology among resources is also not fixed. Thus with this multitude of heterogeneous resources, a proper scheduling and efficient load balancing across the Grid is required for improving performance of the system. The load balancing is done by migrating jobs to the buddy processors, a set of processors to which a processor is directly connected. An algorithm, Load Balancing on Arrival (LBA) is proposed for small-scale (intraGrid) systems. It is efficient in minimizing the response time for small-scale grid environment. When a job arrives LBA computes system parameters and expected finish time on buddy processors and the job is migrated immediately. This algorithm estimates system parameters such as job arrival rate, CPU processing rate and load on each processor to make migration decision. This algorithm also considers job transfer cost, resource heterogeneity and network heterogeneity while making migration decision.
Keywords- Grid systems, load balancing, average response time, migration

Introduction

The Grid [6] is emerging as a wide-scale distributed computing infrastructure that promises to support resource sharing and coordinated problem solving in dynamic multi-institutional Virtual Organizations [5]. The idea is similar to the former metacomputing [16] where the focus was limited to computation resources, whereas Grid computing takes a broader approach. The computational Grid is the cooperation of distributed computer systems where user jobs can be executed on either local or remote computer systems. With its multitude of heterogeneous resources, a proper scheduling and efficient load balancing across the Grid is required for improving the performance of the system. A widely used performance metric is the Average Response Time of tasks. The response time of a task is the time elapsed between its initiation and its completion. Minimizing the average response time is often the goal of load balancing. The system load is a measure of the amount of work that a computer system performs. If load at some computers are typically heavier than at others, or if some processors execute tasks more slowly than others, they will become heavily loaded. The load balancing aims to have all processors equally heavy workloads over the long term. In general, any load-balancing algorithm consists of two basic policies—a transfer policy and a location policy. The transfer policy decides if there is a need to initiate load balancing across the system. By using workload information, it determines when a node becomes eligible to act as a sender (transfer a job to another node) or as a receiver (retrieve a job from another node). The location policy determines a suitably underloaded processor. In other words, it locates complementary nodes to/from which a node can send/receive workload to improve the overall system performance. Location-based policies can be broadly classified as sender initiated, receiver initiated, or symmetrically initiated [4, 7, 12,19]. Based on the information that can be used, load-balancing algorithms are classified as static, dynamic, or adaptive [8, 12, 14, 16]. In a static algorithm, the scheduling is carried out according to a predetermined policy. The state of the system at the time of the scheduling is not taken into consideration. On the other hand, a dynamic algorithm adapts its decision to the state of the system. Adaptive algorithms are a special type of dynamic algorithms where the parameters of the algorithm and/or the scheduling policy itself is changed based on the global state of the system According to another classification, based on the degree of centralization, load-scheduling algorithms

could be classified as centralized or decentralized [12, 16]. In a centralized system, only a single processor does the load scheduling. Such algorithms are bound to be less reliable than decentralized algorithms, where many, if not all, processors do load scheduling in the system. Load balancing involves assigning to each processor work proportional to its performance, thereby minimizing the response time of a job. Normally load balancing is done by migrating the job to buddy processors. A set of processors to which a processor is directly connected constitutes its buddy set. The aim of this paper is to present a load-balancing algorithm adapted to the heterogeneous Grid computing environment. This paper attempt to propose an adaptive decentralized sender-initiated load-balancing algorithm for computational Grid environments.

A. Related Works

Numerous scheduling algorithms have been proposed for parallel and distributed systems, as well as for Grid computing environment. For a dynamic load-balancing algorithm, it is unacceptable to frequently exchange state information because of the high communication overheads. Anand et al. [2] proposed an estimated load information scheduling algorithm (ELISA) and Perfect Information Algorithm (PIA). In PIA when a job arrives, a processor computes the job's finish time on all buddy processors using exact information about the current load of a buddy processor, its arrival rate and service rate. The source processor selects a buddy processor with the minimum finish time and immediately migrate a job on that buddy processor if it can finish the job earlier than this processor. Arora et al. [3] proposed a decentralized load-balancing algorithm for a Grid environment. Although this work attempts to include the communication latency between two nodes during the triggering process on their model, it did not consider the actual cost for a job transfer. The approach proposed takes the job migration cost into account for the load-balancing decision. In [9-11], a sender processor collects status information about neighboring processors by communicating with them at every load-balancing instant. This can lead to frequent message transfers. The status exchange at each load-balancing instant can lead to large communication overhead. The proposed algorithm balance the load by considering the job migration cost, which is primarily influenced by the available bandwidth between the sender and receiver nodes.

B. Contribution

This paper proposes dynamic, adaptive, and decentralized load balancing algorithm for computational Grid environments. The Load Balancing on Arrival (LBA) can be shown to be effective in a small-scale Grid (intraGrid) environment. One of the key strengths of the algorithm is in estimating the system parameters and in proactive job migration. Proposed algorithm considers the job migration cost, which is primarily influenced by the available bandwidth between the sender and receiver nodes, when making a decision for load balancing. Grid infrastructures are dynamic in nature in the sense of resource availability and, hence, a changing network topology. Resource heterogeneity and network heterogeneity also exists in the Grid environment. These facts have also been considered into account by generating a random topology with nodes of varying capacities and varying bandwidth across the links connecting them.

System Model and Problem Definition

Grid system consists of M heterogeneous processors, P1, P2, …, PM, connected via communication channels assuming an arbitrary topology (Fig. 1).

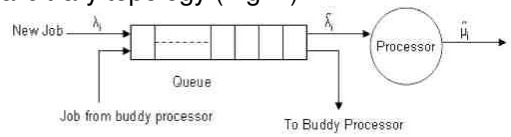

Fig. 1- System model

We assume that each processor has an infinite capacity buffer to store jobs waiting for execution. This assumption eliminates the possibility of dropping a job due to unavailability of buffer space. The jobs are assumed to arrive randomly at the processors, the interarrival time being exponentially distributed with average 1/λi. The jobs are assumed to require service time that is exponentially distributed with mean 1/μi. Each processor is modeled as anM/M/1 Markov chain, with the number of jobs queued up for processing at each processor representing the state of the

system. Job size is assumed to have a normal distribution with a given mean and variance. This job size includes both the program and data sizes. Since in a Grid environment, the network topology is varying, the proposed model captures this constraint as well by considering an arbitrary topology. The data transfer rate is not fixed and varies from link to link. The processors that are directly connected to a processor constitute its buddy set. Here also it is assumed that each processor has knowledge about its buddy processors and the communication latency between them, and load balancing is carried out within buddy sets only. It may be noted that two neighboring buddy sets may have a few processors common to each set. Some key performance metrics of interest considered in this paper are discussed below

A. Performance Metrics

Three performance metrics of relevance at three different levels:

At the job level, we consider the ART of the jobs processed in the system as the performance metric. If N jobs are processed by the system, then

$$ART = \frac{1}{N} \sum_{i=1}^{M} (Finish_i - Arrival_i)$$

where Arrivali is the time at which the ith job arrives and Finishi is the time at which it leaves the system. The delay due to the job transfer, waiting time in the queue, and processing time together constitute the response time. At the system level, we consider the total execution time as the performance metric to measure the algorithm's efficiency. It indicates the time at which all N jobs get executed. At the processor level, we consider resource utilization as the performance metric. It is the ratio between the processor's busy time to the sum of the processor's busy and idle time:

$$U_i = \frac{Busy_i}{(Busy_i + Idle_i)}$$

where Busyi indicates the amount of time Pi remains busy, and Idlei indicates the amount of time Pi remains idle during the total execution time of N jobs.

Thus the objective is to design efficient load-balancing algorithms to minimize the ART of the jobs for computational Grid environments. This algorithm will affect load balancing by careful estimation of the job arrival rates, CPU processing rates, and loads on the processor. Further, we take into account the resource heterogeneity, network heterogeneity, and job migration cost before a load-balancing decision.

Design of Load Balancing Algorithm

In a computational Grid, as resources are geographically distributed and located at different sites, the job transfer time from one site to another site is a very significant factor for load balancing. Further, the communication latency is very large for the WAN through which Grid resources are normally connected. Moreover, due to network heterogeneity, the network bandwidth varies from one link to another. Hence the job transfer cost cannot be ignored when making a job migration decision. And since the resources are heterogeneous, we need jobs have to be assigned to processors according to its performance. Both the algorithms consider these facts. The process of parameter estimation and the way in which load balancing is carried out is described below.

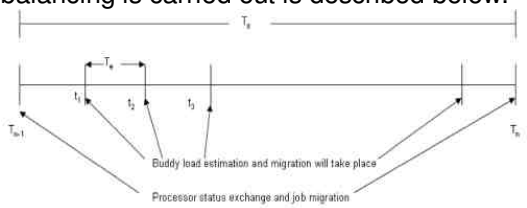

Fig. 2- Estimation and Status Exchange intervals

At each periodic interval of time Ts, called the status exchange interval, each Pi in the system calculates its status parameters, which are the estimated arrival rate, service rate, and load on the processor. Each Pi in the system exchanges its status information with the processors in its buddy set. The instant at which this information exchange takes place is called a status exchange instant. In Fig. 2, Tn-1 and Tn represent the status exchange instant. Each Pi calculates its status information at status exchange instant Tn-1. Each status exchange period is further divided into equal subintervals called estimation interval Te. These points are known as estimation instants. In Figure .2, t1, t2, . . . , t m_1 represent the estimation instants. Each Pi calculates the estimated load on its buddy processor Pk. The status exchange instants and the estimation

ISSN: 0976–0466 & E-ISSN: 0976–0474, Volume 1, Issue 1, 2010

instants together constitute the transfer instants. The decision to transfer jobs and actual transfer of jobs are done at transfer instants.

A Load Balancing on Arrival

LBA, which balances load by transferring a job on its arrival epoch rather than waiting for the next transfer instant. This is clearly a faster reaction to respond to higher arrival rates on smaller Grids. In the LBA algorithm, instead of estimating the expected finish time of a job at every estimation time period Te, it will be calculated on each arrival of a job to a processor. Here, estimating the finish time of a job is an aperiodic event, and job migration will now happen aperiodically. Therefore, when the load is not distributed evenly across all processors, a job will be migrated to lightly loaded processors much faster in the LBA.

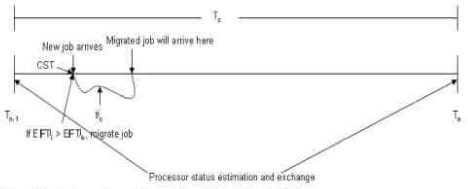

Fig. 3. Job migration decision in LBA.

This approach is shown in Figure. 3. In this approach each processor Pi calculates its status parameters, which are the estimated arrival rate, service rate, and load at every status exchange period Ts. This information gets exchanged to every buddy processor in the buddy set. On every job arrival, processor Pi will calculate the expected finish time of job j on buddy processor Pk by estimating the load on Pk at time CST + tcj (where tcj is the communication time for job j from Pi to Pk and CST is the Current System Time). For this estimation, Pi will calculate the expected number of arrivals and departures for buddy processor Pk for time period t = CST + tcj – Tn-1 (Here Tn-1 is the last status exchange). If any buddy processor Pk can finish the execution of this job before processor Pi, then that job will be migrated immediately to Pk.

Existing Reference Algorithms

There are two existing reference algorithms that are relevant to be discussed as reference algorithms.

A. Perfect Information Algorithm (PIA)

In PIA, each processor has perfect information about the state (in terms of load) of every other processor in its buddy set. This algorithm also uses perfect information about the arrival rate and service rate. When a job arrives, a processor computes the job's finish time on all buddy processors using exact information about the current load of a buddy processor, its arrival rate and service rate. The source processor selects a buddy processor with the minimum finish time and immediately migrate a job on that buddy processor if it can finish the job earlier than this processor. Maintaining up to-date information about all buddy processors require plenty of message transmission. But it is assumed that at the transfer epochs, each processor has perfect information about the state of every other processor in its buddy set. This algorithm can be summarized in the following steps

Main Algorithm

At the transfer epoch, for each processor:
1. Communicate status (queue length) to all processors in the buddy set;
2. Call transfer.

Procedure Transfer

1. Find average queue length of the processors in the buddy set.
2. If queue length of a processor is greater than the average queue length (computed in 1) then:

a) Construct the active set as follows: if a processor in the buddy set has a queue length less than the average queue length, include the processor in the active set;

b) Compute the probability of transferring from the processor (source) to each processor(destination) in the active set such that the source processor load in excess of average queue length is distributed among processors of active set.

3. Transfer the jobs as per the probabilities computed in 2(b). PIA basically provides a lower bound for the LBA algorithm.

B. ELISA

The basic idea behind ELISA is that at periodic intervals of time Ts, called the status exchange interval, the processors in the system exchange their status information, which consists of the queue

length at the instant of information exchange and an estimate of the arrival rate. The instants at which this information exchange takes place is called a status exchange epoch. Each status exchange interval is further divided into equal subintervals called estimation intervals Te. Obviously, Te < Ts. The points of division are called estimation epochs. Further a neighbourhood is defined, termed as a buddy set, for each processor. At the estimation epochs, every processor estimates the load in the processors belonging to its buddy set. All these epochs are actually small intervals in time, but as they are very small compared to Ts and Te, referred to as time instants or epochs. At the transfer epochs, rescheduling of jobs is carried out. Thus, the decision to transfer jobs is taken and the actual transfer of jobs is done at the transfer epochs. Processors in the buddy set, whose estimated queue length is less than the estimated average queue length by more than a threshold θ, form the active set. Now the processor under consideration transfers jobs to the processors in the active set until its queue length is not greater than θ more than the estimated average queue length. Each processor estimates the job arrival rate by considering the number of arrivals in a certain fixed interval of time (called a window) just prior to the instant at which estimation is done.

Table 1- Information Used by Algorithms

Algorithm	Arrival Rate	Service Rate	System State
ELISA	Estimated Information	Perfect Information	Estimated Information
LBA	Estimated Information	Estimated Information	Estimated Information
PIA	Perfect Information	Perfect Information	Perfect Information

Performance Evaluation and Discussion

Table 1 summarizes the information that the algorithms use for the scheduling of jobs. the performance of our proposed LBA algorithm is evaluated with ELISA and PIA

A. Random Arrival and Service Rates

The performance of our LBA algorithm is quantified for real-life situations wherein arrival rates and service rates are completely random. ART for LBA and that for ELISA, that is, both algorithms exhibit an increasing tendency as we increase the arrival and service rates. Both algorithms take almost the same amount of time for the execution of N jobs.

B. Effect of Status Exchange Period

ELISA is highly sensitive to the magnitude of the status exchange period Ts. That is, if we set the value of the status exchange period to be high, then its performance degrades. For LBA, increasing the value of Ts also increases ART, but its performance is much better than that of ELISA. By increasing the value of Ts, there is very high increase in ART for ELISA than for LBA. For PIA, there is no change in ART as it uses perfect information about the system state at each job migration decision. Therefore, for the LBA algorithm, by setting the value of the status exchange period to be large, we can decrease the number of status exchange messages, and communication overheads can be kept at a low value.

C. Effect of uneven load distribution

One of the major advantages of the LBA approach is that it attempts to balance the load on each processor "as soon as possible." Whenever a job arrives at a processor, that processor will determine whether any of its buddy set members can execute the job earlier than itself. If it finds such a member, then the job will be migrated to that processor. In this way, the load will be balanced as soon as possible. However in ELISA, a job has to wait for the next transfer instant before migrating to a lightly loaded processor.

D. Effect of migration limit

One of the important parameters for LBA is the migration limit (denoted as η), that is, maximum number of hops that a job is allowed to migrate before execution. Obviously, this decision depends on the network topology considered. By restricting the value of η to a finite value, we can reduce the job migration cost by reducing the total number of job migrations.

E. Effect of job size

For the LBA algorithm, the job migration cost is also one of the factors for load balancing across its buddy processors. Indeed, we can expect that it should give better performance when we increase the job size. For a larger job size, the performance of LBA is better

ISSN: 0976–0466 & E-ISSN: 0976–0474, Volume 1, Issue 1, 2010

than that of ELISA. This is due to the fact that as the job size increases, the migration cost is expected to increase, which prevents migration in LBA.

Conclusion

This paper presents, decentralized, scalable, adaptive, and distributed algorithm for load balancing across resources for data-intensive computations on Grid environments. The objective is to minimize ART and the total execution time for jobs that arrive at a Grid system for processing. Several constraints such as communication delays due to the underlying network, processing delays at the processors, and an arbitrary topology for a Grid system are explicitly considered in the problem formulation. The proposed algorithm is adaptive in the sense that it estimate different types of strongly influencing system parameters such as the job arrival rate, processing rate, and load on the processor and use this information for estimating the finish time of job on a buddy processor. The LBA algorithm performs load balancing on each job arrival by estimating the expected finish time on a neighboring processor instead of waiting for the next transfer instant (unlike ELISA).

Even though the main objective is to propose load-balancing algorithms using parameter estimation for heterogeneous Grid environments, this work can be extended by providing fault tolerance into the system, as fault tolerance is a very important characteristic for any distributed systems.

References

[1] Ruchir Shah, Bhardwaj Veeravalli and Manoj Mishra "On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Computational Environments", IEEE Trans. Parallel and Distributed Systems, vol 18, no 12, December 2007.

[2] L. Anand, D. Ghose, and V. Mani, "ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing Systems," Int'l J. Computers and Math. with Applications, vol. 37, no. 8, pp. 57-85, Apr. 1999.

[3] M. Arora, S.K. Das, and R. Biswas, "A De-Centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments," Proc. Int'l Conf. Parallel Processing Workshops (ICPPW '02), pp. 499-505, 2002.

[4] Y. Feng, D. Li, H. Wu, and Y. Zhang, "A Dynamic Load Balancing Algorithm Based on Distributed Database System," Proc. Fourth Int'l Conf. High-Performance Computing in the Asia-Pacific Region, pp. 949-952, May 2000.

[5] Ian Foster, Carl Kesselman, Steven Tuecke "The Anatomy of the Grid Enabling Scalable Virtual Organizations" Int'l J. High Performance Computing Applications, vol. 15, no. 3, pp. 200-222, 2001. [6] I. Foster and C. Kesselman, "The Grid: Blueprint for a Future Computing Infrastructure," Morgan Kaufmann, 1999.

[6] H. Lin and C. Raghavendra, "A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC)," IEEE Trans. Software Eng., vol. 18, no. 2, pp. 148-158, Feb. 1992.

[7] G. Manimaran and C. Siva Ram Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real¬Time Systems," IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 3, pp. 312-319, Mar. 1998.

[8] Y. Murata, H. Takizawa, T. Inaba, and H. Kobayashi, "A Distributed and Cooperative Load Balancing Mechanism for Large-Scale P2P Systems," Proc. Int'l Symp. Applications and Internet (SAINT '06) Workshops, pp. 126-129, Jan. 2006.

[9] L. Oliker, R. Biswas, H. Shan, and W. Smith, "Job Scheduling in Heterogeneous Grid Environment," Technical Report LBNL-54906, Lawrence Berkeley Nat'l Laboratory, 2004

[10] H. Shan, L. Oliker, and R. Biswas, "Job Superscheduler Architecture and Performance in Computational Grid Environments," Proc. ACM/IEEE Conf. Supercomputing, Nov. 2003.

[11] N. Shivaratri, P. Krueger, and M. Singhal, “Load Distributing for Locally Distributed Systems,” Computer, vol. 25, no. 12, pp. 33-44, Dec. 1992.
[12] L. Smarr and C.E. Catlett, “Metacomputing,” Comm. ACM, vol. 35, no. 6, pp. 44-52, June 1992.
[13] J. Watts and S. Taylor, “A Practical Approach to Dynamic Load Balancing,” IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 3, pp. 235-248, Mar. 1998.
[14] M. Willebeek-LeMair and A. Reeves, “Strategies for Dynamic Load Balancing on Highly Parallel Computers,” IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 4, pp. 979-993, Sept.1993.
[15] M.J. Zaki and W.L.S. Parthasarathy, “Customized Dynamic Load Balancing for a Network of Workstations,” J. Parallel and Distributed Computing, vol. 43, no. 2, pp.156-162, June 1997.

ISSN: 0976–0466 & E-ISSN: 0976–0474, Volume 1, Issue 1, 2010