On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Computational Grid Environments

Ruchir Shah, Bhardwaj Veeravalli, *Senior Member*, *IEEE*, and Manoj Misra, *Member*, *IEEE*

Abstract—In this paper, we address several issues that are imperative to Grid environments such as handling resource heterogeneity and sharing, communication latency, job migration from one site to other, and load balancing. We address these issues by proposing two job migration algorithms, which are MELISA (Modified ELISA) and LBA (Load Balancing on Arrival). The algorithms differ in the way load balancing is carried out and is shown to be efficient in minimizing the response time on large and small-scale heterogeneous Grid environments, respectively. MELISA, which is applicable to large-scale systems (that is, interGrid [1]), is a modified version of ELISA [2] in which we consider the job migration cost, resource heterogeneity, and network heterogeneity when load balancing is considered. The LBA algorithm, which is applicable to small-scale systems (that is, intraGrid [1]), performs load balancing by estimating the expected finish time of a job on buddy processors on each job arrival. Both algorithms estimate system parameters such as the job arrival rate, CPU processing rate, and load on the processor and balance the load by migrating jobs to buddy processors by taking into account the job transfer cost, resource heterogeneity, and network heterogeneity. We quantify the performance of our algorithms using several influencing parameters such as the job size, data transfer rate, status exchange period, and migration limit, and we discuss the implications of the performance and choice of our approaches.

Index Terms—Grid systems, load balancing, average response time, communication delays, migration.

1 Introduction

The Grid [1] 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 [3]. The idea is similar to the former metacomputing [4] where the focus was limited to computation resources, whereas Grid computing takes a broader approach. On one hand, Grid computing provides the user with access to locally unavailable resource types. On the other hand, there is the expectation that a large number of resources are available. A computational Grid is the cooperation of distributed computer systems where user jobs can be executed on either local or remote computer systems. With the Grid becoming a viable high-performance alternative to the traditional supercomputing environment, various aspects of effective Grid resource utilization are gaining significance. 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.

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 [5], [6], [7], [8]. Further, while balancing the load, certain types of information such as the number of jobs waiting in queue, job arrival rate, CPU processing rate, and so forth at each processor, as well as at neighboring processors, may be exchanged among the processors for improving the overall performance. Based on the information that can be used, load-balancing algorithms are classified as static, dynamic, or adaptive [7], [9], [10], [11]. 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

- *R. Shah and M. Misra are with the Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee, India 247667. E-mail: r4_ruchir@yahoo.co.in, manojfec@iitr.ernet.in.*
- *B. Veeravalli is with the Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576. E-mail: elebv@nus.edu.sg.*

Manuscript received 29 Mar. 2006; revised 27 Nov. 2006; accepted 23 Jan. 2007; published online 10 May 2007.
Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0077-0306.
Digital Object Identifier no. 10.1109/TPDS.2007.1115.

classified as centralized or decentralized [7], [11]. In a centralized system, the load scheduling is done only by a single processor. Such algorithms are bound to be less reliable than decentralized algorithms, where load scheduling is done by many, if not all, processors in the system. However, decentralized algorithms have the problem of communication overheads incurred by frequent information exchange between processors.

Load balancing involves assigning to each processor work proportional to its performance, thereby minimizing the response time of a job. However, there are a wide variety of issues that need to be considered for a heterogeneous Grid environment. For example, the capacities (in terms of processor speed) of the machines differ because of processor heterogeneity. Also, their usable capacities vary from moment to moment according to the load imposed upon them. Further, in Grid computing, as resources are distributed in multiple domains in the Internet, not only the computational nodes but also the underlying network connecting them are heterogeneous. The heterogeneity results in different capabilities for job processing and data access. For instance, a typical bandwidth in a local area network (LAN) vary from $10\ Mbps$ to $1\ Gbps$, whereas it is a few $kbps$ to $Mbps$ for a wide area network (WAN). Also, the network bandwidth across resources varies from link to link for large-scale Grid environments. Further, the network topology among resources is also not fixed due to dynamic nature of the Grid. Many load-balancing algorithms [2], [12], [13] have been proposed for traditional cluster computing and distributed systems. However, little work has been done to cater the following unique characteristics for the Grid computing environment: heterogeneous resources, considerable communication delay, and dynamic network topology. The present work is targeted to the Grid model where heterogeneous resources are connected through an arbitrary topology, and the network bandwidth also varies from link to link. This aspect precludes the application of traditional load-balancing algorithms to Grid computing. The aim of this paper is to present load-balancing algorithms adapted to the heterogeneous Grid computing environment. In this paper, we attempt to formulate two adaptive decentralized sender-initiated load-balancing algorithms for computational Grid environments. Below, we highlight some key contributions from the literature that are relevant to the context of our paper.

1.1 Related Works

Numerous researchers have proposed scheduling algorithms [2], [12], [13] for parallel and distributed systems, as well as for Grid computing environment [14], [15], [16], [17], [18], [19], [20], [21], [22]. For a dynamic load-balancing algorithm, it is unacceptable to frequently exchange state information because of the high communication overheads. In order to reduce the communication overheads, Martin et al. [23] studied the effects of communication latency, overhead, and bandwidth in a cluster architecture to observe the impact on application performance. Anand et al. [2] proposed an estimated load information scheduling algorithm (ELISA), and Mitzenmacher [24] analyzed the usefulness of the extent to which old information can be used to estimate the state of the system. Arora et al. [21] 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. Our approach takes the job migration cost into account for the load-balancing decision. In [15], [16], and [18], 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. For a large-scale Grid environment where communication latency is very large, the status exchange at each load-balancing instant can lead to large communication overhead. In our approach, the problem of frequent exchange of information is alleviated by estimating the load, based on the system state information received at sufficiently large intervals of time.

We have proposed algorithms for a Grid environment that are based on the estimation approach as carried out in the design of ELISA [2]. In ELISA, load balancing is carried out based on queue lengths. Whenever there is a difference in queue length, jobs will be migrated to the lightly loaded processor, ignoring the job migration cost. This cost becomes an important factor when the communication latency is very large such as for a Grid environment and/or the job size is large. Both of our algorithms balance the load by considering the job migration cost, which is primarily influenced by the available bandwidth between the sender and receiver nodes.

1.2 Our Contributions

Our contributions in this paper are multifold. We have proposed two dynamic, adaptive, and decentralized load-balancing algorithms for computational Grid environments that are shown to be applicable in balancing loads depending on the size of the underlying Grid infrastructure. Thus, for smaller size Grids, one of our algorithms, Load Balancing on Arrival (LBA), is shown to be effective, whereas for large-scale Grid systems, the Modified ELISA (MELISA) is shown to have better control in balancing the loads. One of the key strengths of our algorithm is in estimating the system parameters and in proactive job migration. For large-scale Grid environments, resources are geographically distributed, and the communication latency between them is also very large due to the WAN through which they are usually connected. Therefore, the job migration cost, based on the estimate of the traffic and loading conditions, becomes an imperative factor for load balancing. Our proposed algorithms consider 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. Further, 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. We have also considered these facts into account by generating a random topology with nodes of varying capacities and varying bandwidth across the links connecting them.

Although our earlier version of this work [14] demonstrated the feasibility of our estimation approaches, the scope of the study was limited to consider only the effect of estimation for a large-scale Grid environment, and the only metrics being considered are the average response time (ART) and migration costs. The sensitive parameters such as the arrival and service rates, uneven load distribution

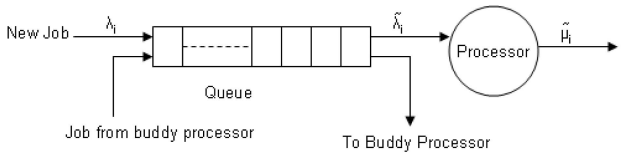


Fig. 1. System model.

scenarios, effects of status exchange information and estimation periods, capturing the migration limits, and utilization are not addressed. In this paper, we present a complete formulation and results for large-scale, as well as for small-scale, Grid environments, including the contributions of [14] and addressing all the abovementioned issues. In all, through this study, we demonstrate the usefulness and effectiveness of the load estimation approach to devise adaptive and dynamic load-balancing strategies for small- and large-scale computational Grid structures.

The paper is organized as follows: In Section 2, we present our Grid system model and problem definition. This section describes the characteristic of a node, job, etc., and also introduces notations that are used throughout the paper. In Section 3, we introduce our two algorithms: MELISA and LBA. In Section 4, we describe the reference algorithms that are used to compare the results of our algorithms. The performance evaluation and discussion of the results are presented in Section 5. Finally, Section 6 concludes the paper.

2 System Model and Problem Definition

Our Grid system consists of M heterogeneous processors, $P_1, P_2, \ldots, P_M$, connected via communication channels assuming an arbitrary topology (Fig. 1). 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 $1/\lambda_i$. The jobs are assumed to require service time that is exponentially distributed with mean $1/\mu_i$. Each processor is modeled as an $M|M|1$ 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, our 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*. We also assume 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. The job arrival rates and service rates are such that for some processor (say, P_i), $\lambda_i > \mu_i$ (that is, P_i is unstable), but the whole system always remains stable, that is, $\sum_{i=1}^{M} \lambda_i < \sum_{i=1}^{M} \mu_i$.

Before presenting the exact problem statement, we first describe the notations and terminology that are used throughout the paper (refer to Table 1).

TABLE 1
List of Notations and Terminology

M	Number of heterogeneous processors $(P_1, P_2, ..., P_M)$
N	Number of jobs to be processed
λ_i	Actual arrival rate for P_i (Poisson distribution)
$1/\mu_i$	Actual mean service time for P_i (Exponential distribution)
CST	Current System Time
T_s	Status exchange period
T_e	Load estimation period
η	Migration Limit
ERT_i^j	Estimated Run Time of job j on P_i
EFT_i^j	Estimated Finish Time of job j on P_i
S_i	Normalized speed measure for P_i
t_c^j	Communication time for job j
$A_i(t)$	Actual number of job arrivals for P_i in time t
$D_i(t)$	Actual number of job departures for P_i in time t
$EA_i(t)$	Expected number of job arrivals for P_i in time t
$ED_i(t)$	Expected number of job departures for P_i in time t
α	Arrival rate estimation factor
β	Service rate estimation factor
$\widetilde{\lambda}_i(T)$	Estimated arrival rate for P_i at time T
$\widetilde{\mu}_i(T)$	Estimated service rate for P_i at time T
$\widetilde{L}_i(T)$	Estimated load on P_i at time T
$\widetilde{L}_{k,i}(T)$	Estimated load on buddy processor P_k calculated by P_i at time T
$Q_i(T)$	Number of jobs waiting in queue for P_i at time T

We will now introduce certain key performance metrics of interest considered in this paper.

2.1 Performance Metrics

In this work, we have considered 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

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

where $Arrival_i$ is the time at which the ith job arrives, and $Finish_i$ 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:

$$U_i = \frac{Busy_i}{(Busy_i + Idle_i)}, \tag{2}$$

where $Busy_i$ indicates the amount of time P_i remains busy, and $Idle_i$ indicates the amount of time P_i remains idle during the total execution time of N jobs.

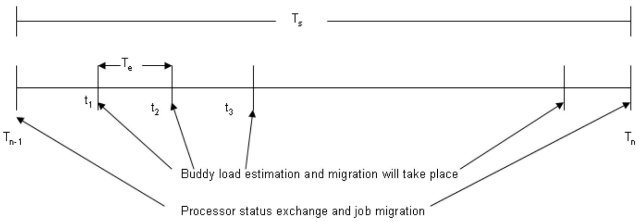


Fig. 2. Estimation and status exchange intervals.

Thus, our objective is to design efficient load-balancing algorithms to minimize the ART of the jobs for computational Grid environments. We propose two different algorithms, LBA and MELISA, for small-scale and large-scale Grid structures, as mentioned briefly in Section 1.2. Our algorithms 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.

3 Design of Load-Balancing Algorithms

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. Due to these reasons, one cannot ignore the job transfer cost when making a job migration decision. Further, when resources are heterogeneous, we need to assign jobs to processors according to its performance. Our both algorithms consider these facts. We shall first describe the process of parameter estimation and the way in which load balancing is carried out in our algorithms in general. This will also be useful in understanding the terminology associated with the notations used.

As shown in Fig. 2, at each periodic interval of time T_s, called the status exchange interval, each P_i in the system calculates its status parameters, which are the estimated arrival rate, service rate, and load on the processor. Each P_i 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, T_{n-1} and T_n represent the status exchange instant. Each P_i calculates its status information at status exchange instant T_{n-1} using the following relationships:

$$\widetilde{\lambda}_i(T_{n-1}) = \alpha * \widetilde{\lambda}_i(T_{n-2}) + (1-\alpha) * (A_i(T_s)/T_s), \quad (3)$$

$$\widetilde{\mu}_i(T_{n-1}) = \beta * \widetilde{\mu}_i(T_{n-2}) + (1-\beta) * (D_i(T_s)/T_s), \quad (4)$$

$$\widetilde{L}_i(T_{n-1}) = Q_i(T_{n-1})/\widetilde{\mu}_i(T_{n-1}). \quad (5)$$

Thus, in the above relationship, by tuning the parameter $\alpha(0 \le \alpha \le 1)$, one can vary the estimate. A value of 0.5 for α would mean that an equal weight has been considered for the current period and the previous estimate of the arrival rate. Similarly, we tune the parameter $\beta(0 \le \beta \le 1)$ for the service rate estimation.

Each status exchange period is further divided into equal subintervals called estimation interval T_e. These points are known as estimation instants. In Fig. 2, $t_1, t_2, \ldots, t_{m-1}$ represent the estimation instants. As each processor balances the load within its buddy set, every processor estimates the load in the processors belonging to its buddy set at each estimation instant. Each P_i calculates the estimated load on its buddy processor P_k using the following equations:

$$\widetilde{L}_{k,i}(T_{n-1}+t_i) = (EA_k(T_e) - ED_k(T_e))/\widetilde{\mu}_k(T_{n-1}) + \widetilde{L}_{k,i}(T_{n-1}+t_{i-1}), \quad (6)$$

where $i = 1, 2, 3, \ldots, m-1$, and $EA_k(T_e) = a$ such that

$$\sum_{x=0}^{a} \frac{e^{(-\widetilde{\lambda}_k(T_{n-1})*T_e)} * (\widetilde{\lambda}_k(T_{n-1}) * T_e)^x}{x!} \cong 1. \quad (7)$$

$ED_k(T_e) = d$[1] such that

$$\sum_{x=0}^{d} \frac{e^{(-\widetilde{\mu}_k(T_{n-1})*T_e)} * (\widetilde{\mu}_k(T_{n-1}) * T_e)^x}{x!} \cong 1. \quad (8)$$

Depending on the accuracy required, the computations of $EA_k(T_e)$ and $ED_k(T_e)$ can be terminated after computing a sufficiently large number of terms in (7) and (8).

The status exchange instants and the estimation instants together constitute the set of transfer instants $(T_{n-1}, t_1, t_2, \ldots, t_{m-1}, T_n)$ in Fig. 2. At the transfer instants, the rescheduling of jobs is carried out. Thus, the decision to transfer jobs and the actual transfer of jobs is done at the transfer instants. By making the interval between status exchange instants large and by restricting the exchange of information to the buddy set, the communication overheads are kept at a minimum.

Here, we would like to mention that the assumption of exponential arrival and service rates can be easily relaxed. In that case, the expressions for the number of job arrivals (7) and job departures (8) on buddy processors would undergo change according to the assumed distribution function, but the rest of the algorithm and decision procedure remain the same. In order to incorporate the dynamic changing environment of the Grid, whenever any new resource joins the Grid system, it should inform to all its buddy nodes about its processing capacity and current load. Therefore, from the next transfer instant (T_s) onward, buddy nodes can consider this newly entered node as a candidate for transferring a job. Similarly, whenever any existing node leaves the system voluntarily, it should inform to all its buddy nodes about it. In case of a node failure, a buddy node will not receive any information from that node during the next status exchange instant (T_s). Therefore, a buddy node can consider that node as not available, and the job will not be transferred to that node. In our Grid system model, we have assumed that the available

1. Note that the number of job departures cannot be greater than the number of job arrivals. That is,

$$ED_k(T_e) \le (EA_k(T_e) + \widetilde{L}_{k,i}(T_{n-1}+t_{i-1}) * \widetilde{\mu}_k(T_{n-1})).$$

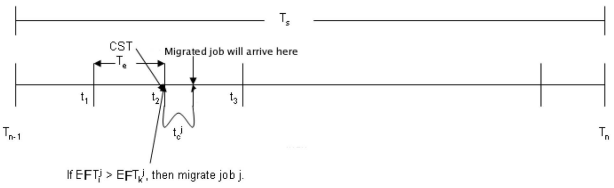


Fig. 3. Job migration decision in MELISA.

network bandwidth remains constant. In the case of a varying network bandwidth, a node should also pass the available network bandwidth during status exchange instant (T_s) along with other system parameters. Buddy nodes should consider this bandwidth as the available bandwidth for next exchange period and should calculate the migration cost using this value.

3.1 Modified ELISA

Although ELISA primarily works on estimates, it is mainly proposed for cluster-based supercomputing systems wherein the communication cost is not very large as resources are connected through a high-bandwidth network. However, for Grid-based supercomputing systems, the transfer delays are significantly high, and network heterogeneity also exists in terms of the varying available network bandwidth contributing to a large communication cost. Thus, the direct applicability of ELISA will yield an inferior performance that is unacceptable for Grid-based systems where heterogeneity exists in terms of resource and network. Later, in our simulation study, we highlight this fact. Hence, we revisit the design of ELISA and introduce the job transfer rate explicitly in a formulation that is more akin for Grids.

In ELISA, at every status exchange time period T_s, each P_i communicates its status (queue length, estimate of the arrival rate) to all its buddy processors. At each estimation instant T_e, every processor calculates the queue length on buddy processors using the estimated arrival rate and exact service rate of a buddy processor. P_i will make a decision of job migration if its queue length is greater than the average queue length in its buddy set.

In the design of MELISA, as shown in Fig. 3, each P_i estimates its arrival rate, service rate, and the load using (3), (4), and (5) at each status exchange instant. At each estimation instant, P_i calculates the load on its all buddy processors using (6), (7), and (8). Based on this calculated buddy load, each processor calculates the average load in its buddy set. P_i will make a decision of job migration if its load is greater than the average load in its buddy set and will try to distribute its load such that load on all buddy processors get finished at almost the same time, taking into account the node's heterogeneity in terms of processor speed. This average buddy load can be calculated using the following relationships.

Let S_i denote the weight of a processor P_i, which is a normalized measure of its speed. Therefore, a value of 2 for S_i means that P_i will take half amount of time to execute a job than the time taken by the reference processor[2] having a value of 1 for S_i. Here, each P_i will calculate the average

MELISA: Main Algorithm

At the status exchange instant, for each processor:

1. Estimate the arrival rate and the service rate using equations (3) and (4). Also determine the load on a processor by dividing its queue length with estimated service rate using (5).
2. Communicate the status defined by a 3-tuple as: <estimate of load, arrival rate, service rate> to all processors in the buddy set.
3. Call TRANSFER.

At the estimation instant, for each processor

1. Estimate the load for each processor in the buddy set.
2. Call TRANSFER.

Fig. 4. Main algorithm for MELISA.

normalized buddy load (NBL_{avg}) using the value of $\widetilde{L}_{k,i}(T)$ and S_i by the following equation:

$$NBL_{avg} = \frac{\sum_{k \epsilon buddyset_i} S_k * \widetilde{L}_{k,i}(T)}{\sum_{k \epsilon buddyset_i} S_k}. \tag{9}$$

NBL_{avg} indicates the average load for a reference processor. P_i is considered as a sender processor if $NBL_{avg} < S_i * \widetilde{L}_i(T)$. Now, P_i will try to transfer its extra load to all receiver processors P_k such that they receive extra load based on their current load $(\widetilde{L}_{k,i}(T))$ and processor weight (S_k). After determining how much load P_i can transfer to P_k, as shown in Fig. 3, P_i will calculate the expected finish time of job j on buddy processor (P_k) by estimating the load on P_k at time $CST + t_c^j$ (where t_c^j is the migration time for job j from P_i to P_k). A job will be migrated to P_k *only if* $EFT_k^j < EFT_i^j$, where

$$EFT_i^j = Q_i(CST)/\widetilde{\mu}_i(T_{n-1}) + ERT_i^j, \tag{10}$$

$$EFT_k^j = max(\ (\widetilde{L}_{k,i}(CST) + (EA_k(t_c^j) - ED_k(t_c^j))/\widetilde{\mu}_k(T_{n-1})), t_c^j) + ERT_k^j. \tag{11}$$

In (11), the first term, which is the maximum of two values—approximate wait time of job j on P_k and job transfer time—indicates the expected starting time of job j on P_k. We assume that these activities can be performed simultaneously. Therefore, a job will be migrated only if its expected finish time on the destination processor is less than the expected finish time on the source processor. In Figs. 4 and 5, the complete workings of MELISA described above are captured.

Procedure TRANSFER by Pi, i=1,2,...,M.

1. Estimate an average normalized buddy load.
2. If load of a processor is greater than avg load (as computed in 1), then
   a) Construct active set as follows: if a processor in the buddy set has load less than avg normalized buddy load, include processor in active set.
   b) Determine how much load can be transferred to buddy processors in active set such that load on all processors get finished at almost same time.
   c) Attempt to migrate the load in excess over avg buddy load to all buddy processors in active set by calculating EFT_k^j on destination processor and migrating job only if $EFT_k^j < EFT_i^j$

Fig. 5. Procedure TRANSFER for MELISA.

2. This could be an abstract processor within the system.

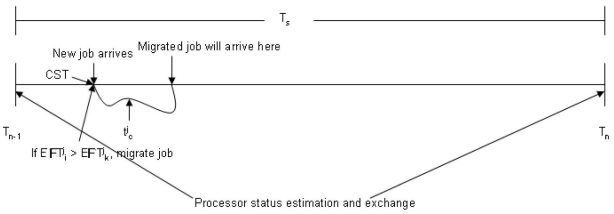


Fig. 6. Job migration decision in LBA.

3.2 Load Balancing on Arrival (LBA)

As stressed in the introduction, the applicability of MELISA is more appropriate to cases wherein the Grid infrastructure is large, and communication delays are significant. Moreover, the load balancing is done only at transfer instants (which can be an estimation instant or a status exchange instant) for ELISA and MELISA, which is acceptable as communication delays are severe. However, this becomes an obvious inherent disadvantage when either of these algorithms is applied to small-scale Grids. That is, with these approaches, jobs need to wait till the next transfer instant for migration, and due to the random arrival rate and service rate at each processor, it is possible that the load does not get distributed evenly across all processors. In this case, there can be large waiting times at highly loaded processors, whereas lightly loaded processors continue to remain idle. Jobs from highly loaded processors will be migrated to a lightly loaded processor after a finite amount of time depending on the values of T_e, T_s, and the distance between the highly loaded processor and lightly loaded processor. This can lead to performance degradation even for moderate values of T_e and T_s. Our simulation results also support this observation. Consequently, there seems to be a need for designing an alternate load-balancing algorithm to take into account such situations.

We design and propose a new algorithm, referred to as 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 T_e, 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 approach than in (M)ELISA.

In this approach (refer to Fig. 6), similar to the MELISA algorithm, each processor P_i calculates its status parameters, which are the estimated arrival rate, service rate, and load at every status exchange period T_s, using (3), (4), and (5). This information gets exchanged to every buddy processor in the buddy set. On every job arrival, processor P_i will calculate the expected finish time of job j on buddy processor P_k by estimating the load on P_k at time $CST + t_c^j$ (where t_c^j is the communication time for job j from P_i to P_k) using (11). For this estimation, P_i will calculate the expected number of arrivals and departures for buddy processor P_k for time period $t = CST + t_c^j - T_{n-1}$.[3] If any buddy processor P_k can finish the execution of this job before processor P_i, then that job will be migrated immediately to P_k. The flowchart for LBA is shown in Fig. 7.

3. Here, T_{n-1} is the last status exchange instant.

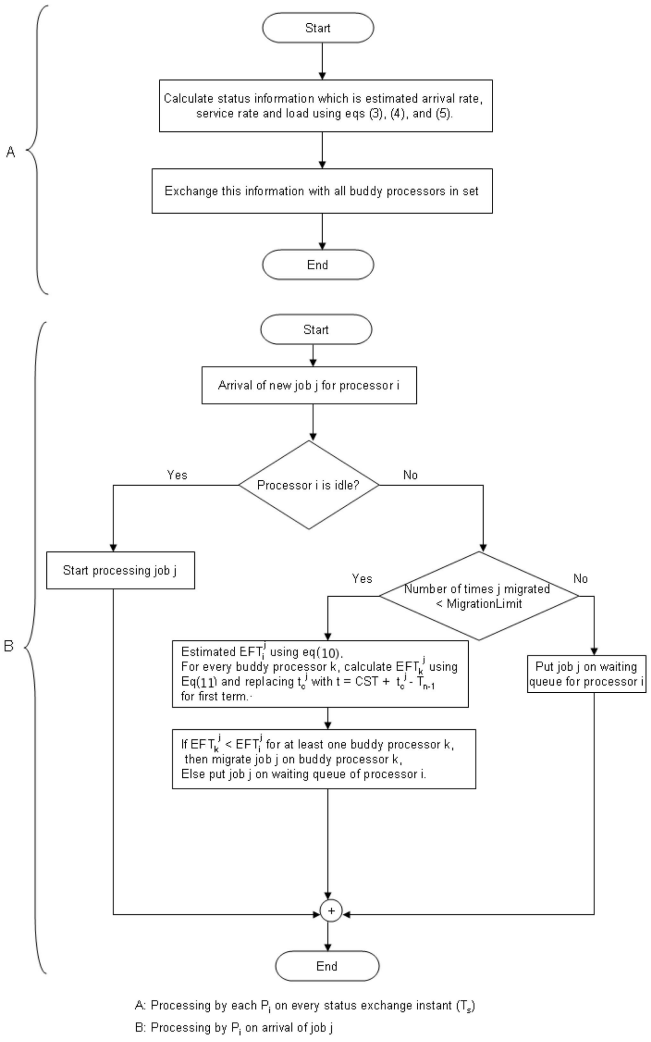


Fig. 7. Flowchart for Load Balancing on Arrival (LBA).

4 Existing Reference Algorithms

We have used two algorithms, which are relevant to our context, as reference algorithms to compare the results of our algorithms.

4.1 Perfect Information Algorithm (PIA)

In PIA [2], 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. Although maintaining up-to-date information about all buddy processors require plenty of message transmission, this algorithm basically provides a lower bound for our LBA algorithm.

4.2 ELISA

In ELISA [2], each processor estimates the queue length of its buddy processor at the estimation instant using information exchanged at the status exchange instant. The

TABLE 2
Information Used by Algorithms

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

information exchanged at the status exchange instant includes the queue length and the estimated arrival rate. This algorithm assumes to have perfect information about the service rate of each processor.

5 Performance Evaluation and Discussions

Here, we present the results of our simulation study and compare the performance of our proposed algorithms with the other algorithms discussed in Section 4. The amount of information that is made available for use at the instant of decision making for the transfer of jobs is expected to have a significant effect on the relative performance of the algorithms. Table 2 summarizes the information that the algorithms use for the scheduling of jobs.

In our simulation model, we have considered heterogeneous processors connected by communication channels assuming an arbitrary topology generated by a graph generator tool. The network bandwidth connecting two nodes is also arbitrary and varies from $0.5\ Mbps$ to $10\ Mbps$. The various parameter values used for the simulation are shown in Table 3. All time units are in seconds, so performance metrics (which are ART and the total execution time) are also measured in seconds. These parameter values are used for all cases unless otherwise stated explicitly.

5.1 ELISA versus MELISA

As we have already mentioned that MELISA is largely suitable for a large-scale grid environment, we conducted

TABLE 3
Parameter Values

Parameter	Value
Mean inter-arrival time	Exponentially distributed in $[1, 4]$
Mean service time	Exponentially distributed in $[1, 4]$
Threshold level	1
N	10000
T_s	20
T_e	4
α	0.5
β	0.5
η	4
Job Size	Normal distribution with μ=5MB and σ=1MB

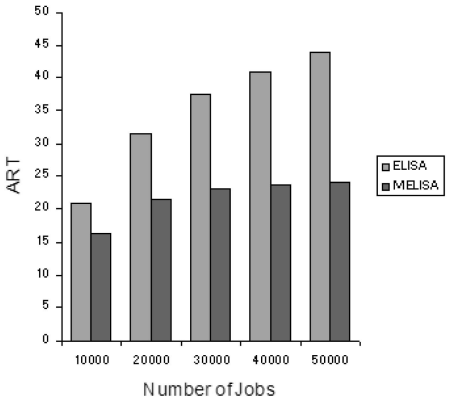


Fig. 8. ART comparison for the heterogeneous case.

experiments for the large-scale Grid with 128 nodes. The main difference between ELISA and MELISA is that MELISA takes into account the job migration cost and resource heterogeneity when balancing the load across buddy processors. The following three sections compare the results of MELISA with ELISA for a heterogeneous environment and for different job migration costs.

5.1.1 Performance for the Heterogeneous Case

For the heterogeneous case, we considered different data transfer rates for each link. We also considered resource heterogeneity by setting the value of S_i to 2 for randomly half of the processors. As can be seen from Fig. 8, the performance of MELISA is far better than that of ELISA for a heterogeneous Grid environment. In Fig. 9, there is also some improvement in the total execution time for MELISA. This result shows that MELISA is largely suitable for a heterogeneous Grid environment.

5.1.2 Performance for the Homogeneous Case

This is a special case to our heterogeneous environment. In this case, we have considered all nodes to be homogeneous, which means that S_i is set to 1 for all processors. Also, the network bandwidth is fixed, and it is $10\ Mbps$ for all links. As shown in Figs. 10 and 11, ART and the total execution time are almost same for both algorithms. This observation indicates that our algorithm gives as good performance as that given by ELISA in the homogeneous case.

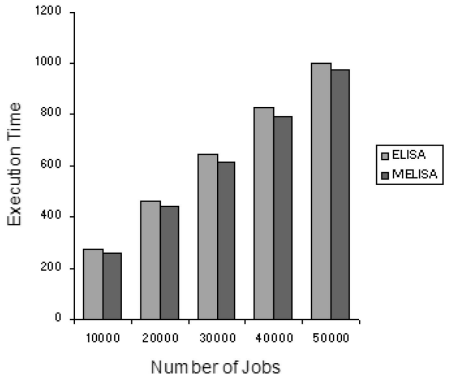


Fig. 9. Total execution time comparison for the heterogeneous case.

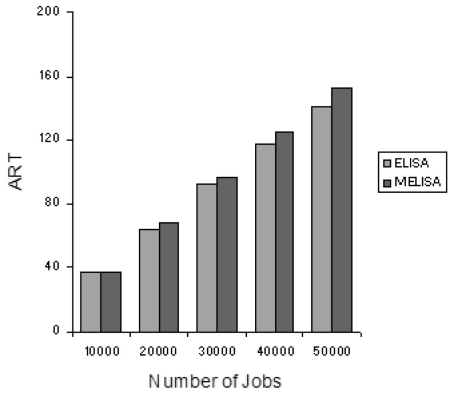


Fig. 10. ART comparison for the homogeneous case.

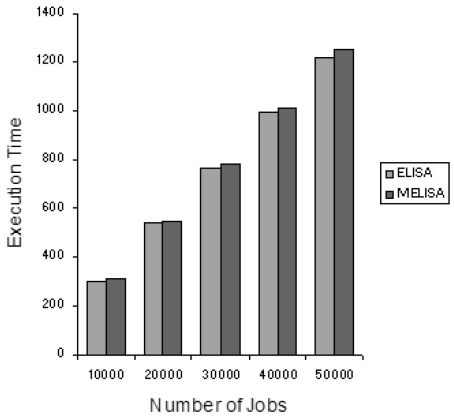


Fig. 11. Total execution time comparison for the homogeneous case.

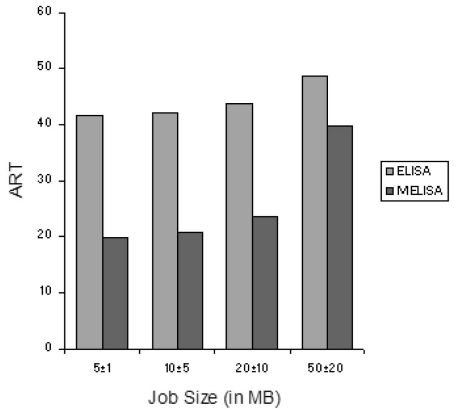


Fig. 12. ART comparison for varying job sizes.

5.1.3 Effect of Job Size

MELISA also takes into account the job migration cost, and as job size largely affects the job migration cost, it will be very interesting to measure the performance of the algorithms by varying the job size. In this set of experiments, we varied the job size from $5\ Mbytes \pm 1\ Mbyte$ to $50\ Mbytes \pm 20\ Mbytes$. In Figs. 12 and 13, as we increase the job size, the performance of our algorithm becomes far better than ELISA in terms of the decrease in ART and the total execution time. This result indicates that the performance of MELISA outperforms that of ELISA when the job migration cost is very large, which is the case for a large-scale Grid environment.

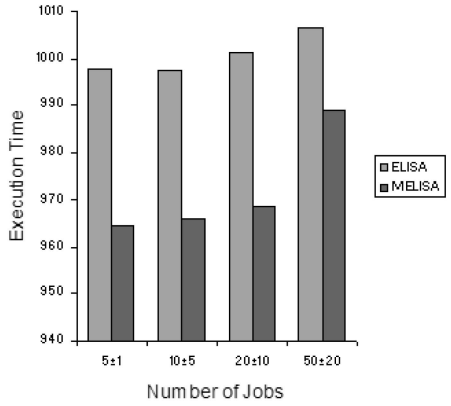


Fig. 13. Total execution time comparison for varying job sizes.

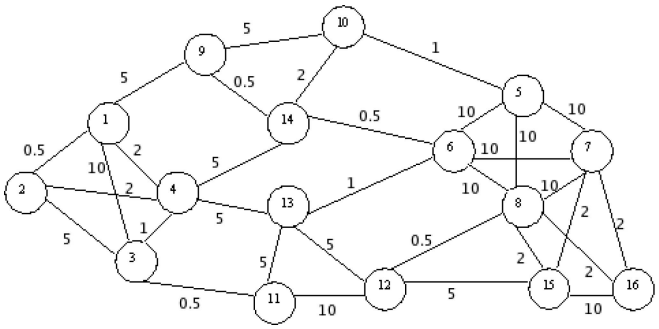


Fig. 14. Network topology.

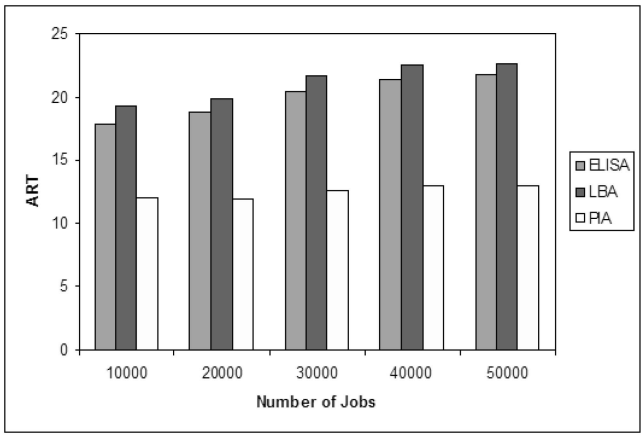


Fig. 15. ART comparison for random arrival and service rates.

5.2 Performance of LBA

In this section, we will evaluate the performance of our proposed LBA algorithm with ELISA and PIA. As LBA is proposed for a small-scale grid environment, we conducted an experiment for a Grid with 16 nodes, as shown in Fig. 14. The weight on each link indicates the data transfer rate in $Mbps$ in Fig. 14. We have considered different cases to measure the performance of LBA. The following sections describe the various cases and performance analysis.

5.2.1 Random Arrival and Service Rates

In this set of experiments, we have quantified the performance of our LBA algorithm for real-life situations wherein arrival rates and service rates are completely random. As seen in Fig. 15, there is not much difference in

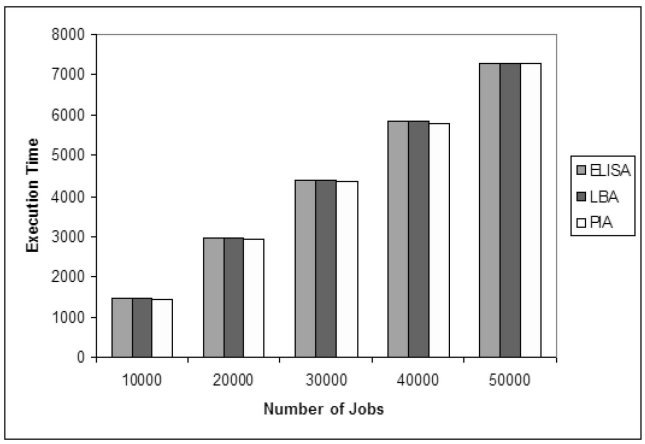


Fig. 16. Total execution time comparison for random arrival and service rates.

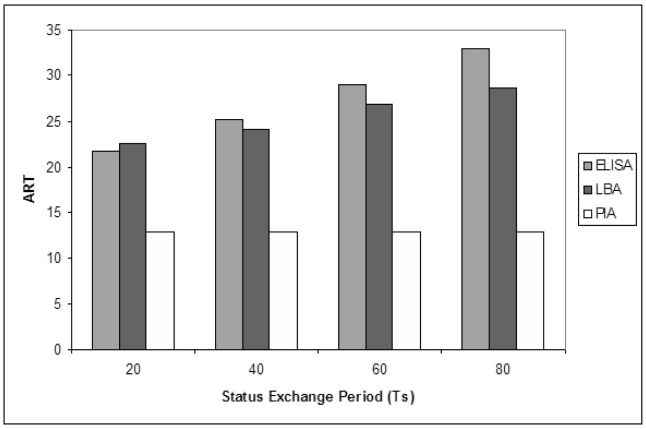


Fig. 17. Effect of the status exchange period (T_s) on ART.

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 as we can observe it even in Fig. 16.

5.2.2 Effect of the Status Exchange Period

ELISA is highly sensitive to the magnitude of the status exchange period T_s. 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 T_s also increases ART, but its performance is much better than that of ELISA. As seen in Fig. 17, by increasing the value of T_s, 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.

5.2.3 Effect of an 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.

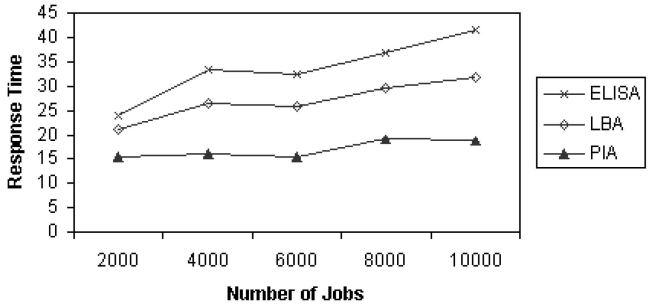


Fig. 18. ART comparison for an uneven load distribution.

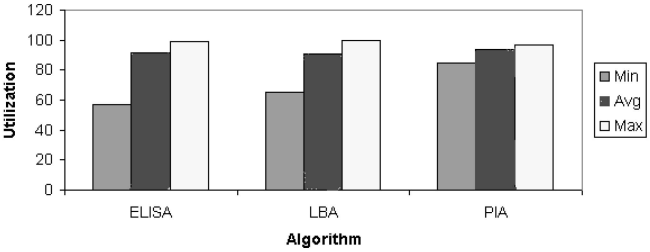


Fig. 19. Resource utilization comparison for an uneven load distribution.

For this set of experiments, we set the values of the arrival rates and service rates of processors in such a way that the load distribution is uneven across all processors. Here, (refer to Fig. 14) processors 1, 2, 3, and 4 are highly loaded, whereas processors 7, 8, 15, and 16 are lightly loaded. Other processors are moderately loaded. The results are shown in Figs. 18 and 19. From the graphs, we observe that LBA gives much better performance for ART. Fig. 19 shows the results for the minimum utilization of a processor, average utilization of the system, and maximum utilization of a processor, after executing all N jobs. These values indicate how the load has been balanced across processors. In Fig. 19, it may be observed that for ELISA, there is wide variation in terms of processor utilization. In LBA, the variation in processor utilization is less than that for ELISA and for PIA; there is very little or no variation in processor utilization.

5.2.4 Effect of Migration Limit

One of the important parameters for LBA is the *migration limit* (denoted as η), that is, *how many hops we should allow a job to migrate before execution*. Obviously, this decision depends on the network topology considered. By setting the value of η to the *maximum path length* of the graph, we can obtain almost the same result as when η is very large.[4] 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.

We conducted a set of experiments by varying the migration limit for the network topology shown in Fig. 14.

4. Theoretically, we are setting it to infinity.

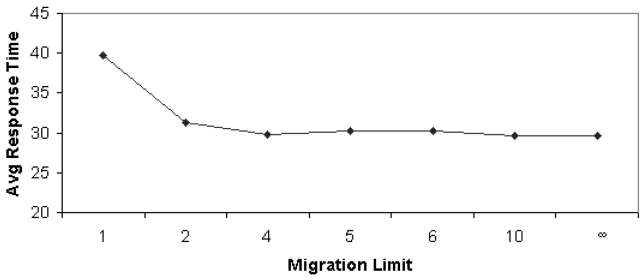


Fig. 20. Effect of η on ART.

As in Fig. 20, when $\eta > 4$, there is hardly any change in ART. Also, when $\eta = 4$, performance is better for the execution time (refer to Fig. 21). Thus, we observe that by setting the value of η around the maximum path length gives a better acceptable performance.

5.2.5 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. Fig. 22 shows the results for various job sizes ranging from $5\ Mbytes \pm 1\ Mbyte$ to $50\ Mbytes \pm 20\ Mbytes$. As it can be observed in Fig. 22, for a larger job size, the performance of LBA is better 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.

Remarks. As seen above, an increase in job size would lead to an increase in the job migration cost. However, this is handled differently by MELISA and LBA as follows. In MELISA, for each transfer instant, P_i will calculate the expected finish time (EFT_k^j) of every job j on buddy processor k. If $EFT_k^j < EFT_i^j$, then the job will be migrated to buddy processor k. In LBA, each P_i will calculate EFT_k^j for job j on buddy processor k only on its arrival. If no buddy processor k can finish the job earlier than P_i, then the job will be placed in the waiting queue of P_i. In LBA, this job j will be never migrated to any buddy processor and will be executed on P_i, whereas in MELISA, this job can be migrated if, after some time, there is at least one buddy processor who can finish the job earlier than this processor.

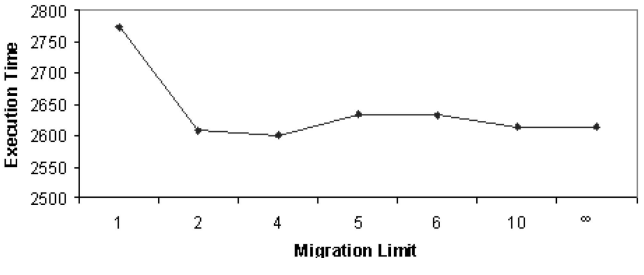


Fig. 21. Effect of η on the total execution time.

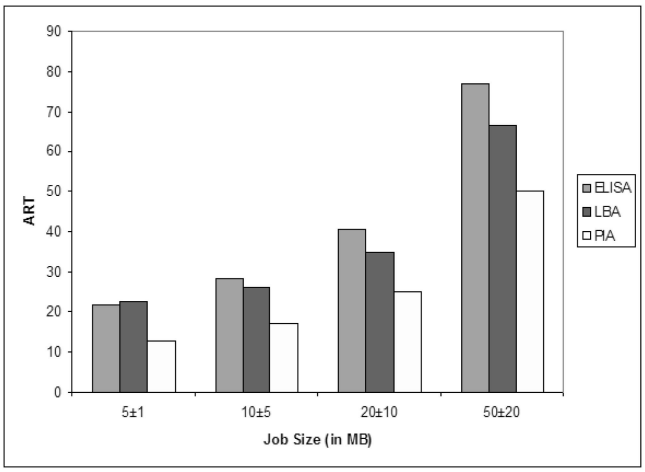


Fig. 22. Effect of job size on ART.

6 Conclusions

In this paper, we presented decentralized, scalable, adaptive, and distributed algorithms 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. Our algorithms are adaptive in the sense that they 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. Through this study, we demonstrate the usefulness and effectiveness of the load estimation approach to devise adaptive and dynamic load-balancing strategies for data-hungry computational Grid structures.

The simulation study shows that MELISA gives 44.9 percent better performance than ELISA for a heterogeneous environment when 50,000 jobs get completed. Our algorithms also consider overheads of job migration due to the large communication latency between Grid resources. MELISA makes a decision of job migration based on its expected finish time on a buddy processor, which also includes the transfer time of the job. MELISA gives 18 percent better performance than ELISA for large-scale Grid environments, that is, when the available bandwidth between two processors is very low and/or the job size is large. Therefore, MELISA is better suited for large-scale Grid environments. 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. Results show that the LBA algorithm gives 25 percent better performance when the load is not evenly balanced across all resources, whereas it gives 12.82 percent better performance than

ELISA in case of a high migration cost. For small-scale Grids, LBA is the best solution for load balancing across processors.

Through this study, we also observed the influence of λ (the arrival rates) at the nodes. We noted that by increasing the value of λ_i for P_i, the ART of the system also increases. This can be taken care of by tuning the estimation time epoch T_e. Thus, by decreasing the value of T_e (in case of (M)ELISA), we can improve the performance of the system as the load is now balanced more frequently. Further, there is a small nonzero probability that a load can shuttle between processors. In our strategies, we prevented this through a control parameter, the migration limit (denoted by η). That is, if a job is migrated for η times, then it will be not transferred again.

Although the main objective of our study 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.

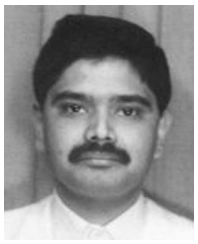
ACKNOWLEDGMENTS

This work is supported in part by Grant 052 015 0024/R-263-000-350-592 from the National Grid Office via A*Star, Singapore. Bharadwaj Veeravalli would like to thank the funding agency in supporting this research.

REFERENCES

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann, 1999.
[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] I. Foster, C. Kesselman, and S. 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.
[4] L. Smarr and C.E. Catlett, "Metacomputing," *Comm. ACM,* vol. 35, no. 6, pp. 44-52, June 1992.
[5] 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.
[6] 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.
[7] N. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *Computer,* vol. 25, no. 12, pp. 33-44, Dec. 1992.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "On the Design and Evaluation of Job Scheduling Algorithms," *Proc. Fifth Workshop Job Scheduling Strategies for Parallel Processing,* pp. 17-42, 1999.
[13] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," *Proc. Third Workshop Job Scheduling Strategies for Parallel Processing,* pp. 1-34, 1997.
[14] R. Shah, B. Veeravalli, and M. Misra, "Estimation Based Load Balancing Algorithm for Data-Intensive Heterogeneous Grid Environments," *Proc. 13th Int'l Conf. High Performance Computing (HiPC '06),* pp. 72-83, 2006.
[15] 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.
[16] 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.
[17] Z. Zeng and B. Veeravalli, "Design and Analysis of a Non-Preemptive Decentralized Load Balancing Algorithm for Multi-Class Jobs in Distributed Networks," *Computer Comm.,* vol. 27, pp. 679-693, 2004.
[18] H. Shan, L. Oliker, and R. Biswas, "Job Superscheduler Architecture and Performance in Computational Grid Environments," *Proc. ACM/IEEE Conf. Supercomputing,* Nov. 2003.
[19] Y. Wong, K. Leung, and K. Lee, "A Stochastic Load Balancing Algorithm for I-Computing," *Concurrency and Computation: Practice and Experience,* vol. 15, no. 1, pp. 55-787, Jan. 2003.
[20] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "Distributed Job Scheduling on Computational Grid Using Multiple Simultaneous Requests," *Proc. 11th IEEE Symp. High Performance Distributed Computing (HPDC '02),* July 2002.
[21] 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.
[22] H.A. James and K.A. Hawick, "Scheduling Independent Tasks on Metacomputing Systems," *Proc. 12th Int'l Conf. Parallel and Distributed Computing Systems (PDCS '99),* Mar. 1999.
[23] R. Martin, A. Vahdat, D. Culler, and T. Anderson, "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," *Proc. 24th Ann. Int'l Symp. Computer Architecture (ISCA '97),* pp. 85-97, June 1997.
[24] M. Mitzenmacher, "How Useful Is Old Information," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 1, pp. 6-20, Jan. 2000.

Ruchir Shah received the degree BE in information technology from Sardar Patel University, India, in 2003 and the MTech degree in computer science from the Indian Institute of Technology, Roorkee, India, in 2006. He is currently working at Yahoo! (R&D), Bangalore, India. His research interests include Grid computing, load balancing and scheduling in parallel and distributed systems, and computer networks.



Bhardwaj Veeravalli received the BSc degree in physics from Madurai-Kamaraj University, India, in 1987, the master's degree in electrical communication engineering from the Indian Institute of Science (IISc), Bangalore, India, in 1991 and the PhD degree from the Department of Aerospace Engineering, IISc, Bangalore, India, in 1994. He did his postdoctoral research in the Department of Computer Science, Concordia University, Montreal, in 1996. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering Division, National University of Singapore, Singapore, as an associate professor. His mainstream research interests include multiprocessor systems, cluster/Grid computing, scheduling in parallel and distributed systems, bioinformatics, and multimedia computing. He has published extensively in high-quality international journals and conference proceedings and is a coauthor of three research monographs in the areas of parallel and distributed systems, distributed databases, and networked multimedia systems. He is currently serving the editorial board of the *IEEE Transactions on Computers*, *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, and the *International Journal of Computers and Applications* as an associate editor. He is a senior member the IEEE and the IEEE Computer Society.

Manoj Misra received the PhD degree in computer science from the University of Newcastle, Tyne, United Kingdom. He is currently an associate professor in the Department of Electronics and Computer Engineering, Indian Institute of Technology (IIT), Roorkee. He has published more than 50 papers in reputed international journals and conference proceedings. His current research interests are mobile computing, performance modeling, and distributed computing. He is a member of the IEEE.