On the performance-driven load distribution for heterogeneous computational grids

Kai Lu, Riky Subrata, Albert Y. Zomaya*

Networks & Systems Lab, School of Information Technologies, University of Sydney, NSW 2006, Australia
Received 11 March 2006; received in revised form 3 October 2006
Available online 24 February 2007

Abstract
Load balancing has been a key concern for traditional multiprocessor systems. The emergence of computational grids extends this challenge to deal with more serious problems, such as scalability, heterogeneity of computing resources and considerable transfer delay. In this paper, we present a dynamic and decentralized load balancing algorithm for computationally intensive jobs on a heterogeneous distributed computing platform. The time spent by a job in the system is considered as the main issue that needs to be minimized. Our main contributions are: (1) Our algorithm uses site desirability for processing power and transfer delay to guide load assignment and redistribution, (2) Our transfer and location policies are a combination of two specific strategies that are performance driven to minimize execution cost. These two policies are the Instantaneous Distribution Policy (IDP) and the Load Adjustment Policy (LAP), (3) The communication overhead involved in information collection is reduced using mutual information feedback. The simulation results show that our proposed algorithm outperforms conventional approaches over a wide range of system parameters.

Crown Copyright © 2007 Published by Elsevier Inc. All rights reserved.

Keywords: Computational grids; Load balancing; Distributed computing; Heterogeneity; Migration

1. Introduction
The demand for extra computing power is never ending. Users would like to utilize the CPU cycles of idle or under-utilized computing resources. Emerging distributed computing environments, such as the Computational Grids [1] provide an opportunity to share a large number of resources among different organizations. Each of these grid sites could be a distributed network of workstations, a cluster, a distributed memory multi-processor system, or a massively parallel supercomputer.

Due to uneven job arrival patterns and unequal computing capacities and capabilities, the computers in one grid site may be overloaded while others in a different grid site may be under-utilized. It is therefore desirable to dispatch jobs to idle or lightly loaded computers in the grid environment to achieve better resource utilization and reduce the average job response time. This is a natural extension of the existing work on load balancing in a traditional distributed
system. Many load balancing algorithms (e.g. [2–5]) have been proposed for traditional distributed systems. However, the structure of the grid environment is much more complex than that of a single distributed system.

Only a handful of papers have proposed serious solutions to the scheduling problem given the unique characteristics of grid environments [14]: large-scale, heterogeneous grid sites, and considerable transfer delay. The main reason for this is the intractable nature of the scheduling problem given such a complex set of constraints. In this paper, we propose a novel dynamic load balancing algorithm especially designed to tackle the above characteristics. We utilize site desirability to guide load assignment and redistribution. For each site \( s_i \) in the grid, our algorithm uses desirability of other sites to \( s_i \) based on processing power to form \( k \) number of partners and uses desirability of other sites to \( s_i \) based on transfer delay to form \( p \) number of neighbors for the site \( s_i \). Partners are sites with comparable or greater processing powers, and neighbors are nearby sites with low transfer delays. The set of partners for a site \( s_i \) are updated dynamically at runtime based on feedback information, which means a heavily loaded partner site can be replaced with a low loaded site from a set of preferred sites for site \( s_i \). The set of preferred sites for a site \( s_i \) can be found using sites clustering approach with the aid of a set of reference sites. A new job arriving at a site \( s_j \) is immediately distributed to the site \( s_j \) or one of its partner sites. Continuous load adjustment is employed among neighbor sites. In order to reduce/minimize the state-collection overhead in our proposed load balancing algorithm, state information exchange is done via mutual information feedback. Extensive simulation studies were conducted to analyze the performance of our proposed decentralized dynamic load balancing algorithm. The proposed algorithm is compared with a number of well-known algorithms. It is observed that our algorithm reduces the average job response time over a wide range of system parameters.

The rest of the paper is organized as follows. Section 2 presents related work and our motivation. Section 3 presents the system model. Section 4 describes in details the design of the proposed algorithm. In Section 5, the performance of our algorithm is compared with other well known algorithms in a series of simulations. Finally, this paper is concluded in Section 6.

2. Related work and our motivation

In the past decade, a lot of research has focused on the development of effective load balancing algorithms for distributed computing systems. Load balancing algorithms can be classified into static and dynamic approaches [6].

Static load balancing algorithms (e.g. [7]) assume that \textit{a priori} information about all the characteristics of the jobs, the computing nodes and the communication network are known and provided. Load balancing decisions are made deterministically or probabilistically at compile time and remain constant during runtime. The static approach is attractive because of its simplicity and the minimized runtime overhead. However, it has two major disadvantages. Firstly, the workload distribution of many applications cannot be predicted before program execution. Secondly, it assumes that the characteristics of the computing resources and communication network are all known in advance and remain constant. Such an assumption may not apply to a grid environment. Because the static approach cannot respond to the dynamic runtime environment, it may lead to load imbalance on some nodes and significantly increase the job response time.

In contrast, dynamic load balancing algorithms attempt to use the runtime state information to make more informative decisions in sharing the system load. It is now commonly agreed that despite the higher runtime complexity, dynamic algorithms can potentially provide better performance than static algorithms.

Dynamic load balancing algorithms can be further classified into a centralized approach and a decentralized approach. In the centralized approach (e.g. [8,9]), only one node in the distributed system acts as the central controller. It has a global view of the load information in the system, and decides how to allocate jobs to each of the nodes. The rest of the nodes act as slaves; they only execute the jobs assigned by the controller. The centralized approach is more beneficial when the communication cost is less significant, e.g. in the shared-memory multi-processor environment. Many authors argue that this approach is not scalable, because when the system size increases, the central controller may become a system bottleneck and the single point of failure.

In the decentralized approach (e.g. [10]), all nodes in the distributed system are involved in making the load balancing decision. It is commonly agreed that decentralized algorithms are more scaleable and have better fault tolerance. Since the load balancing decisions are distributed, it is costly to let each node obtain the dynamic state information of the whole system. Hence, most algorithms [11–13] only use partial information stored in the local node to make a sub-optimal decision.
In addressing this problem, Mosix [11] used a simple probabilistic model to choose a random subset of hosts to talk to and cut down communication. As the job needs to wait for the polling result, polling will increase the response time of the waiting job. This is a problem if the transfer delay is significant. The diffusion-based approaches [12, 13] are a highly distributed local approach which makes use of near-neighbor load information to apportion surplus load from heavily loaded processors to under-loaded neighbors in the system. Global balancing is achieved as tasks from heavily loaded neighborhoods diffuse into lightly loaded areas in the system. Their approach is valid for tightly-coupled multiprocessor systems, and do not take jobs’ execution cost into account.

Although only a small number of studies addressed dynamic load balancing problems for large heterogeneous grid systems, however, several important outcomes have already been realized. A recent study in [14] proposed a decentralized scheduling and load balancing algorithm for a grid environment which considers job resource requirements and overhead of searching for appropriate N-resource node by overlapping the time of the actual processing of ready jobs. The goal of the study is different from this work. Our algorithm aims to minimize the average response time of jobs. In our previous study [15], we proposed a dynamic and decentralized load balancing algorithm for computational grids. In this paper, we describe a more efficient version of the algorithm that includes a more realistic job execution cost model, the consideration of performance benefit regarding job migration, and better approaches for the selection of partner sites.

The main motivation of our study is to propose a decentralized dynamic load balancing algorithm that can cater for the following unique characteristics of practical Grid Computing environment.

- **Large-scale.** As a grid can encompass a large number of high performance computing resources that are located across different domains and continents, it is difficult for centralized model to address communication overhead and administration of remote workstations.
- **Heterogeneous grid sites.** There might be different hardware architectures, operating systems, computing power and resource capacity among different sites. In this study, heterogeneity only refers to the processing power of the site.
- **Effects from considerable transfer delay.** The communication overhead involved in capturing load information of sites before making a dispatching decision can be a major issue negating the advantages of job migration. We should not ignore the considerable dynamic transfer delay in disseminating load updates on the Internet.

Our target is to devise a general and practical algorithm to cater for the above grid environment characteristics.

### 3. System model

#### 3.1. Architecture model

It is assumed that the grid system consists of a collection of sites \( S \) connected by a communication network, as shown in Fig. 1. The set \( S \) contains \( n \) sites, labeled as \( s_1, \ldots, s_n \). Logically, the site architecture is hierarchical and is divided into three levels: the Grid-Level, the Site-Level and the Node-Level. In Fig. 1, they are denoted as G, S and N, respectively. The Grid-Level scheduler is responsible for load control among grid sites. The Site-Level consists of a collection of computing nodes. The Site-Level scheduler can fully control the computing nodes within the site but cannot operate the computing nodes in other sites directly. The management of jobs at Site-Level is addressed by many prototype and commercial systems [16]: Condor, Load Sharing Facility, Portable Batch System, LoadLeveler, and many others. The Node-Level is a computing node. These computing nodes at each grid site are typically interconnected by a high-speed local network and protected by firewalls from the outside world. The sites in the grid system may have different processing power. Processing power is measured by the average CPU speed across all computing nodes within a grid site \( s_i \), denoted as \( \text{APW}_i \). For \( i \neq j \), \( \text{APW}_i \) may be different from \( \text{APW}_j \). The processing power information can be retrieved from resource information provider such as GIIS (Grid Index Information Service) [17], which is a component of the Metacomputing Directory Service (MDS) that provides resource information based on static attributes (architecture, processing power, etc.). To clarify the statement and emphasize our main idea, we simplify the model of grid site as one computing node with a single processor. Actually, our scheduling can be easily extended to accommodate aforementioned complicated cases.
3.2. Communication model

The sites $S$ are fully interconnected, meaning that there exists at least one communication path between any two sites in $S$. The only way for inter-site communication is through message passing. There is a non-trivial transfer delay on the communication network between the sites. The transfer delay is different between different pairs of sites. The underlying network protocol guarantees that messages sent across the network are received in the order sent. The sites are interconnected by point-to-point links. There is no efficient broadcasting service available.

Our communication model represents the network performance between any site pair $(S_i, S_j)$ using two parameters: a transfer delay $TD_{ij}$ and a data transmission rate $BW_{ij}$. The communication time for sending a $Z$ bytes message between these sites is then given by $TD_{ij} + Z/BW_{ij}$, where $Z/BW_{ij}$ is the transmission time. The two parameters abstractly represent the total time for traversing all the links on the path between $S_i$ and $S_j$. $TD_{ij}$ includes a start-up cost and delays incurred by contention at intermediate links on the path between $S_i$ and $S_j$. $TD_{ij}$ and $BW_{ij}$ can be dynamically forecasted by the Network Weather Service [18]. Other research on estimating host distance between any two IP addresses can also be found in the literature [19–21].

3.3. Job model

For any site $s_i \in S$, there are jobs arriving at $s_i$. The jobs are assumed to be computationally intensive, mutually independent, and can be executed at any sites. Job execution is not time-shared but dedicated. Each job $j_x$ that arrives at a grid site $s_i$ has mainly one parameter: an expected execution time $ETC(j_x, s_{std})$, that is, the time that would be taken at a standard platform (with an $APW$ equal to 1) for processing that job. On a site $s_i$ with $APW_i$, the expected execution time of a job $ETC(j_x, s_i)$ will therefore be $ETC(j_x, s_{std})/APW_i$. The assumption that the expected job execution times are known is commonly made when studying scheduling and load balancing techniques for heterogeneous computer systems [22–25]. Several approaches for performing this estimation can be found in [18,26–28]. As soon as a job arrives, it must be assigned to exactly one site for processing. When a job is completed, the executing site will return the results to the originating site of the job. We use $J$ to denote the set of all jobs generated at $S$, $J = \{j_1, \ldots, j_r\}$. We also assume that the job file size is known and job result file size can be estimated accurately.

3.4. Job queue model

We assume that there exists a global job-waiting queue at each site, which holds those jobs waiting to be assigned to a computing site (see Fig. 2). Only the jobs in the global job-waiting queue are eligible for load migration. We use $GJQ(s_i)$ to denote the global job-waiting queue of the site $s_i$. The jobs in the global job-waiting queue are processed in “First-Come-First-Serve” order.
3.5. Job migration

Some researchers have considered job migration (migration of partly executed jobs) in their load balancing algorithms (e.g. [29,30]). However, job migration is far from trivial in practice. It involves collecting all system states (e.g. virtual memory image, process control blocks, unread I/O buffer, data pointers, timers, etc.) of the job, which is very formidable. Many studies (e.g. [4,31,32]) have shown that: (1) job migration is often difficult in practice, (2) the operation is generally expensive in most systems, and (3) there are no significant benefits of such a mechanism over those offered by non-migratory counterparts. Hence, we only consider the migration of jobs that are not already running in this paper.

Because each site scheduler acts independently, there is a small non-zero probability that a job can shuttle between sites. This can be prevented in various ways. One approach used throughout our simulation makes the job join not at the end of the queue, but at the position where it should have been if the job had arrived at that queue. This means that we keep track of the time at which it left the last site in the job transfer procedure. This can considerably reduce the probability of the job being transferred once again and can guarantee minimizing the response time of that job.

A more conservative approach is used to reduce the rate at which jobs are moved from one site to another. This can be achieved by restricting the maximum number of jobs transmitted between sites (i.e. maximum one job) at any given time. This approach might be less responsive in some cases, but is more robust and requires minimal processing power and time at each site.

3.6. Definition and objective

∀ji ∈ J we define the following functions, which will be used throughout this paper:

1. bornSite(ji) denotes the originating site of the job ji.
2. exeSite(ji) denotes the executing site of the job ji.
3. bornTime(ji) denotes the arrival time of job ji, which is the time when the job is generated at bornSite(ji).
4. deathTime(ji) denotes the finish time of ji, which is the time when site bornSite(ji) receives ji’s results from exeSite(ji).
5. respTime(ji) denotes the completion time of ji; respTime(ji) = deathTime(ji) − bornTime(ji).

The objective of our load balancing algorithm is defined by:

\[
\text{minimize} \left( \frac{\sum_{ji \in J} \text{respTime}(ji)}{u} \right),
\]

where u represents the number of jobs submitted to the system and completed in the time interval [t1, t2], which is called the observation period. In other words, our goal is to minimize the average job response time, denoted as ART in the paper. Note that u < r.
Dynamic load balancing algorithms can be classified into sender-initiated, receiver-initiated and/or symmetrically-initiated algorithms according to their location policies [3,33]. Sender-initiated algorithms allow the heavily loaded sites to take the initiative to request the lightly loaded sites to receive the jobs, while receiver initiated algorithms permit the lightly loaded sites to invite heavily loaded sites to send their jobs. Symmetrically-initiated algorithms combine the advantages of these two by requiring both senders and receivers to look for appropriate sites. In this study, we only consider sender-initiated algorithms.

Our objective is achieved by utilizing site desirability to guide load assignments. Site desirability is based on how site characteristics will affect the performance of future load balancing. Site desirability of a site $s_j$ to a site $s_i$ includes two definitions: desirability based on processing power of site $s_j$, and desirability based on transfer delay between $s_j$ and $s_i$.

In the following sections we discuss in details the proposed decentralized dynamic load balancing algorithm.

4.1. Sites clustering

Here, site desirability of processing power is accounted for using sites cluster. Our sites clustering algorithm uses a set of reference sites of size $m$ (e.g. $m = 10$ reference sites). The reference sites are chosen at random with the only condition that the reference sites be separated from each other by a large enough difference in processing power. Similar approaches have been widely used to generate proximity information (e.g., [21,35]). It is based on an intuition that sites close to each other in processing power are likely to have similar distances to a few selected sites.

These reference sites are sorted by $APW$ in descending order before applying sites clustering approach. For each grid site $s_j$, the clustering algorithm first measures the difference in $APW$ of site $s_j$ to the reference sites and calculate a reference vector $\langle d_1, d_2, \ldots, d_m \rangle$. Two grid sites with similar reference vector are ‘close’ to each other in terms of processing power. The grid sites are then clustered into $C_1, C_2, \ldots, C_m$ clusters. Finally, empty clusters in $C_1, C_2, \ldots, C_m$ are removed so that we have $C_1, C_2, \ldots, C_q (q \leq m)$, which are also in decreasing order of $APW$. We denote the cluster ID that a grid site $s_j$ belongs to as $\Omega_i$, with possible values of positive integer between 1 and $q$. Note that sufficient number of reference sites will need to be used to reduce the probability of false clustering where sites that have very different processing power have similar/close reference vectors. In our study, we use 10 reference sites for the sites clustering and these reference sites are separated from each other by 10 percent difference in processing power.

The approach outlined above is a coarse-grained approximation and is not effective in differentiating sites with close distances. However, our experiment results show that the method works well for our load balancing scheme. This is largely because our load balancing scheme does not require very precise measurements. The clusters generated are then used in the partner sites generation, described below.

4.2. Partners

Each scheduler automatically maintains $k$ number of partner sites $PSet_i$, which the scheduler will use to select a partner site for processing new arriving jobs.

When a site first joins the grid system, it will attempt to determine partners for itself. We employ a simple heuristic to find partner sites in terms of their processing power. It is natural to consider more powerful sites as partners. Here we consider heterogeneity of sites. Pseudo-code for our partners’ selection procedure is given in Algorithm 1 shown in Fig. 3. $Q_i$ is a preferred collection of sites of site $s_i$ and are also used in Partners Adjustment Policy. The sites in $Q_i$ have greater or comparable processing power to site $s_i$. Note in the algorithm that the set of preferred sites $Q_i$ may be updated, as necessary. Although the approach described here does not guarantee to find optimal partners, the methodology provides a scalable and performance-efficient approach to initial partner sites formation.

4.3. Neighbors

Each scheduler also maintains $p$ number of neighboring sites $NSet_i$, which the scheduler will use to select a neighboring site for offloading jobs. This can reduce the cost of load movement, and enable quick response to load imbalances. Neighbors for each site are formed in terms of transfer delay. For a site $s_i$, a site $s_j$ is considered as its
Algorithm 1 (Procedure FindPartners(s_i, k)).

Find all sites s_j \in S (i \neq j) with \Omega_j < \Omega_i. Denote this set of sites as Q_i
If \gamma \geq k then \gamma^+ is the size of Q_i^+/
Pick k sites from Q_i randomly and add them to PSet_i
Else
\gamma = k - \gamma
Add Q_i to PSet_i.
\nu = \Omega_i
While \gamma > 0
Q_i = Q_i \cup C_v
If \gamma^+ \geq l then \gamma^+ is the size of C_v^+/
Pick \gamma sites from C_v randomly and add them to PSet_i
Break
Else
Add C_v to PSet_i.
\gamma = \gamma - \gamma^+
\nu = \nu + 1
End if
End While
End if

Fig. 3. Initial partners formation.

neighboring site as long as the transfer delay between the site s_j and s_i is within \varepsilon times of the transfer delay between the site s_i and the nearest site. For our experiments, we have found \varepsilon = 1.5 to yield very good results and this value is used throughout the experiments. Note that, any number of set relationships between PSet_i and NSet_i is possible, including intersect, disjoint, include, etc.

4.4. Load index

An important issue in designing a dynamic load balancing algorithm is to identify the load index that measures the current loading of a site. Kunz [34] reports that the simple CPU queue length load index is the most effective. Most algorithms in the literature have used the instantaneous run-queue length (i.e. the number of jobs being served or waiting for service at the sampling instant) as the load index [4,34]. This approach is based on the ‘join the shortest queue’ intuition. The run-queue length may be a good load index if we assume that all the nodes of the system are homogeneous and the inter-node transfer delay is negligible or constant. However, it is not a reliable load indicator in a heterogeneous environment. It ignores the variations in computing power. Owing to the above reasons, we do not use the run-queue length as the load indicator. Instead, a cumulative job execution time is utilized.

\forall s_i \in S, the load index of s_i at a particular instant of time t is defined as

LD_{i,t} = TET_{i,t} + RET_{i,t},

where TET_{i,t} is the total estimated job execution time of all jobs currently waiting in the job queue on s_i at time instant t, and RET_{i,t} is the estimated remaining time of the job currently being processed by site s_i at time instant t.

4.5. Execution cost

Unlike conventional approaches that only consider the load index in calculating the cost of executing a job on a computing node we include the dynamic communication cost in the cost calculation. This is because the dynamic and considerable communication cost may have a great influence on the performance of a load balancing algorithm in a grid environment. It may be more efficient to send a job to a node with heavier load but smaller communication cost.

\forall s_i, s_j \in S, the execution cost of sending a job j_x \in J from s_i to s_j at time instant t is estimated by s_i as

If TRAN_IN(j_x, s_i, s_j, t) \geq LD_{j,t}, Then
EC(j_x, s_i, s_j, t) = TRAN_IN(j_x, s_i, s_j, t) + ETC(j_x, s_j) + TRAN_OUT(j_x, s_j, s_i, t)
Else
EC(j_x, s_i, s_j, t) = LD_{j,t} + ETC(j_x, s_j) + TRAN_OUT(j_x, s_j, s_i, t)
End if
where

- TRAN_IN\((j_x, s_i, s_j, t)\) measures how long it takes to transfer a job from site \(s_i\) to site \(s_j\).
- TRAN_OUT\((j_x, s_j, s_i, t)\) measures how long it takes to transfer a job result from site \(s_j\) to site \(s_i\).
- \(LD_{j,t}\) is the recent load index of site \(s_j\) at time instant \(t\) that was recorded in site \(s_i\).
- ETC\((j_x, s_j)\) denotes the expected execution time of job \(j_x\) at site \(s_j\).

4.6. Performance benefit

The performance benefit associated to a job \(j_x\) is based on the idea that better migration can be achieved by assigning a job to a grid site that would ‘benefit’ most in terms of expected response time if that grid site is assigned to it. In this study, the value of performance benefit of a job \(j_x\) is the difference between its estimated execution cost at local site and its estimated execution cost at a remote site, labeled as \(B_x\).

4.7. Information policy

Each site \(s_i\) maintains the state information of other sites by using a state object \(O_i\). The state object helps a site to estimate the load of other sites at any time without message transfer. The state object \(O_i\) of a site \(s_i \in S\) is an \(n\)-dimensional ArrayList object maintained by \(s_i\). Each item \(O_i[j]\) is a state object and has a property list \((LD, LT)\):

- \(O_i[j].LD\) denotes the load information of site \(s_j\).
- \(O_i[j].LT\) denotes the site \(s_j\)’s local time when the load status information is reported.
- Each site collects and maintains the state information of only its partners and neighbors. For any site \(s_j \in S\), \(s_j\) maintains its state information in its state object element \(O_i[i].O_i[j].LD\) and \(O_i[j].LT\) is maintained through message exchanges with partners and neighbors.

We could not use state-broadcast because the broadcast services are not available in grid. We also could not use the state-polling approach because it has few problems in practice:

- Repeated polling wastes the processing time of the polling sites and polled sites. This problem becomes significant when the general system load is heavy. When most of the sites are heavily loaded, they keep on polling each other for the sparse lightly loaded site. In the worst case, polling may cause system instability when all the sites are heavily loaded.
- Repeated polling generates a large amount of network traffic. This problem will become more significant if the network bandwidth is limited.
- As the job needs to wait for the polling result, polling will increase the response time of the waiting job. This is a problem if the communication delay is significant.
- It is difficult to set a good value for the probe limit. In a medium to heavily loaded system, if the probe limit is small, lightly loaded nodes may not be discovered. If the probe limit is large, then: (a) most of the heavily loaded nodes may find the same lightly loaded nodes and dump their loads to them, (b) the above problems caused by repeated polling will multiply.

In order to reduce/minimize the overhead of information collection, state information exchange is done by mutual information feedback. Algorithm 2 shown in Fig. 4 outlines the procedure when \(s_i\) transfers a job \(j_x\) to a neighbor or partner site \(s_j\) for processing. \(s_i\) appends the load information of itself and \(\omega_P\) (a small positive integer) random neighbors or partners to the job transfer request sent to \(s_j\) by piggybacking. \(s_j\) then updates the corresponding load information in its state object by comparing the timestamps if the sites contained in the transfer request belong to its neighbors or partners. Similarly, \(s_j\) inserts the current load information of itself and \(\omega_P\) random sites from its \(NSet_j\) and \(PSet_j\) in the job acknowledge or completions reply to \(s_i\), so \(s_i\) can update its state objects.

For any site \(s_i \in S\), if the state object element \(O_i[j]\) (\(\forall s_j \in NSet_i \cup PSet_i, \ i \neq j\)) has not been updated for a predefined period \(T_P\), then the load balancing scheduler will send an information exchange message to \(s_j\). The procedure is the same as the Algorithm 2 shown in Fig. 4.
Algorithm 2 (Job transfer procedure and Information exchange).

Steps processed in $s_i$:

1. $Y \leftarrow s_i + \{\omega_P\text{ random sites from } NSet_i \cup PSet_i - s_j\}$
   /* $s_i$ select neighbors or partners for information exchange */
2. $\forall y \in Y, s_i$ appends $(O_i[y].LD, O_i[y].LT)$ to the job transfer request TR
3. $s_i$ appends the departure time $DT_x$ to TR
4. $s_i$ appends bornsite $(jx)$ to TR
5. $s_i$ sends message TR to $s_j$

Steps processed in $s_j$:

Upon receiving TR:

1. $\forall y \in Y$: If $(O_i[y].LT > O_j[y].LT)$ and $(s_y \notin NSet_j \cup PSet_j)$ then $O_j[y] \leftarrow O_i[y]$
   /* $s_j$ updates the state object using $s_i$'s info */
2. Insert the job $j_x$ in the position where it should have been in terms of $DT_x$
   /* Reducing the probability of job shuttle between sites */
3. $Z \leftarrow s_j + \{\omega_P\text{ random sites from } NSet_j \cup PSet_j - s_i\}$
4. $\forall z \in Z, s_j$ appends $(O_j[z].LD, O_j[z].LT)$ to the acknowledge reply AR
5. $s_j$ sends message AR to $s_i$

Upon completion of job $j_x$:

1. $Z \leftarrow s_j + \{\omega_P\text{ random sites from } NSet_j \cup PSet_j - s_i\}$
2. $\forall z \in Z, s_j$ appends $(O_j[z].LD, O_j[z].LT)$ to the completion reply CR
3. $s_j$ sends message CR to bornsite $(jx) = s_i$

Steps processed in $s_i$:

Upon receiving the reply AR or CR:

$\forall z \in Z$: If $(O_j[z].LT > O_i[z].LT)$ and $(s_z \notin NSet_i \cup PSet_i)$ then $O_i[z] \leftarrow O_j[z]$

Algorithm 3 (Procedure PartnersAdjustment$(s_j, k)$).

$S_1 \leftarrow \phi$

$\forall y \in Y$: If $(s_y \notin NSet_j \cup PSet_j)$ and $(s_y \in Q_j)$ then $S_1 \leftarrow S_1 \cup s_y$
If $S_1 \neq \phi$ then
    $S_1 \leftarrow S_1 \cup PSet_j$
    Sort $S_1$ by $LD$ in ascending order
    Remove all sites from $PSet_j$
    Pick the first $k$ sites from $S_1$ and add them to $PSet_j$
End if

Fig. 4. Job transfer procedure and information exchange.

Algorithm 3 (Procedure PartnersAdjustment$(s_j, k)$).

$S_1 \leftarrow \phi$

$\forall y \in Y$: If $(s_y \notin NSet_j \cup PSet_j)$ and $(s_y \in Q_j)$ then $S_1 \leftarrow S_1 \cup s_y$
If $S_1 \neq \phi$ then
    $S_1 \leftarrow S_1 \cup PSet_j$
    Sort $S_1$ by $LD$ in ascending order
    Remove all sites from $PSet_j$
    Pick the first $k$ sites from $S_1$ and add them to $PSet_j$
End if

Fig. 5. Partners adjustment policy.

4.8. Partners adjustment policy

The dynamic Partners Adjustment policy is triggered whenever a site $s_i$ receives load information message from a neighbor or partner. If a site $s_j$ in the preferred sites $Q_j$ of site $s_i$ is found in the message, it will be involved in partner adjustment of site $s_i$. It is possible that the site $s_j$ becomes a partner site of site $s_i$ if its load is lower than the highest load in the partner sites of site $s_i$.

Algorithm 3 shown in Fig. 5 outlines the procedure when $s_j$ receives an information message from its neighbor or partner site $s_i$ for processing. The procedure is similar to the information exchange procedure at site $s_j$ as shown in Fig. 4.

4.9. Transfer policy and location policy

Our transfer policy and location policies are a combination of two policies—*instantaneous distribution policy* (IDP) and *load adjustment policy* (LAP). These are described below.
Algorithm 4 (Instantaneous Distribution Policy).
\[ \forall j_x \in J \text{ with } \text{bornSite}(j_x) = s_i \in S \]
For each \( s_j \) in \( PSet_i \)
  - Calculate \( EC(j_x, s_i, s_j, t) \)
  - Calculate related benefit value \( B_x \)
End For
Find the partner site \( s_j \) that gives the maximum \( B_x \)
If \( B_x > \theta \) then /* \( \theta \) is a positive real constant close to zero */
  - Transfer the job \( j_x \) to the partner site \( s_j \)
  - Update load index of site \( s_j \) recorded at the site \( s_i \)
Else
  - \( \text{GJQ}(s_i) \leftarrow \text{enqueue}(j_x) \) /* put the job \( j_x \) in the job queue \text{GJQ}(s_i) */
End if

Fig. 6. Instantaneous distribution policy.

Algorithm 5 (Load Adjustment Policy).
For each \( s_x \) in \( NSet_i \)
  - For each Job \( j_x \) in \( \text{GJQ}(s_i) \)
    - Calculate \( EC(j_x, s_i, s_x, t) \)
  End for
End for
For each Job \( j_x \) in \( \text{GJQ}(s_i) \)
  - Find the site \( s_y \) that gives the minimum execution cost
  - Calculate related benefit value \( B_x \)
End For
Sort the jobs in \( \text{GJQ}(s_i) \) in ascending order by their benefit value
Pick the Job \( j_y \) with the biggest benefit value \( B_y \)
Find the neighboring site \( s_j \) that gives the maximum \( B_y \) to \( j_y \)
If \( B_y > \theta \) then /* \( \theta \) is a positive real constant close to zero */
  - Remove the job \( j_y \) from \( \text{GJQ}(s_i) \)
  - Transfer the job \( j_y \) to the neighboring site \( s_j \)
  - Update load index of site \( s_j \) recorded at the site \( s_i \)
End if

Fig. 7. Load adjustment policy.

4.9.1. Instantaneous distribution policy (IDP)

When a new job arrives at site \( s_i \), the load balancing algorithm decides whether it is to be sent to the global job queue of site \( s_i \) or other partner sites \( PSet_i \). The decision depends on whether the job can get performance benefit if it is distributed to one of its partner sites. If there is no performance benefit the job can gain, it is placed in the global job queue of site \( s_i \) and involved in load balancing performed by another policy at a later time (see Section 4.9.2). The policy has two advantages: First, the policy tries to control the job processing rate on each site in the system; Second, the policy makes more powerful sites carry more loads, and jobs executed at fast sites are more likely to execute at a high speed. If there are two partner sites with the same minimum load, the nearest partner site is chosen. The instantaneous distribution policy for a site \( s_i \) is described in Fig. 6.

4.9.2. Load adjustment policy (LAP)

The load adjustment policy for a site \( s_i \) tries to continuously reduce load difference among the site \( s_i \) and its neighbors \( NSet_i \) by migrating jobs from heavily loaded sites to lightly loaded neighboring sites. The load adjustment policy is triggered whenever a site \( s_i \) receives load information from a neighbor. The load balancing algorithm will use the most recent load status information to decide whether a migration is initiated. The job that benefits most in the global job queue \( \text{GJQ}(s_i) \) is considered first for migration. If there are two neighboring sites that give the same performance benefit, the faster neighboring site is chosen. The load adjustment policy algorithm for a site \( s_i \) is described in Fig. 7.
5. Experiments

We only consider sender-initiated algorithms. In the simulation, our algorithm (labeled as DLB) is compared with the following algorithms in the simulation.

- Local. All jobs are locally processed by their originating sites.
- Random. A site is selected at random to process the arriving job.
- Best-neighbor load balancing Algorithm (labeled as BN). This is a modified sender-initiated diffusion (SID) approach because the traditional SID approaches are not applicable to the heterogeneous grid system as they do not take the job execution cost into account. A new job is sent to local site or one of neighbor sites immediately (IDP). Then continuous load adjustment is employed among neighbor sites (LAP). A job is assigned or redistributed to the site that it would benefit most. State information exchange is done via mutual information feedback among neighbor sites. We select the algorithm because it represents a typical class of decentralized approaches and bears similarity to our work.

5.1. Simulation model

In this section, we study the performance of the algorithms under different system parameters via simulations. Several assumptions were devised for the simulation model. These are:

- Jobs arrive at each site $s_i$, $i = 1, 2, \ldots, n$, according to a Poisson process with rate $\lambda_i = \lambda \times P_i$, where $P_i = 1/n$. The actual inter arrival time of jobs is adjusted to give the required overall average system loading (see below).
- The expected execution times of jobs are assumed to follow a two-phase hyper-exponential distribution [36] with a mean of $X$ time unit and a coefficient of variation (CV).
- The transfer delay that may be incurred between any site pairs in the grid system is chosen from a lognormal distribution with a mean of $\tau$ time unit and a standard deviation $\sigma_c$.
- To simplify the model and simulation, the transmission time for a job is assumed to be subject to the same time as the transfer delay.
- Let $\rho$ be the required average system load for our simulation, which is the average job arrival rate divided by the average job processing rate. Using this definition, we adjust the job mean inter-arrival time $1/\lambda$ needed to get the desired $\rho$.

For each simulation run, the simulation time is set to 1300 time units, during which, the first 300 time units are considered as “warm-up time.” After the warm-up time, we trace the jobs’ creation time, processing time and completion time. Thus, the observation period $[t_1, t_2]$ is $[301, 1300]$. We carry out each measurement three times with different random seeds. Table 1 displays the default values of the parameters used in the simulations. Except for experiment S2 and S3, all the other experiments use the highly heterogeneous system configuration shown in Table 2. In Table 2, we present a highly heterogeneous system configuration with four different processing powers. They are 1, 2, 5, and 10, respectively. The first row contains the APW of each of the four site types. The second row contains the number of sites in the system corresponding to each site type. We consider only situation where the fastest sites have at most 10 times higher relative processing power than the slowest site, because this is the case in most of the current heterogeneous grid systems.

5.2. Effect of system with high heterogeneity

In experiment S1, we carry out a series of simulations with the algorithms described above for a system that has a relatively high heterogeneity, under different system utilization parameter, $\rho$. We vary the system load by varying the mean inter-arrival time (initiation time) of the jobs, $1/\lambda$. Results are shown in Fig. 8.

As shown in Fig. 8, the higher the load, the higher the mean response time of any of the algorithms. We can conclude from observing Fig. 8 that the DLB algorithm behaves best in a highly heterogeneous system. DLB gives the minimum average response time across all values of $\rho$. At light loads (from 10% to 20%), the Random algorithm performs better than BN. The reason for that is Random algorithm selects candidate sites from a larger pool than BN
algorithm, i.e. it is not restricted to the sites in the neighbor sites. Thus it has higher chance to identify some sites with better processing power in this heterogeneous system. When the system loading becomes high, the difference between the mean response time of BN and Random algorithm increases. DLB has an average improvement factor of 27.2, 44 and 63.1% over BN, Random and Local, respectively. When the system loading is light or moderate, for algorithm DLB, IDP plays a crucial role and the LAP makes little influence on the mean response time of the jobs. The BN algorithm migrates jobs to an idle neighboring site, which can be much slower in a highly heterogeneous system than a faster non-neighboring site that has only a small amount of jobs in the queue (or that is currently processing a job and has an empty queue). At high system load, DLB first dispatches new jobs to faster partners with minimum load, which means that it is more likely for jobs to shorten mean response time at faster sites. Hence, DLB gives a better performance.

5.3. Effect of system with low heterogeneity

In experiment S2, we focus our analysis on the case where the system is much less heterogeneous. We consider a low heterogeneous system consisting of two site classes with class 1 APW of 1 and class 2 APW of 3. The processing power of the fastest site is only 3 times as high as the processing power of the slowest site. We divided the 32 sites equally between the two site classes. By observing the average response time of the algorithms, as shown in Fig. 9, we conclude that the response time of the system that results from applying the DLB is lower than the response time of the algorithm BN under all loads, but the difference is not significant. DLB has an average improvement factor of 8.8, 30.7 and 39.1% over BN, Random and Local, respectively. The conclusion is due to the fact that the IDP makes main contribution at low or medium system loading.

Table 1
Simulation parameters (tu = time unit, pt = percent)

<table>
<thead>
<tr>
<th>Simulation parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of system, n</td>
<td>32</td>
</tr>
<tr>
<td>Mean service time of jobs, X</td>
<td>1.0 tu</td>
</tr>
<tr>
<td>CV</td>
<td>4</td>
</tr>
<tr>
<td>Mean transfer delay, τ</td>
<td>0.05 tu</td>
</tr>
<tr>
<td>Average system utilization, ρ</td>
<td>80 pt</td>
</tr>
<tr>
<td>Standard deviation of transfer delay, σc</td>
<td>50 pt</td>
</tr>
<tr>
<td>Period for periodic information exchange, Tp</td>
<td>10 tu</td>
</tr>
<tr>
<td>Number of partners, k</td>
<td>4</td>
</tr>
<tr>
<td>Number of random partners/neighbors for information update, ωP</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2
Highly heterogeneous system configuration

<table>
<thead>
<tr>
<th>Relative processing power</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sites</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Fig. 8. Average response time of the algorithms in experiment S1.
5.4. Effect of system size

In experiment S3, we focus our analysis on the case where the number of sites in the grid system is varied. We consider a highly heterogeneous system consisting of two site classes with class 1 APW of 1 and class 2 APW of 10. The processing power of the fastest site is 10 times as high as the processing power of the slowest site. We divided the sites equally between the two site classes. By observing the average response time of the algorithms when the number of sites increases from 8 to 48, as shown in Fig. 10, we conclude that the response time of the system that results from applying the DLB is lower than the response time of the other algorithms under all \( n \). DLB has an average improvement factor of 57.6, 35.9 and 13.6% over Local, Random and BN, respectively.

5.5. Effect of communication delay

In experiment S4, we vary the mean network transfer delay \( \tau \), from 0.05 to 0.3 time units. The results shown in Fig. 11 illustrate the following points.

DLB consistently gives the best performance across all the values of \( \tau \). DLB gives an average improvement of 46.2 and 19.4% over Random and BN, respectively. The performance is especially apparent when the mean transfer delay is high. Besides Local algorithm that has no network traffic, the average response time of all the other algorithms increases when the mean transfer delay increases. But the increasing rate of DLB and BN is much smaller than that of Random. We suggest that this is because DLB and BN take the communication delay into account when making a load distribution decision and use the Mutual Information Feedback policy for information collection between a site and its neighbors or partners. Another possible reason is that each site receives jobs more frequently at high system load, which means each site transfers jobs more frequently, and in turn receives load information more frequently and accurately.
5.6. Effect with different job arrival pattern

All the experiment results discussed in the previous experiments are generated under the assumption that all sites have the same job arrival rate. In reality, job arrival rates usually differ from one site to another. To evaluate the effect of different job arrival rate on the average response time, we have conducted another experiment S5, in which we randomly choose ten of the sites as lightly-loaded sites ($\rho = 0.3$), eleven of the sites as moderately-loaded sites ($\rho = 0.6$) and eleven of the sites as highly-loaded sites ($\rho = 0.9$). It can be observed from Fig. 12 that the average response time using the DLB algorithm has an average improvement of 70, 51.1 and 19.1% over Local, Random and BN, respectively.
5.7. Effect of partners with different \( k \) value

In experiment S6, we vary the number of partners of a site while keeping the system loading at a constant \( \rho = 0.9 \). The results shown in Fig. 13 illustrate the following points. The average response time goes down as the number of partner increases. However, the improvements come at a decreasing rate. In moving from a size of 4 to a size of 8, the benefits of load balancing are very few or do not exist, and there seems to be a saturation point. This suggests that small-sized partners can be more effective with respect to load balancing.

6. Conclusions and future work

The computing grid is a new type of distributed computing that involves heterogeneous grid sites from different organizations on the Internet. Due to the issues of scalability, site heterogeneity, and significant communication overheads, these characteristics make grid systems different from the traditional distributed systems and have a significant impact on the performance of load balancing. In this paper, we have proposed a dynamic decentralized load balancing algorithm to cater for these characteristics. Through simulation, it was observed that our algorithm can give a shorter average job response time than the Local, Random and Best Neighbor Load Balancing algorithm over a wide range of system parameters.

Our research in this area is still at an early stage and there are many aspects worthy of further study. First, we have not modeled the impacts of accuracy of job execution time estimation on the effectiveness of our proposed load balancing algorithm. Second, we will utilize migration threshold dynamically based on real-time observation of load behavior of system resources. Finally, we do not take network and hardware failure into account in this study. A failure model may be employed to study this influence. Owing to the dynamic nature of the practical grid environment, designing an ideal load balancing algorithm still remains a challenge.

References


