

# Multidomain Load Balancing\*

Samuel T. Chanson, Wantao Deng, Chi-Chung Hui, Xueyan Tang, and Ming Yan To  
Department of Computer Science  
Hong Kong University of Science and Technology  
Clear Water Bay, Hong Kong  
E-mail: {chanson, dwt, cchui, tangxy, myto}@cs.ust.hk

## Abstract

*This paper investigates dynamic load balancing issues in the multidomain environment where local area networks (LANs) are interconnected by the Internet. Because of the much slower Internet communication speed and limited bandwidth, existing load balancing algorithms for LANs are unsuitable for the multidomain environment. New issues such as lag time in updating load information and network cost of transferring jobs must be addressed. To tackle these problems, the conventional least load scheduler is extended to the multidomain environment by employing a hierarchical structure, and several quick update techniques are proposed. Also, a heuristic taking both the machine load and the network cost into consideration is developed to evaluate the benefits of sending jobs to computers in different domains. A set of experiments conducted on the BALANCE testbed showed that the proposed techniques provide significant performance improvement over existing algorithms.*

## 1 Introduction

Computers nowadays continue to get cheaper and faster. The development of high speed networks (e.g., Gigabit Ethernet) further reduces the cost of communication between computers. Moreover, the rapid growth of the Internet makes it possible for computer networks to cover a large geographical area. Driven by these technologies, parallel and distributed computing over networks of computers are becoming increasingly popular.

Contemporary computing facilities of large organizations often consist of personal computers and workstations connected by one or more local area networks (LAN) which are in turn interconnected via the Internet. This type of environment is known as the multidomain environment and the computers within a LAN constitute a domain. The intranets

owned by large corporations are examples of the multidomain environment. Due to uneven job arrival patterns, time-zone differences and possibly unequal computing capacities of the individual computers, computers in one domain may be overloaded while others in a different domain are underutilized. Therefore, it is desirable to dispatch jobs to idle or lightly loaded computers in the multidomain environment to achieve better resource utilization and reduce job completion time. This is a natural extension of the existing work on load balancing in a single LAN environment [9, 10, 12, 15].

The structure of the multidomain environment is much more complex than that of a single LAN. Due to the much slower communication speed and limited bandwidth of most Internet links, longer delays are expected in disseminating load updates. Moreover, unlike a single LAN, different LANs often do not use a globally shared file system in practice. Scheduling jobs to remote domains requires the transfer of related programs and data files. These factors make existing load balancing techniques for LANs unsuitable for the multidomain environment. The objective of this paper is to develop effective multidomain load balancing algorithms. We extended the conventional least load scheduler to the multidomain environment by employing a hierarchical architecture. Several techniques are proposed to tackle the new problems in multidomain scheduling. These include quick update of load information and maintaining multiple least loaded computers for each remote domain. In addition, a heuristic is developed to estimate the benefits of executing jobs on computers in different domains. It takes both the machine load and the network cost into consideration. We have evaluated these techniques on an experimental testbed and the results showed that the proposed techniques effectively reduce errors in job scheduling and outperform existing algorithms.

The rest of the paper is organized as follows. Previous work on multidomain load balancing is outlined in Section 2. Section 3 presents the hierarchical multidomain scheduling model. Sections 4 and 5 describe new techniques to address load update delay and network cost respectively. The

\*This work was partially supported by a grant from the Research Grant Council of Hong Kong (Grant No. HKUST6080/97E).

experimental environment and performance results are presented in Sections 6 and 7 respectively. Finally, Section 8 concludes the paper.

## 2 Related Work

Previous work on multidomain load balancing has concentrated on augmenting existing algorithms for LANs with hierarchical structures to achieve scalability.

Epema *et al.* [3] extended the *Condor* load sharing facility [13] to work in the wide area network (WAN) environment, where a central manager and a gateway machine are placed in each Condor pool. The gateway machine maintains a list of idle computers in the local pool and exchanges the list with other gateways. It periodically presents a randomly selected computer from the idle lists to the central manager for computing the scheduling decision. Since only one computer is selected at a time, resource sharing between pools may not be maximized. Lu *et al.* [14] suggested to organize computers into a cluster hierarchy based on the communication costs among clusters. The manager in each cluster explores the spare capacity of its neighboring clusters from the bottom up to the top layer in the hierarchy. The algorithm takes speed heterogeneity into account and uses a threshold to classify the clusters as either saturated or idle. Jobs are only scheduled to remote domains if they are considered idle. Hui *et al.* [5] proposed a two-level load balancing scheme for interconnected LANs. It uses group communication at the intra-LAN level to improve efficiency and adopts the hydrodynamic model [7] at the inter-LAN level with point-to-point connections. In this scheme, the gateway computer represents the LAN where it resides in interacting with other gateways to provide global load balancing, and acts as an ordinary computer for decentralized load sharing within the LAN. Large networks of computers are partitioned into small groups in the strategies proposed by Evans *et al.* [4]. Besides geographical locations, the loadings of computers are also suggested to be the partition criteria. Membership of computer groups are periodically adjusted based on current machine loads to reduce the load difference among the groups. The drawback of this scheme in the multidomain environment is that the computers grouped together may be located at large physical distances from one another, thus introducing high overheads in disseminating load updates and transferring tasks at the intra-group level.

Most of the work mentioned above ignore practical issues such as load update delays and file transfer costs which are critical to the performance of multidomain systems. We have focused on these issues in our study and some solutions are presented in this paper to enhance multidomain scheduling performance.

## 3 Hierarchical Scheduling Model

A common way in addressing the scalability problem in large computer networks is to adopt a hierarchical approach. It can be used in load balancing in the multidomain environment as shown in Figure 1. A gateway computer is designated in each domain to handle inter-domain communication. It collects load information of computers in the local domain and exchanges the information with other gateways. The gateway computers are also responsible for sending jobs to and receiving jobs from remote domains. Non-gateway computers only communicate with the gateway in their domain and do not directly contact computers outside the local LAN for load balancing purposes.

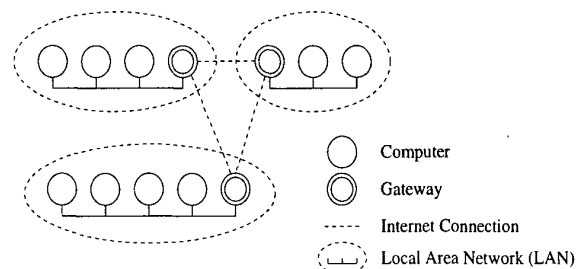


Figure 1. The Multidomain Environment

Load control at the inter-domain level can be either centralized or distributed. The advantage of making decisions centrally is simplicity because load information need not be replicated at different locations. However, the central scheduler may become a performance bottleneck as the system size increases. Moreover, for a large scale system that is geographically distributed, consulting the central scheduler incurs non-negligible overhead and network delay. The centralized scheme also creates fault-tolerance problems due to single point of failure. Therefore, a distributed control mechanism is used at the inter-domain level in our proposal. On the other hand, since computers in the same LAN are typically connected by a high speed network, it is feasible to apply a centralized control scheme at the intra-domain level to improve efficiency. Therefore, our model uses distributed control at the inter-domain level, and centralized control within a LAN (intra-domain level).

In our scheduling model, a job scheduler (called gateway scheduler) is associated with each gateway computer. All jobs submitted in the local domain are scheduled by the gateway scheduler. Every gateway  $g$ , executes the same algorithm BASIC-MDS (see Algorithm 1). The data structure *loadlist* is an ordered set that stores the information of computers in the same domain in increasing order of their loads. The information for each computer has the format  $info(c) = \langle c, speed(c), load(c) \rangle$ , where  $c$  is the name of the computer,  $speed(c)$  and  $load(c)$  are the relative speed

of  $c$  and the load on  $c$  respectively. In BASIC-MDS, the *loadlist* sent to a remote domain only includes the information of the least loaded computer in the sender's domain in order to simplify computation and conserve network bandwidth. The run queue length of  $c$  is used to characterize *load(c)* as a previous study suggests that this metric best describes the load of a computer [11]. To handle the differences in computing capacities, this load metric is normalized by the computer's speed in our scheduling model. Specifically, the load of a computer is defined as

$$load = \frac{run\ queue\ length + 1}{speed}.$$

This metric approximates the effective computing capacity available to a new job if it is scheduled to run on the computer (hence the "+1" in the numerator). Note that the projected load is always positive (greater than zero).

**Algorithm 1** BASIC-MDS ( $i$ )

$g_i$ : gateway computer where algorithm is executed.  
 $d_i$ : domain where  $g_i$  resides.

1. For each domain  $d_j$ , let  $loadlist_j = \phi$ .
2. Let  $loadlist_i = \bigcup_{c \in d_i} \{info(c) : info(c) = \langle c, speed(c), \frac{1}{speed(c)} \rangle\}$ .
3. Sort all  $info(c) \in loadlist_i$  in increasing order of  $info(c).load$ .
4. Send message  $\langle$  "update-remote-load",  $loadlist_i[1]$   $\rangle$  to gateway  $g_j, \forall g_j \neq g_i$ .
5. Receive an incoming message  $M$ .
6. If  $M.t =$  "local-job-arrival"
  - (6.a) Find  $c_{least}$  such that  $info(c_{least}).load = \min\{info(c).load : info(c) \in \bigcup_j loadlist_j\}$ .
  - (6.b) If  $c_{least} \in d_i$  then
    - (6.b.1) Send  $M.job$  to run on computer  $c_{least}$ .
  - (6.c) If  $c_{least} \notin d_i$  then
    - (6.c.1) Send message  $\langle$  "remote-job-arrival",  $M.job$   $\rangle$  to gateway  $g_j$ , where  $c_{least} \in d_j$ .
7. If  $M.t =$  "remote-job-arrival"
  - (7.a) Find  $c_{least}$  such that  $info(c_{least}).load = \min\{info(c).load : info(c) \in loadlist_i\}$ .
  - (7.b) Send  $M.job$  to run on computer  $c_{least}$ .
8. If  $M.t =$  "update-remote-load"
  - (8.a) Let  $loadlist_j = \{M.info(c)\}$ , where  $c \in d_j$ .
9. If  $M.t =$  "update-local-load"
  - (9.a) Let  $loadlist_i = loadlist_i \cup \{M.info(c)\}$ .
  - (9.b) Sort all  $info(c) \in loadlist_i$  in increasing order of  $info(c).load$ .

(9.c) If  $loadlist_i[1]$  changes during sorting

(9.c.1) Send message  $\langle$  "update-remote-load",  $loadlist_i[1]$   $\rangle$  to gateway  $g_j, \forall g_j \neq g_i$ .

10. Goto step 5.

After initialization (steps 1 to 3),  $g_i$  broadcasts its state (i.e., the least loaded computer in its domain) to other gateways and starts processing incoming messages in the main loop. There are two classes of messages in the algorithm: "load update" messages and "job arrival" messages. A load update message  $M$  has the format  $\langle t, info(c) \rangle$  where  $t$  is the message type and  $info(c)$  contains the information of a computer  $c$ . When the state of a computer in domain  $d_i$  changes, an "update-local-load" message is sent to gateway  $g_i$  with  $M.info$  containing the updated information of the computer. On receiving the updated information,  $g_i$  sorts all computers in the local domain (i.e.,  $loadlist_i$ ) in increasing order of their loads (step 9.b). If the least loaded computer in  $loadlist_i$  (or its load) changes after sorting,  $g_i$  sends an "update-remote-load" message to the other gateways and stores the new information of the least loaded computer in  $M.info$  (step 9.c). A type  $t$  job arrival message  $M$  has the format  $\langle t, job \rangle$  where  $job$  denotes the job to be scheduled. A "local-job-arrival" message is sent to gateway  $g_i$  when a new job is submitted in domain  $d_i$ . The job is directly sent to run on the least loaded computer  $c_{least}$  in the network if  $c_{least}$  belongs to  $d_i$  (steps 6.a and 6.b). Otherwise, the job is sent to the gateway in  $c_{least}$ 's domain by a "remote-job-arrival" message (step 6.c). After receiving the message, the remote gateway may reallocate the job if the least loaded computer in its domain has changed (step 7). Note that the job is not allowed to be rescheduled to another domain to avoid transferring the job between domains indefinitely.

BASIC-MDS does not consider lag time of load updates and network cost of file transfers. In the next two sections, we propose several new techniques to enhance the performance of BASIC-MDS.

## 4 Load Update Delay

The delay in load information update is an important issue in multidomain load balancing. The lag time in disseminating load information is typically much larger than that in a LAN and can vary greatly due to network congestion. Hence it cannot be ignored. It is possible that the scheduler has not received the updated load information from a remote domain when a new job arrives. As a result, a lightly loaded computer can quickly become overloaded because all the schedulers send jobs to it before the new load information is available. This is known as the *herd effect* [2] and it often leads to incorrect job scheduling and poor performance.

Solutions to the herd effect are not straightforward. Some obvious solutions such as not assigning any more jobs

to the same computer until the updated load of that computer arrives do not work well as shown in our simulation experiments. The reason is two folded. First, it is possible that the remote computer remains to be the least loaded machine after receiving the new job. Second, if the scheduler has a large list of jobs waiting to be scheduled, after dispatching a job to each remote domain, the rest of the jobs must wait until the next load update message arrives. In both cases, there are unnecessary wait times.

In the rest of this section, we describe two techniques that reduce incorrect job scheduling due to load update delay. Their effectiveness is verified by the experimental results given in Section 7.

#### 4.1 Quick Load Update Technique

The strategy proposed by Hui *et al.* [8] takes into consideration the load update delay, but works only in the centralized scheduling environment. It suggests that the scheduler keeps track of the number of jobs currently allocated to each computer and updates the load of the destination computer immediately after a scheduling decision is made. An improved technique which we call quick load update technique has been developed to allow efficient scheduling in the multidomain environment.

The main idea of quick load update is for the scheduler to refresh the load information of the computers as early as possible. Consider the situation where the gateway  $G_A$  of domain  $A$  allocates a job to the least loaded computer  $c$  in the same domain (see Figure 2). As soon as the scheduling decision is made but before the job is dispatched,  $G_A$  can immediately update its database by adding  $\frac{1}{speed(c)}$  to  $load(c)$  (i.e., increasing the run queue length of  $c$  by 1) to approximate the new load of  $c$  (step 3). Notice that after the update, either  $Least_A$  or  $load(Least_A)$  is changed, where  $Least_A$  denotes the least loaded computer in domain  $A$ . Therefore,  $G_A$  needs to broadcast the new least load information to gateways in the other domains (step 4). This can also be done before  $G_A$  actually sends the job to  $c$ . The quick update technique enables gateway machines to refresh the load information before the load monitor on  $c$  detects a load change. The gateways use this "updated" load information until the new load index reported by  $c$  is received via  $G_A$  (steps 7 and 8).

#### 4.2 Keeping Multiple Least Loaded Computers

If the gateway scheduler has up-to-date information instantaneously, it is sufficient to maintain only the least loaded machine of each remote domain for scheduling purposes. However, due to the slow inter-domain connections, what the scheduler believes to be the least loaded machine may be obsolete even with the quick update technique. As a result, the scheduler may make errors in scheduling jobs,

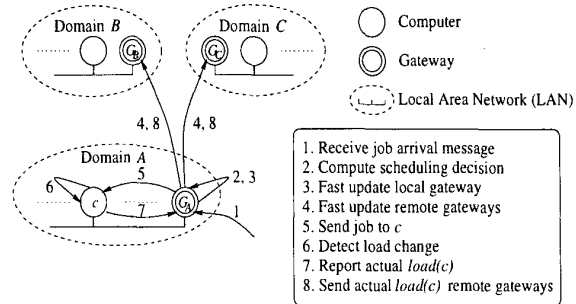


Figure 2. Quick Load Update Technique

especially when the number of jobs waiting to be scheduled is large. We suggest to let the gateway scheduler maintain multiple (say  $n$ ) least loaded computers for each remote domain instead of only one. This gives the scheduler a better picture of the load status in the remote domains. Once the scheduler allocates a job to the least loaded computer of a remote domain, it updates the load index of that computer by adding 1 to its run queue length and sorts the loads of the  $n$  computers for that domain in its database. This technique further shortens the delay in disseminating load updates.

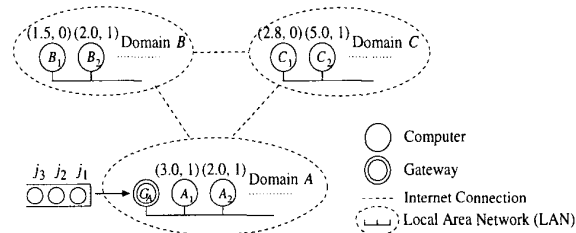


Figure 3. Example of Keeping Two Least Loaded Computers

Consider the example in Figure 3, where a batch of 3 jobs ( $j_1, j_2$  and  $j_3$ ) are submitted in domain  $A$ , and  $n = 2$ .  $X_1$  and  $X_2$  denote the least loaded and the second least loaded computers in domain  $X$  respectively ( $X = A, B, C$ ). A 2-tuple  $(s, q)$  is associated with each computer, where  $s$  is the relative speed and  $q$  is the current run queue length of the computer. According to the load metric formula  $\frac{run\ queue\ length + 1}{speed}$ , the gateway  $G_A$  will allocate  $j_1$  to  $C_1$  ( $load(C_1) = \frac{1}{2.8} = 0.36$  is the smallest). Suppose  $G_A$  only keeps one least loaded machine for domains  $B$  and  $C$  respectively. Since the new load of  $C_1$  (i.e.,  $\frac{2}{2.8} = 0.71$ ) is higher than those of  $A_1$  and  $B_1$  ( $load(A_1) = \frac{2}{3.0} = 0.67$ ,  $load(B_1) = \frac{1}{1.5} = 0.67$ ),  $j_2$  and  $j_3$  will be sent to  $A_1$  and  $B_1$  respectively. On the other hand, if  $G_A$  has information of the two least loaded machines in each remote domain,  $j_2$  and  $j_3$  will be scheduled to run on  $C_2$  because its loading is lighter than those of  $A_1$  and  $B_1$  ( $load(C_2) = \frac{2}{5.0} = 0.4$ ,

and after receiving  $j_2$ ,  $load(C_2) = \frac{3}{5.0} = 0.6$ ). The overall performance will improve as a consequence.

It is intuitive that the larger the number of least loaded computers kept by the scheduler, the better the performance. Experiments were performed on a discrete event simulator for a range of system configurations and system loading is kept near the saturation point (since load balancing is not useful when the workload is light). The results showed that keeping more than two least loaded computers did not further improve system performance substantially [1]. The reason is that in our scheduling model, jobs sent to a remote domain may be reallocated by the gateway there based on more recent load information (step 7 of BASIC-MDS in Section 3). Hence, aggressively using more least load information of remote domains does not enhance performance greatly. Furthermore, system overhead increases with the amount of least load information maintained. Therefore, we propose to maintain two least loaded computers of each remote domain at the gateways.

### 4.3 Formal Description of the Algorithm

The improved scheduling algorithm MDS is presented more formally in Algorithm 2. It integrates the above techniques into the BASIC-MDS algorithm. Different from BASIC-MDS, an “update-remote-load” message in MDS has the format  $\langle t, info(c_1), info(c_2) \rangle$  where  $info(c_1)$  and  $info(c_2)$  store the load information of the two least loaded computers respectively (steps 4, 6.d.1, 7.d and 9.c.1). Moreover, on receiving an “update-local-load” message, new least load information is sent to remote gateways when either one of the first two items in  $loadlist_i$  changes after sorting (step 9.c). The quick load update technique is used in steps 6.b, 6.c and 6.d.1 for “local-job-arrival” messages, and in steps 7.b to 7.d for “remote-job-arrival” messages. Steps 6.b and 6.c also serve the purpose of refreshing  $loadlist_j$  when a job is scheduled to a remote domain  $d_j$ .

#### Algorithm 2 MDS ( $i$ )

$g_i$ : The gateway computer where the algorithm is executed.  
 $d_i$ : The domain where  $g_i$  resides.

1. For each domain  $d_j$ , let  $loadlist_j = \phi$ .
2. Let  $loadlist_i = \bigcup_{c \in d_i} \{info(c) : info(c) = \langle c, speed(c), \frac{1}{speed(c)} \rangle\}$ .
3. Sort all  $info(c) \in loadlist_i$  in increasing order of  $info(c).load$ .
4. Send message  $\langle$  “update-remote-load”,  $loadlist_i[1], loadlist_i[2] \rangle$  to gateway  $g_j, \forall g_j \neq g_i$ .
5. Receive the first incoming message  $M$ .
6. If  $M.t =$  “local-job-arrival”
  - (6.a) Find  $c_{least}$  such that  $info(c_{least}).load = \min\{info(c).load : info(c) \in \bigcup_j loadlist_j\}$ .

$$(6.b) \text{ Let } info(c_{least}).load = info(c_{least}).load + \frac{1}{info(c_{least}).speed}.$$

(6.c) Sort all  $info(c) \in loadlist_j$  in increasing order of  $info(c).load$ , where  $c_{least} \in d_j$ .

(6.d) If  $c_{least} \in d_i$  then

(6.d.1) Send message  $\langle$  “update-remote-load”,  $loadlist_i[1], loadlist_i[2] \rangle$  to gateway  $g_j, \forall g_j \neq g_i$ .

(6.d.2) Send  $M.job$  to run on computer  $c_{least}$ .

(6.e) If  $c_{least} \notin d_i$  then

(6.e.1) Send message  $\langle$  “remote-job-arrival”,  $M.job \rangle$  to  $g_j$ , where  $c_{least} \in d_j$ .

7. If  $M.t =$  “remote-job-arrival”

(7.a) Find  $c_{least}$  such that  $info(c_{least}).load = \min\{info(c).load : info(c) \in loadlist_i\}$ .

$$(7.b) \text{ Let } info(c_{least}).load = info(c_{least}).load + \frac{1}{info(c_{least}).speed}.$$

(7.c) Sort all  $info(c) \in loadlist_i$  in increasing order of  $info(c).load$ .

(7.d) Send message  $\langle$  “update-remote-load”,  $loadlist_i[1], loadlist_i[2] \rangle$  to gateway  $g_j, \forall g_j \neq g_i$ .

(7.e) Send  $M.job$  to run on computer  $c_{least}$ .

8. If  $M.t =$  “update-remote-load”

(8.a) Let  $loadlist_j = \{M.info(c_1), M.info(c_2)\}$ , where  $c_1, c_2 \in d_j$ .

9. If  $M.t =$  “update-local-load”

(9.a) Let  $loadlist_i = loadlist_i \cup \{M.info(c)\}$ .

(9.b) Sort all  $info(c) \in loadlist_i$  in increasing order of  $info(c).load$ .

(9.c) If  $loadlist_i[1]$  or  $loadlist_i[2]$  changes during sorting

(9.c.1) Send message  $\langle$  “update-remote-load”,  $loadlist_i[1], loadlist_i[2] \rangle$  to  $g_j, \forall g_j \neq g_i$ .

10. Goto step 5.

## 5 Network Cost

Since computers in a local area network usually use the same file system and there are dedicated file servers, program and data files do not have to be transferred when a job is scheduled to run on a different computer in the same LAN (only a command line is sent). Hence, the network cost of remote job execution can be ignored in the single LAN environment. However, in the multidomain environment, the related files of a job need to be transferred through much slower Internet links if the job is scheduled to run in a remote domain. Therefore, the cost of file transfers must be taken into consideration in the scheduling algorithm. For example, when two computers have similar loadings, the

one connected to the local gateway with faster communication link should be selected. Experimental results have shown that even though the network cost may be small compared to the job execution time, mean response ratio and fairness (see Section 6.4 for definitions) can be substantially improved if network cost is taken into account.

Consider the situation where a job  $J$  submitted in domain  $A$  is scheduled to run on computer  $c$  in a remote domain  $B$ . The completion time of job  $J$  is approximated by the following formula:

$$\frac{run-time(J)(run-queue-length(c) + 1)}{speed(c)} + \frac{job-size(J)}{link-speed(A, B)}, \quad (1)$$

where the meanings of the variables are listed in Table 1.

$run-time(J)$	execution time of job $J$ when it runs by itself on a computer with relative speed 1
$job-size(J)$	total size of job $J$ (program, data and system information) that has to be transferred to run on a remote computer
$run-queue-length(c)$	current run queue length of computer $c$
$speed(c)$	relative speed of computer $c$
$link-speed(A, B)$	speed of the communication link between domains $A$ and $B$

**Table 1. The Meaning of Variables in Formula (1)**

The first term in formula (1) estimates the elapsed time of job  $J$  on computer  $c$  under the processor sharing discipline and the second term estimates the job transfer time. The job should be scheduled to the computer with the minimum expected completion time. Unfortunately, the run time of a job is usually not known when the job arrives. Even if the job size is known, it does not help much in making the scheduling decision without knowledge of the job run time. Notice that  $run-time(J)$  does not depend on the speed of the computer (see Table 1 for definition). Therefore, formula (1) can be divided by  $run-time(J)$  without affecting the scheduling decision:

$$\frac{run-queue-length(c) + 1}{speed(c)} + \frac{job-size(J)}{run-time(J)} \cdot \frac{1}{link-speed(A, B)}. \quad (2)$$

The value computed by the above formula is referred to as the relative completion time of the job.  $\frac{job-size(J)}{run-time(J)}$  is the only unknown term in (2) and we propose to use a constant  $C$  to replace it. Thus, formula (2) becomes:

$$\frac{run-queue-length(c) + 1}{speed(c)} + C \cdot \frac{1}{link-speed(A, B)}. \quad (3)$$

This is a heuristic load metric that takes link speed into consideration. It can be used to evaluate the benefits of sending jobs to computers in different domains. The computer that provides the smallest value according to formula (3) should be selected. The constant  $C$  is selected such that  $C \cdot \frac{1}{link-speed(A, B)}$  is much smaller (e.g., one or two orders of magnitude lower<sup>1</sup>) than  $\frac{run-queue-length(c)+1}{speed(c)}$ . The actual value of  $C$  within this range is not very critical to the performance. This is because by adding the network cost term  $C \cdot \frac{1}{link-speed(A, B)}$  in the load metric, a job has higher probability to be scheduled to domains that introduce less (or no) network costs. Therefore, for short jobs where network cost is comparable to run time, the completion time using this load metric would decrease significantly. On the other hand, for long jobs where network cost is negligible compared to run time, the completion time would not be affected very much. As a result, the overall system performance, especially mean response ratio and fairness (see Section 6.4 for definitions), is improved (see experimental results in Section 7).

Either formula (2) or (3) can be augmented on top of the MDS scheduler (in step (6.a)). Notice that since no network costs are incurred if a job is scheduled to run in the local domain, the second item in formula (2) or (3) (i.e., the file transfer costs) is set to 0 for local computers. MDS augmented with formulas (2) and (3) are referred to as the MDS-NET and the MDS-C algorithms respectively.

## 6 Experimental Environment

### 6.1 The BALANCE Testbed

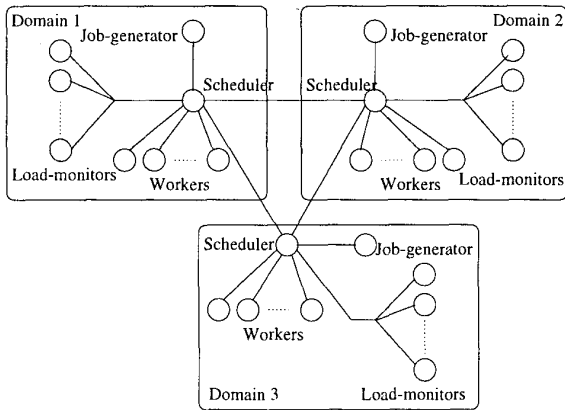
The experiments were conducted on the BALANCE testbed [6]. BALANCE is a flexible load balancing system that provides efficient remote execution facilities and allows the user to implement his/her new system schedulers. Currently, it runs on Solaris, SunOS and the Windows NT environment. The gateway scheduler is implemented by a daemon process `balgateway` running on the gateway computer.

### 6.2 System Architecture

The experimental environment consists of eleven computers running at different speeds. They are grouped into three domains and every inter-domain communication link is configured to run at a specified *speed*. One of the computers in each domain is designated as the domain gateway. The software architecture includes a set of communicating processes shown in Figure 4.

A *Job-generator* is placed in each domain to generate jobs which are sent to the *Scheduler* of the local gateway.

<sup>1</sup>Load balancing in the Internet environment is beneficial mainly for jobs with large computational requirements.



**Figure 4. Schematic Diagram of Software Processes in the Experimental Environment**

The *Scheduler* assigns jobs to the computers as computed by the scheduling algorithm. Based on the computed decision, the *Scheduler* either sends the job to a remote gateway or spawns a *Worker* at the selected computer in the local domain to execute the job. A *Load-monitor* runs on every computer and sends the load information to the *Scheduler* when there is a change. The *Scheduler* computes the least load information in the domain and informs the other *Schedulers* whenever it is updated. Once started, the jobs run to completion on the assigned computers and are not rescheduled.

### 6.3 Workload Characteristics

The experiments are designed to test the performance of the load balancing algorithms over a wide range of workload characteristics. Each job is associated with two parameters: *run time* and *job size* (refer to Section 5 for definitions), which are characterized in terms of seconds and kilobytes respectively. In our experiments, the run time follows a segmented uniform distribution. For example, the distribution in Table 2 specifies that 20% of the jobs have run times  $U(0, 150)$  seconds, 60% of the jobs have run times  $U(150, 400)$  seconds and so on, where  $U(x, y)$  is a uniformly distributed number between  $x$  and  $y$ . The job size follows a discrete distribution. The example in Table 2 means that 40% of the jobs have sizes of 100KB, 40% of the jobs have sizes of 200KB and so on. The lag time of transferring a job between domains is calculated as follows. Assume  $L = S/B$  is the job transfer time when the inter-domain communication link has no other traffic, where  $S$  is the job size and  $B$  is the speed of the link. The job transfer delay is set to  $U(L, 3L)$  and  $U(3L, 5L)$  with probabilities of 90% and 10% respectively.

In order to investigate the performance of the schedul-

Run Time Distribution		Job Size Distribution	
range	percentage	size	percentage
0 – 150 sec	20%	100KB	40%
150 – 400 sec	60%	200KB	40%
400 – 600 sec	15%	1MB	20%
600 – 1500 sec	5%		

**Table 2. Example of Run Time and Job Size Distributions**

ing algorithms in balancing the workload among domains, one of the three domains is intentionally designed to have low aggregate processing speed and high job arrival rate. This domain is referred to as the *slow domain*. The ratio of the job arrival rates between the “slow domain” and the other domains is set to 3:1. A batch arrival pattern is assumed in our experiments. The batch size is set to be  $U(\text{minbatchsize}, \text{maxbatchsize})$ , where *minbatchsize* and *maxbatchsize* are the minimum and maximum number of jobs in a batch respectively. Unless mentioned explicitly, the average batch inter-arrival times are set such that the system utilization approaches 100%. A total of 250 jobs were generated in each run, of which 150 were generated in the “slow domain” and 50 were generated in each of the other domains. Each run typically takes several hours depending on the workload and all data shown in Section 7 were the average result of 2 to 3 independent runs.

### 6.4 Performance Metrics

The following three metrics were selected to evaluate the scheduling performance:

1. *Mean response time*: This is defined as the average completion time of all the jobs. It is a commonly used metric to evaluate scheduling performance.
2. *Mean response ratio*: The *response ratio*  $R$  of a job is defined by the formula  $R = E_j/E_1$ , where  $E_j$  is the actual completion time of the job measured in the experiment, and  $E_1$  is the job’s run time as defined in Section 5. Mean response ratio is defined as the average response ratio over all the jobs. This metric is more objective as the effect of job size on performance is eliminated. The smaller the mean response ratio, the better the performance.
3. *Fairness*: Fairness is defined as the standard deviation of  $R$ . This definition of fairness is reasonable as users are likely to expect short delays for small jobs but willing to tolerate longer delays for larger jobs. The smaller the fairness, the better the performance.
4. *Network Cost per Job*: This is defined as the average lag time introduced by file transfers over all the jobs.

It is used to compare MDS with algorithms that take network cost into consideration i.e., MDS-NET and MDS-C (see Sections 7.2 and 7.3).

## 7 Experimental Results

The performance results are presented in this section. It is found that both mean response time and mean response ratio are reduced on using the quick update technique. Furthermore, load metric taking network cost into account improves mean response ratio and fairness significantly. The proposed algorithm performs well over a wide range of system loads.

### 7.1 Effect of Load Update Delay

The proposed techniques for load update delay are evaluated using the system configuration listed in Table 3 and the workload characteristics shown in Table 2 (see Section 6.3). The batch size of jobs is uniformly distributed between 1 and 3. In order to assess the benefits of individual techniques separately, an additional scheduling algorithm QUICK-MDS which only employs the quick load update technique (but not the two least loaded machines) was also tested. The experimental results are summarized in Table 4.

System Configuration		
Domain	Number of Computers	Relative Speeds of Computers
$D_1$	2	3.0, 3.0
$D_2$	5	1.5, 1.5, 1.5, 1.5, 1.5
$D_3$ (slow)	4	1.2, 1.2, 1.2, 1.0
Link Speeds: $[D_1 - D_2]$ : 128Kbps $[D_1 - D_3]$ : 256Kbps $[D_2 - D_3]$ : 256Kbps		

Table 3. Experimental Setting 1

	BASIC -MDS	QUICK -MDS	MDS
Quick load update	×	√	√
Two least loaded computers	×	×	√
Mean response time (sec)	391	283	257
Mean response ratio	1.46	1.04	0.96
Fairness	0.83	0.64	0.60

Table 4. Performance for Experimental Setting 1

From Table 4, it can be seen that the load update delay is a critical factor affecting the performance of multidomain job scheduling. Algorithms taking this factor into consideration improve the system performance significantly. QUICK-MDS outperforms BASIC-MDS by nearly 30% in terms of both mean response time and mean response ratio. MDS further enhances the scheduling performance over

QUICK-MDS by almost 10%. These observations show that the proposed techniques effectively reduce the duration of load update delay so that gateway computers have more up-to-date information on system loadings thereby making fewer scheduling errors. A number of experiments for different system settings have been carried out and they all show the same performance trends [1].

### 7.2 Effect of Network Cost

In this section, we investigate the impact of file transfer costs on scheduling performance. The system configuration is listed in Table 5. Job size distributions and run time distributions are specified in Tables 6 and 7. The batch size is uniformly distributed in the range [1 – 3].

System Configuration		
Domain	Number of Computers	Relative Speeds of Computers
$D_1$	2	3.0, 3.0
$D_2$	6	1.5, 1.5, 1.5, 1.5, 1.5, 1.2
$D_3$ (slow)	3	1.2, 1.2, 1.0
Link Speeds: $[D_1 - D_2]$ : 1024Kbps $[D_1 - D_3]$ : 56Kbps $[D_2 - D_3]$ : 256Kbps		

Table 5. Experimental Setting 2

Small Job Size		Large Job Size	
size	percentage	size	percentage
100KB	40%	200KB	40%
200KB	40%	500KB	40%
1MB	20%	3MB	20%

Table 6. Job Size Distributions

Small Run Time		Medium Run Time	
range (sec)	percentage	range (sec)	percentage
10 – 60	20%	10 – 60	20%
60 – 600	20%	60 – 1200	20%
600 – 1800	35%	1200 – 3600	35%
1800 – 3600	20%	3600 – 7200	20%
3600 – 7200	5%	7200 – 21600	5%
Large Run Time			
range (sec)	percentage		
20 – 120	20%		
120 – 1800	20%		
1800 – 7200	35%		
7200 – 21600	20%		
21600 – 43200	5%		

Table 7. Run Time Distributions

Table 8 shows the experimental results of MDS and MDS-NET under different job size distributions. The small

Job Size Distribution	Small Job Size		Large Job Size	
Algorithm	MDS	MDS-NET	MDS	MDS-NET
Mean Response Time (sec)	1180	1088	1180	1190
Mean Response Ratio	1.36	0.97	3.16	1.02
Fairness	2.83	0.41	19.5	0.42
Network Cost per Job (sec)	34.4	28.7	94.0	55.0

**Table 8. Performance for Different Job Size Distributions**

Run Time Distribution	Small Run Time		Medium Run Time		Large Run Time	
Algorithm	MDS	MDS-NET	MDS	MDS-NET	MDS	MDS-NET
Mean Response Time (sec)	1180	1190	2530	2470	7060	7000
Mean Response Ratio	3.16	1.02	1.43	1.00	1.25	1.07
Fairness	19.5	0.42	2.14	0.39	1.02	0.43
Network Cost per Job (sec)	94.0	55.0	67.1	47.2	62.0	58.1

**Table 9. Performance for Different Run Time Distributions**

run time distribution (see Table 7) is used in this set of experiments.

Table 9 presents the performance results for different run time distributions. The large job size distribution (see Table 6) is used in this set of experiments.

As shown in Tables 8 and 9, MDS-NET reduces mean response ratios considerably compared to those of MDS. This is because MDS-NET assigns jobs to remote domains only when the reduction in job elapsed time on the designated computer outweighs the network cost of sending the job there. As a result, the average network cost decreases and more jobs are processed by local computers under MDS-NET than MDS. Table 10 shows the number of jobs that are generated and processed by the same domain for different job size distributions.

Algorithm	Small Job Size		Large Job Size	
	MDS	MDS-NET	MDS	MDS-NET
Domain 1	17	33	26	32
Domain 2	28	34	30	41
Domain 3	28	48	31	52

**Table 10. Examples of Numbers of Jobs Allocated to Local Domains (250 jobs per run)**

Another observation is that the higher the network cost, the greater the improvement of MDS-NET over MDS. In Table 8, the reduction of mean response ratio increases with increasing job size (28.7% for small job size and 67.7% for large job size). Table 9 shows that the reduction of mean response ratio is more pronounced when the jobs have smaller run times (14.4% for large run time, 30.1% for medium run time and 67.7% for small run time). This is consistent with formula (2) which estimates the relative job completion time. The scheduling performance is more sensitive to network cost if the average job size increases or the average

run time decreases (i.e., the ratio of  $\frac{job-size(J)}{run-time(J)}$  increases).

According to formula (2) in Section 5, network cost does not have much effect on scheduling decisions of jobs with large run times (i.e., when the network cost is negligible compared to the run time). In other words, jobs with small run times benefit most from MDS-NET. Although the response ratios of these jobs decreases significantly, the absolute values of the reductions in their response times are not large. Therefore, MDS-NET has little impact on mean response time over all jobs. This explains why MDS-NET show similar mean response times compared to MDS but much better mean response ratios and fairness in Tables 8 and 9. For example, MDS-NET improves mean response ratio by 30.1%, and fairness by 81.8% under medium run time and large job size distributions.

### 7.3 Performance of Network Cost Heuristic

The performance of using the heuristic formula (3) to predict scheduling benefits is evaluated in this subsection. The experimental settings include the system configuration shown in Table 11, the small job size distribution and the medium run time distribution (see Tables 6 and 7). The job batch size is uniformly distributed in the range [1 – 3]. The constant  $C$  is set at 5.6 and the link speeds are expressed in Kbps (e.g., for the setting in Table 11, the second term in formula (3) has the value of  $\frac{5.6}{56} = 0.1$  when  $D_2$  is estimating the completion time of executing jobs in  $D_3$ ).

Table 12 shows the performance results for system utilizations of 100% and 80%. Similar to MDS-NET, the MDS-C scheduler gives higher priorities to the local computers and those connected by fast Internet links. It gives better mean response ratios compared to those of MDS (11.8% for system load of 100% and 18.7% for system load of 80%). The scheduling fairness is also improved considerably. These observations indicate that the heuristic effec-

System Configuration		
Domain	Number of Computers	Relative Speeds of Computers
$D_1$	2	3.0, 3.0
$D_2$	6	1.5, 1.5, 1.5, 1.5, 1.5, 1.2
$D_3$ (slow)	3	1.2, 1.2, 1.0
Link Speeds: $[D_1 - D_2]$ : 1024Kbps $[D_1 - D_3]$ : 256Kbps $[D_2 - D_3]$ : 56Kbps		

**Table 11. Experimental Setting 3**

System Utilization	100%		80%	
	MDS	MDS-C	MDS	MDS-C
Algorithm	MDS	MDS-C	MDS	MDS-C
Mean Response Time (sec)	2450	2400	1650	1570
Mean Response Ratio	1.27	1.12	1.23	1.00
Fairness	1.88	1.53	2.51	1.03
Network Cost per Job (sec)	44.3	34.7	47.4	34.8

**Table 12. Performance of Network Cost Heuristic**

tively approximates the file transfer costs. The average network costs under MDS-C are 21.7% and 26.6% lower than those of MDS. On the other hand, for similar reason given in the previous subsection, MDS-C does not improve mean response time very much. We have performed a large number of experiments for different values of  $C$  on a discrete event simulator and the results showed that system performance is not very sensitive to the value of  $C$  when the ratio  $\frac{C}{link-speed}$  is in the range 0.01 to 0.2.

## 8 Conclusion

In this paper, we have investigated load balancing strategies in the multidomain environment where the computers are located in different local area networks which are physically wide apart from one another. A hierarchical architecture is integrated into the conventional least load scheduler to achieve scalability. Several techniques are proposed to tackle the problems inherent in the multidomain environment. Two strategies including quick load update and maintaining multiple least loaded computers at the scheduler are proposed to reduce the lag time in disseminating load updates. In addition, a heuristic that considers both machine load and network speed is suggested to estimate the completion time of executing jobs in remote domains. These techniques have been implemented in the hierarchical scheduler and tested on the BALANCE testbed. Performance results have indicated that the enhanced hierarchical scheduler adapts well to the highly dynamic multidomain environment and improves system performance

significantly over existing algorithms in terms of mean response time, mean response ratio and fairness.

## References

- [1] S. T. Chanson, W. Deng, C.-C. Hui, X. Tang, and M. Y. To. Multidomain load balancing. Technical Report HKUST-CS99-18, Department of Computer Science, HKUST, Dec. 1999.
- [2] M. Dahlin. Interpreting stale load information. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 285–296, June 1999.
- [3] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12(1):53–65, May 1996.
- [4] D. J. Evans and W. U. N. Butt. Load balancing with network partitioning using host groups. *Parallel Computing*, 20(3):325–345, Mar. 1994.
- [5] C.-C. Hui and S. T. Chanson. Efficient load balancing in interconnected LANs using group communication. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 141–148, May 1997.
- [6] C.-C. Hui and S. T. Chanson. Flexible and extensible load balancing. *Software Practice and Experience*, 27(11):1283–1306, Nov. 1997.
- [7] C.-C. Hui and S. T. Chanson. Hydrodynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 10(11):1118–1137, Nov. 1999.
- [8] C.-C. Hui and S. T. Chanson. Improved strategies for dynamic load balancing. *IEEE Concurrency*, 7(3):58–67, July–September 1999.
- [9] O. Kremien and J. Kramer. Methodical analysis of adaptive load sharing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 3(6):747–760, Nov. 1992.
- [10] P. Krueger and N. G. Shivaratri. Adaptive location policies for global scheduling. *IEEE Transactions on Software Engineering*, 20(6):432–444, June 1994.
- [11] T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, July 1991.
- [12] R. Leslie and S. McKenzie. Evaluation of loadsharing algorithms for heterogeneous distributed systems. *Computer Communications*, 22(4):376–389, Mar. 1999.
- [13] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, June 1988.
- [14] S. Lu and L. Xie. A scalable load balancing system for nows. *ACM SIGOPS Operating Systems Review*, 32(3):55–63, July 1998.
- [15] S. Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(9):1327–1341, Sept. 1988.