Adaptive Memory Allocations in Clusters to Handle Unexpectedly Large Data-Intensive Jobs

Li Xiao, *Member*, *IEEE Computer Society*, Songqing Chen, *Student Member*, *IEEE Computer Society*, and Xiaodong Zhang, *Senior Member*, *IEEE*

Abstract—In a cluster system with dynamic load sharing support, a job submission or migration to a workstation is determined by the availability of CPU and memory resources of the workstation at the time [21]. In such a system, a small number of running jobs with unexpectedly large memory allocation requirements may significantly increase the queuing delay times of the rest of jobs with normal memory requirements, slowing down execution of each individual job and decreasing the system throughput. We call this phenomenon the *job blocking* problem because the big jobs block the execution pace of majority jobs in the cluster. Since the memory demand of jobs may not be known in advance and may change dynamically, the possibility of unsuitable job submissions/migrations to cause the blocking problem is high, and existing load sharing schemes are unable to effectively handle this problem. We propose two schemes to address this problem. The first scheme, *Network RAM supported load sharing*, combines job migrations with network RAM, which uses remote execution to initially allocate a job to the most lightly loaded workstation and, if necessary, network RAM to provide a global memory space for the job larger than it would be available otherwise. This scheme has the merits of both job migrations and network RAM. Our experiments show its effectiveness and scalability. However, this scheme requires a network RAM facility in the cluster, which may cause additional overhead and increase cluster network traffic. In order to address this limit, we propose a second scheme, *memory reservation*, incorporated with dynamic load sharing, which adaptively reserves a small set of workstations to provide special services to the jobs demanding large memory allocations. As soon as the blocking problem is resolved by the memory reservation scheme, the system will adaptively switch back to the normal load sharing state. Both schemes target on handling large data-intensive jobs in clusters, and are mutually complementary. The network RAM supported load sharing scheme can fully utilize the cluster global memory space, while the memory reservation scheme has the advantage of simple implementations and low overhead. Thus, they both can be effective alternatives, and practically deployed in cluster computing under different system conditions.

Index Terms—Cluster computing, distributed systems, load sharing, job blocking, memory-intensive workloads, trace-driven simulations.

1 Introduction

Load sharing provides a system mechanism to dynamically migrate jobs from heavily loaded workstations to lightly loaded workstations, aiming at fully utilizing system resources. Following the load sharing principle, researchers have designed different alternatives by balancing the number of jobs/tasks among the workstations (see, e.g., [11], [13]), by considering memory allocation requirements of jobs (see, e.g., [3], [4]), and by considering both CPU and memory resources (see, e.g., [22], [23], [2]). Recently, we have developed dynamic load sharing schemes to schedule or migrate jobs without the knowledge of their memory allocation sizes before jobs start running [21].

In a cluster system with dynamic load sharing support, a new job can be submitted to a workstation or a running job can be migrated to the workstation under following conditions. When the workstation has idle memory space, the job can be accepted if the number of running jobs in the workstation is still less than a predetermined threshold which is the maximum number of job slots a CPU is willing to take (also called the CPU Threshold). When the workstation does not have idle memory space, or is even oversized, no jobs will be accepted without further checking the status of the CPU threshold. This strategy has shown its effectiveness in load sharing, particularly to schedule jobs with unknown memory allocation sizes [21]. However, in such a system, a small number of running jobs with large memory allocation requirements can be scattered among workstations to quickly use up the memory space, impeding job submissions to these workstations. Since these large jobs normally have long remaining processing times, eventually, all the workstations may become heavily loaded, stopping job submissions and migrations. We call this phenomenon the *job blocking* problem, which is rooted from unsuitable placements of these large jobs. The existence of these large jobs in a few workstations may increase the queuing delay times of the rest of jobs with relatively small memory requirements, slowing down executions of individual jobs and decreasing the cluster system's throughput. Since job sizes including the memory allocations are unknown in advance, the possibility of unsuitable job placements to cause the blocking problem is high, and existing load sharing schemes are unable to effectively handle this problem.

- *L. Xiao is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824. E-mail: lxiao@cse.msu.edu.*
- *S. Chen and X. Zhang are with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187. E-mail: {sqchen, zhang}@cs.wm.edu.*

Manuscript received 20 Feb. 2003; revised 27 Aug. 2003; accepted 18 Nov. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0013-0203.

When both job submissions and migrations are blocked in a cluster, it implies that the resource allocation in each workstation either reaches its memory threshold due to arrivals of some jobs with large memory demands, or reaches its CPU threshold, or both. Further job submissions or migrations will cause more page faults or queuing delays in a destination workstation. One simple solution would be to temporarily suspend the large jobs so that the job submissions will not be blocked. However, this approach will not be fair to the large jobs that may starve if job submissions continue to flow, or that can be executed only when the cluster becomes lightly loaded.

We have observed that CPU and memory resources are actually not fully utilized during the period of blocking [21], and will further present our performance results in Section 6. For example, some workstations reaching their CPU thresholds may still have idle memory space, while some workstations experiencing page faults may still have additional job slots available. Our recent experiments show that, when a cluster system is not able to further accept or migrate jobs, there are still large accumulated idle memory space volumes available among the workstations. This is because demanded memory allocations of a handful of jobs could not fit in any single workstation with other running jobs. We have also found that jobs are not evenly distributed among workstations, which increases the total job queuing time. Unfortunately, the dynamic load sharing scheme or the aftermentioned job suspension is not able to efficiently resolve this blocking problem by further utilizing the available resources. We target to address these inefficient resource allocations. Our observations and experimental results have motivated us to propose two schemes to further utilize resources and to quickly resolve the blocking problem.

The first scheme, network RAM supported load sharing, combines job migrations with network RAM. It uses remote execution to initially allocate a job to the most lightly loaded workstation and, if necessary, network RAM to provide a larger memory space for the job than would be available otherwise. The merits of remote execution has been well discussed in [6]. This scheme has the merits of both job migrations and network RAM. Our experiments show its effectiveness and scalability. However, this scheme requires a network RAM facility in the cluster, which may cause additional overhead and increase cluster traffics.

The second proposed scheme, memory reservation, is to address the limit of the first one. It adaptively reserves a small set of workstations (called reserved workstations) to provide special services to the jobs demanding large memory space. This scheme will quickly enforce the cluster to allocate and accumulate sufficient memory resources for unexpectedly large data-intensive jobs. As soon as the blocking problem is resolved by the memory reservation, the system will adaptively switch back to the normal load sharing state. On one hand, this method can improve the utilization of CPU and memory resources in nonreserved workstations because jobs with normal sizes can be smoothly executed without the interference of large jobs. On the other hand, large jobs are treated fairly because they are served by reserved workstations. We will show the two schemes are mutually complementary for the same objective.

The paper is organized as follows: We present the network RAM supported scheme in Section 2 and the memory reservation scheme in Section 3. We discuss the performance evaluation environment in Section 4. We evaluate the two schemes in Sections 5 and 6. We further compare the two schemes in Section 7 and conclude our study in Section 8.

2 Network RAM Supported Load Sharing

With the rapid development of CPU chips and the increasing demand of data accesses in applications, memory resources in a workstation cluster become more and more expensive relative to CPU cycles. Effective usage of global memory resources is an important consideration in the design of load sharing policies for cluster computing. When a workstation does not have sufficient memory space for its assigned jobs demanding unexpected large memory space, the system will experience a large number of page faults, resulting in a long delay for each job. There are two major approaches to more effectively use global memory resources in a workstation cluster, aimed at minimizing page faults in each local workstation and improving overall performance of cluster computing: 1) job-migration-based load sharing schemes (such as [11], [3], [23]) and 2) network RAM (such as [8], [14], [1]). A job-migration-based load sharing system attempts to migrate jobs from a workstation without sufficient memory space to a lightly loaded workstation with large idle memory space for the migrated jobs. When a job migration is necessary, the migration can be either a remote execution (where a job is initiated on a remote workstation) or a preemptive migration which suspends the selected job and moves it to a remote workstation where the job is restarted. In a network RAM system [9], if a job cannot find sufficient memory space for its working sets, it will utilize idle memory space from other workstations in the cluster through remote paging. Since remote paging is relatively slower than accessing local memory, but much faster than local disk access, the idle global memory space or the network RAM can be considered as another layer between the local memory and the local disk in the memory hierarchy of a workstation.

Besides sharing the same objective of reducing page faults in each local workstation, the two approaches share another common technical feature in their implementations. Both systems maintain a global load index for each workstation about how its CPU and/or memory resources are being utilized. This record is either stored in a master workstation or distributed among the workstations and is updated periodically by the cluster workstations.

There are several major differences between the two approaches in the ways that the global memory resources are shared. Because of these differences, each approach has its own merits and limits. First, in a network RAM cluster system, a workstation is provided with a huge global memory space for its jobs. The global memory space can even be larger than its local disk space. Thus, it is possible to eliminate accesses to local disks due to page faults in a network RAM cluster. In contrast, memory allocations of a job could be limited by the local memory size of a workstation in a migration-based load sharing cluster system where local memory modules are not shared by other workstations. Thus, a network RAM cluster system could be more beneficial to large or nonmigratable data-intensive jobs than a migration-based load sharing cluster system. Second, the effectiveness of global paging operations in a network RAM cluster system

is heavily dependent on the cluster network speed. In contrast, the network, in general, is less frequently used in a remote-execution-based load sharing cluster system. In other words, a remote-execution-based load sharing system relies less on the network speed than a network RAM system. Finally, a migration-based load sharing system is able to balance the workloads among workstations by sharing both CPU and memory resources, while a network RAM system only considers global memory resources for load sharing. Without job migrations, job executions may not be evenly distributed in a cluster—some workstations can be more heavily loaded than others. Although the lightly loaded workstations in a network RAM cluster system can be used as memory servers for heavily loaded workstations, their CPU resources are not fully utilized by the cluster.

2.1 Job-Migration-Based Load Sharing, Network RAM, and Network RAM Supported Load Sharing

Network RAM and job-migration-based load sharing related operations on workstation j, for $j = 1, \ldots, P$, are characterized by the following variables:

1. RAM_j: the total memory space provided for user-level programs in MBytes of the workstation,
2. RP_j: the amount of remote paging in MBytes from the workstation,
3. FM_j: the idle memory space in MBytes of the workstation, and
4. MT_j: the memory threshold (the memory space for the stable working set) in MBytes is the total amount of memory thresholds accumulated from the running jobs on the workstation.

If $RAM_j > MT_j$, page faults will rarely occur, otherwise, paging will be frequently conducted during executions of jobs in the workstation.

2.2 Network RAM Organizations

A network RAM organization allows each workstation to use not only its own local memory, but also to access idle memory space of other workstations through remote paging in a cluster. The memory allocation decision for a job on workstation j is made by

$$memory\ allocation = \begin{cases} local\ memory & \text{if } MT_j < RAM_j \\ global\ memory & \text{if } MT_j \geq RAM_j, \end{cases}$$

where the global memory allocation is implemented by finding the most lightly loaded workstation one by one for remote paging based on the following search algorithm:

Allocate the idle local memory space to the arrival job;
$MD_j = MT_j$;
While ($MD_j \geq RAM_j$) and
(idle memory space is available elsewhere)
do
find workstation i with the largest idle memory space among $P - 1$ workstations
(excluding workstation j);
allocate $RP_i = min\{MD_j - RAM_j, FM_i\}$ MBytes from workstation i to the job in workstation j;
$FM_i = FM_i - RP_i$;
$MD_j = MD_j - RP_i$;

where MD_j represents the current local memory demand on workstation j. The *while* loop continues until the memory demand is met or no idle memory available in the system. If $MD_j \geq RAM_j$ after the global allocations, disk accesses due to page faults will occur in workstation j. In order to minimize the global paging, we give local memory accesses the highest priority. The global paging is only conducted when the remote workstation has additional idle memory space. Therefore, when a new local job arrives, the network RAM paging services for remote jobs will be transferred to other workstations if any memory space occupied by remote pages is needed for this new job.

2.3 CPU-Memory-Based Load Sharing

The job-migration-based load policy we have selected for this comparative study is the CPU-Memory-based load sharing scheme [23], which makes a job migration decision by considering both CPU and memory resources comprehensively. The basic principle of this scheme is as follows: When a workstation has sufficient memory space for both running and requesting jobs ($MT_j < RAM_j$), the load sharing decision is made by a CPU-based policy where the load index in each workstation is represented by the length of the CPU waiting queue. As long as the CPU waiting queue is not larger than the threshold which is set based on the CPU capability, the requesting jobs will be locally executed in the workstation. Otherwise, the load sharing system finds the remote workstation with the shortest waiting queue to either remotely execute this job or to preemptively migrate an eligible job from the local workstation to the remote workstation. When the workstation does not have sufficient memory space for the jobs ($MT_j \geq RAM_j$), the load sharing scheme attempts to migrate jobs to suitable workstations or even to hold the jobs in a waiting pool if necessary.

During an execution of a memory-intensive job, page faults may occur periodically. Each such period is called a *transition*, where page faults are conducted to bring a working set into memory. The data references will then be memory hits for a while until the working set changes and page faults are conducted, forming the next transition period. The local reference period is called a *phase*. If the phases of a job are clearly distinguished, the best time to do a preemptive migration is at the end of a phase and before another transition period is started (bring in the next working set into memory). The migrated job will carry no data or a small data set to a remote workstation. However, in practice, it may be difficult to predict the data access phase and transition patterns of so many different jobs. If this prediction is impossible, remote executions should be a practically optimal solution for load sharing of memory-intensive jobs [23]. For this reason, remote executions are used in our CPU-Memory-based load sharing policy.

2.4 Network RAM Supported Load Sharing

We have discussed advantages and limits of the network RAM and the remote-execution-based load sharing scheme. A further optimization step for overcoming the limits of each scheme and resolving job blocking problems is to combine them, and we call this scheme network RAM supported load sharing. Here is the basic idea of this

improved load sharing scheme. When a workstation has sufficient space for both current and requesting jobs, the job execution location will be determined by the CPU-based policy. When a workstation runs out of memory space for both current and requesting jobs, the scheme attempts to migrate the new arrival job to the most lightly loaded workstation. If the workstation does not have sufficient memory space for the job, the network RAM will be used to satisfy the memory allocation of the job through remote paging. The scheme is outlined as follows:

If ($MT_j \geq RAM_j$)
  find workstation i with the largest idle memory
  space among P workstations;
  If $i \neq j$
    remotely execute the job at workstation i;
  If ($MT_i \geq RAM_i$)
    allocate global memory by using network RAM;
else
  schedule the job by the CPU-based load sharing policy;

2.5 Potential Limits of Network RAM Supported Load Sharing

We will show later in the paper the performance and resource utilization advantages of the Network RAM supported load sharing to resolve job blocking problems. In practice, there will be two limits for this approach. First, Network RAM facility may not be available in every cluster. Widely global memory sharing in a cluster may not be allowed for system security and reliability reasons. Second, Network RAM will cause certain system overhead. A particular concern can be focused on the increased cluster network traffic due to remote memory accesses. If the cluster is also used by running parallel processing jobs, the performance of these jobs will be certainly affected by the limited bandwidth of the cluster. In order to address the limits of the Network RAM supported load sharing, we propose another alternative that will be presented in the next section.

3 Memory Reservation

The objective of the second scheme is to quickly make a workstation with large memory space to be dedicated to the job with unexpectedly large memory allocation. We assume that a workstation with large memory space will be available in the cluster to satisfy the large job and eventually resolve the blocking problem. This is certainly viable because the memory space becomes increasingly large with proportionally decreased price. Under such an assumption, our scheme does not require Network RAM, and will execute jobs locally, minimizing the cluster network traffic.

In order to make our solution effective, we need to address two potential concerns. First, we need to dynamically identify large jobs and to find suitable workstations for them to execute on. Second, the policy should be beneficial to both large and other jobs. We propose the second scheme that is our software method for adaptive memory reservation.

3.1 Job Reallocations

Here is the basic idea of our software method of memory reservation, which can be easily incorporated with the dynamic load sharing scheme. The blocking problem is initially detected when a workstation experiences a certain amount of page faults, but the scheduler cannot find a qualified destination to migrate jobs from this workstation. If the accumulated idle memory space size in the cluster is larger than the average user memory space of workstations in the cluster, the reservation routine is activated. The routine first identifies several workstations with comparably largest idle memory space, and then selects one running the smallest number of jobs. We call this selected workstation the *lightly loaded workstation* for the purpose of memory reservation. The routine continues to block job submissions and migrations to this workstation. The time period between identifying the workstation and completions of the running jobs in the workstation is called the *reserving period*. (One alternative is to end the reserving period as soon as the available memory space in the reserved workstation is sufficiently large for a job migration with large memory demand). During the reserving period, if the blocking problem disappears, the system will be back to the normal load sharing state. If the blocking problem still exists after the reserving period, the reservation routine will migrate a job with the largest memory demand suffering serious page faults to the reserved workstation. When the blocking problem is detected again in a workstation, the reservation routine will first try to migrate a job to a reserved workstation if it exists, that is able to provide sufficient memory space and job slots. Otherwise, the reservation routine will start another reserving period to identify an appropriate workstation. As soon as the blocking problem is resolved by the reservation, the system will be adaptively switched back to the normal load sharing state. The transition between the memory reservation for reserved computing and normal load sharing is quite natural. As a reserved workstation completes its special service, the scheduler will view it as a regular workstation and resume normal job submissions to the workstation. Notice that the processes of starting and releasing a reservation are not only adaptive, but also cause little additional overhead.

The framework of the reservation routine is embedded in the dynamic load sharing system in a workstation as follows:

While the load sharing system is on
    if job submissions or/and migrations are allowed
      $general\_dynamic\_load\_sharing()$;
    else start reservation by {
      if ($\exists\ reservation\_flag(reserved\_ID) == 1$)
      && (the workstation has enough available
      resources)
        $node\_ID = reserved\_ID$;
      else {
        $node\_ID = reserve\_a\_workstation()$;
        $reservation\_flag(node\_ID) = 1$;
        }
      $job\_ID = find\_most\_memory\_intensive\_job()$;
        $migrate\_job(job\_ID, node\_ID)$;
        }

The functions in the framework are defined as follows:

- $general\_dynamic\_load\_sharing()$: conducts regular operations of dynamic load sharing including monitoring, local/remote job submissions, and job migrations [21].
- $reserve\_a\_workstation()$: selects the *lightly loaded workstation* in the cluster, continues to block job submissions to the workstation until execution completions of all running jobs in the workstation, and returns its $node\_ID$ and sets the $reservation\_flag$.
- $find\_most\_memory\_intensive\_job()$: identifies the job with the largest memory demand, and returns its job_ID.
- $migrate\_job(job\_ID, node\_ID)$: migrates the identified job to the reserved workstation.

The $reservation\_flag$ is turned off when the reserved workstation completes executions of all the migrated jobs, which resumes the normal job submissions and migrations to the workstation.

3.2 The Rationale of Our Solution

The potential performance gain of our approach comes from four sources. First, *the idle memory space is available among workstations*. Unfortunately, the available space in each individual workstation is not large enough to serve any incoming jobs. A considerable amount of accumulated idle memory space in the cluster can be utilized by job reallocations. Reserving a workstation plays an equivalent role to move some accumulated idle memory space to the reserved workstation, so that the workstation is able to serve large jobs that could not fit in any individual idle memory space before the reservation.

Second, *the identified large job is likely to be a large job with long lifetime*. The job is identified after the reserving period. If a job is observed to demand a large memory space, causing page faults for a period of time, this job will be likely to continue to stay and execute for a longer time than other jobs in the workstation for two reasons: 1) Experiments have shown that a job with a large memory demand in process interactions is less competitive than jobs with small memory allocations in conventional operating systems, such as Unix and Linux [12]. 2) Experiments have also shown that a job having stayed for a relatively long time is predicted to continue to stay for a even longer time than other jobs [11]. After a large job is migrated away, the rest of the jobs will be served quickly, and submissions to the workstation will continue to flow. The principle of the shortest remaining processing time policy [15] is implicitly applied here.

Third, *the concern of unfairly treating large jobs does not exist*. We specially reserve workstations to process these large jobs that are less competitive than jobs with small memory allocations.

Finally, *in practice, the percentage of large jobs in a job pool is low*. If there are too many large jobs, the proposed method will reserve too many workstations so that normal jobs cannot run. This causes unfairness to normal jobs. This concern could be easily addressed because several studies (e.g., [11] and [17]) have shown that the percentage of exceptionally large jobs is very low in real-world workloads.

Regarding the fairness in job scheduling, a recent paper classifies existing scheduling policies into one of the three categories: "always fair," "sometimes unfair," or "always unfair" [20]. We characterize our proposed solutions as "try to be always fair with strong adaptive efforts."

3.3 What is the Memory Reservation Not Able to Do?

When the accumulated idle memory space in the cluster is not sufficiently large, the memory reservation will not be effective. If the accumulated idle memory space is smaller than the user space of a single workstation, it will be difficult to reserve a workstation providing its entire memory space. Under such a condition, the cluster memory resources have been sufficiently utilized. Examining the accumulated idle memory space in the cluster is one way to detect this condition. If a workstation cannot be reserved within a predetermined time interval, it implies that the cluster is truly heavily loaded.

In a heterogeneous cluster system, a reserved workstation will be the one with relatively large physical memory space. If the user space in the reserved workstation is still not sufficiently large for a migrated job, this implies that this job may not be suitable in this cluster unless our first scheme, Network RAM supported load sharing, is applied. If this job has to be executed in the cluster, the memory reservation method will provide a reserved workstation for dedicated service, where its page faults will not affect performance of other jobs. The memory reservation may not work well for specific workloads where big jobs are dominant. Again, this case is rare in practice.

4 Performance Evaluation Environment

The two proposed schemes have been evaluated in a trace-driven simulation environment. The traces have been collected by our kernel instrumentation tool. The two schemes and their variations on a cluster of workstations are simulated to accept these traces as workloads. In this section, we will present our performance evaluation environment.

4.1 Tracing Job Execution at the Kernel Level

The lifetime of a job has been used as an important factor in load sharing designs. Which process to be migrated in existing scheduling policies is considered by predicting the lifetime of CPU intensive jobs. Detailed breakdowns of the lifetime will provide more insightful considerations for load sharing decisions.

We have developed facilities by kernel instrumentation [21], which measures different portions of the lifetime for a job execution. Particularly, the instrumentation records when a job process is interrupted for a system event, and how long this event lasts. A trace buffer is initially allocated when the system is booted to collect the system traces. The facilities also dynamically measure

1. current ages and lifetime of jobs,
2. the sizes of memory allocation for each running job and idle memory space in each workstation,
3. events of page faults in each workstation,
4. the read/write operations of each running job, and
5. the status of I/O buffer cache in each workstation.

TABLE 1
Execution Performance and Memory Related Data of the Six Spec 2000 Benchmark Programs

Programs	description	input file	working set (MB)	lifetime (s)
apsi	climate modeling	apsi.in	196.0	2,619.0
gcc	optimized C compiler	166.i	145.0	228.0
gzip	data compression	input.graphic	195.0	249.0
mcf	combinatorial optimization	inp.in	80.0	969.0
vortex	database	lendian1.raw	115.0	345.0
bzip	data compression	input.graphic	200.0	403.0

4.2 Application Workloads

We have selected two groups of workloads. The first group (workload group 1) consists of six SPEC-2000 benchmark programs: *apsi, gcc, gzip, mcf, vortex*, and *bzip*, which are both CPU and memory intensive. Using the facilities described above, we first run each program in a dedicated environment to observe the memory access behavior without major page faults (except for cold misses) and page replacement (the demanded memory space is smaller than the available user space). We selected a 400 MHz Pentium II with 384 Mbyte physical memory and a swap space of 380 MBytes to run workload group 1. The operating system is Redhat Linux release 6.1 with the kernel 2.2.14. Table 1 presents the basic experimental results of the six SPEC-2000 programs, where the "description" gives the application nature of each program, the "input file" is the input file names from SPEC200 benchmarks, the "working set" gives the maximum size of the allocated memory space during the execution, and the "lifetime" is the total execution time of each program.

The second group (workload group 2) consists of seven large scientific and system programs that are representative CPU-intensive, memory-intensive, and/or I/O-active jobs: *bit-reversals* (bit-r), *merge-sort* (m-sort), *matrix multiplication* (m-m), *a trace-driven simulation* (t-sim), *partitioning meshes* (metis), *cell-projection volume rendering for a sphere* (r-sphere), and *cell-projection volume rendering for flow of an aircraft wing* (r-wing). The descriptions and related citations of these applications can be found in [21].

We first measured the execution performance of each program and monitored their memory performance related activities in a dedicated computing environment of a 233 MHz Pentium PC with 128 MByte main memory and a swap space of 128MB, running Linux version 2.0.38. The program memory demands in this group are smaller than the ones in workload group 1. So, the workstations we selected here are less powerful than the workstations used to run programs of workload group 1. Table 2 presents the experimental results of all the seven programs, where the "data size" is the number of entries of the input data, the "working set" gives a range of the memory space demand during the execution, and the "lifetime" is the total execution time of each program.

4.3 Trace-Driven Simulations

4.3.1 Two Simulated Clusters

We have simulated two homogeneous clusters, each of which has 32 workstations. Application programs in workload group 1 are run in cluster 1, where the CPU speed is 400 MHz, memory size is 384 MBytes, and the swap space is 380 MBytes. While application programs in workload group 2 are run in cluster 2, where the CPU speed is 233 MHz, memory size is 128 MBytes, and the swap space is 128 MBytes. In simulations of both clusters, the memory page size is 4 KBytes. We use the parameters in Cheetah X15 disk (15,000 rpm) in our simulator: seek time = 3.9 ms, rotation latency = 2 ms, internal transfer time for 4 KB is 0.08 ms, and the total disk latency is 5.98 ms. We also use UltraSCSI I/O bus as the connection between the disk and the main memory, whose bandwidth is 160MBps. Thus, it takes 6 ms to fetch and transfer a page of 4 KByte data from disk to DRAM in our simulation. The context switch time is set to 2.5 μs based on the CPU speeds. A recent measurement study in [7] shows that the context switch time is proportionally reduced as the CPU speed increases, based on measurements on a large range of CPUs from 90 MHz to 1.467 GHz. The selected context switch time in our simulation is consistent with theirs. Ethernet connection speed, B, is 100 Mbps or 1 Gbps. The remote submission/execution cost, r, is 0.025 second. The preemptive migration cost is estimated by assuming the entire memory image of the working set will be transferred from a source to a destination node for a job migration, which is $r + \frac{D}{B}$, where r is a fixed remote execution cost in second, and D is the amount of data in bits to be transferred in the job migration. Each workstation maintains a global load index containing CPU, memory, and I/O load status information of other computing nodes. The load sharing system periodically collects and distributes the load information among the workstations.

4.3.2 Workload Traces

The two groups of workload traces are collected by using our facilities to monitor the execution of the six SPEC 2000 benchmark programs (workload group 1) and the seven

TABLE 2
Execution Performance and Memory Related Data of the Seven Application Programs

Programs	data size	working set (MB)	lifetime (s)
bit-r	2^{23}	64.22	192.26
m-sort	2^{23}	64.27	82.76
m-m	$1,700^2$	66.37	4902.29
t-sim	31,061	4.64	41.63
metis	1M-4M	1.37-4.30	124.41
r-sphere	150,000	36.84 — 39.66	318.64
r-wing	500,000	19.53 — 23.39	72.78

application programs (workload group 2) at different submission rates on a Linux workstation. The programs in each workload group are randomly selected to submit. Job submission rates are generated by a lognormal function:

$$R_{ln}(t) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\ln t-\mu)^2}{2\sigma^2}} & t > 0 \\ 0 & t \leq 0, \end{cases} \quad (1)$$

where $R_{ln}(t)$ is the lognormal arrival rate function, t is the interarrival time for job submissions in a unit of seconds, and the values of μ and σ adjust the degrees of the submission rate. The lognormal job submission rate has been observed in several practical studies (see, e.g., [10], [19]). Five traces for each group are collected in each workload group (1 and 2) with five different arrival rates:

- *Trace-1* (light job submissions): $\sigma = 4.0$, $\mu = 4.0$, and 359 jobs submitted in 3,586 seconds.
- *Trace-2* (moderate job submissions): $\sigma = 3.7$, $\mu = 3.7$, and 448 jobs submitted in 3,589 seconds.
- *Trace-3* (normal job submissions): $\sigma = 3.0$, $\mu = 3.0$, and 578 jobs submitted in 3,581 seconds.
- *Trace-4* (moderately intensive job submissions): $\sigma = 2.0$, $\mu = 2.0$, and 684 jobs submitted in 3,585 seconds.
- *Trace-5* (highly intensive job submissions): $\sigma = 1.5$, $\mu = 1.5$, and 777 jobs submitted in 3,582 seconds.

The jobs in each trace were randomly submitted to 32 workstations. When we run clusters of 16 workstations and eight workstations, the numbers of jobs are scaled down proportionally. Each job has a header item recording the submission time, the job ID, and its lifetime measured in the dedicated environment. Following the header item, the execution activities of the job are recorded in a time interval of every 10 ms including CPU cycles, the memory demand/allocation, buffer cache allocation, number of I/Os, and others. Thus, dynamic memory and I/O activities can be closely monitored. During job interactions, page faults are generated accordingly by an experiment-based model presented in [21].

The five traces for workload group 1 are represented by *SPEC-Trace-1*, *SPEC-Trace-2*, *SPEC-Trace-3*, *SPEC-Trace-4*, and *SPEC-Trace-5*; and the five traces for workload group 2 are represented by *App-Trace-1*, *App-Trace-2*, *App-Trace-3*, *App-Trace-4*, and *App-Trace-5*.

The *slowdown* of a job is the ratio between its wall-clock execution time and its CPU execution time. A major performance metric we have used is the average slowdown of all jobs in a trace. Major contributions to slowdown come from queuing time waiting for CPU service, the delay of page faults, and the overhead of migrations and remote submission/execution. The average slowdown measurement can determine the overall performance of a load sharing policy, but may not be sufficient to provide performance insights.

The following additional performance metrics are also used in our evaluation:

- *Average execution time per job* is defined as $\bar{t} = \frac{\sum_{i=1}^{n} t_i}{n}$, where t_i is the measured execution time of an individual job, and n is the number of jobs in a given workload.
- *Execution time breakdowns*: The average execution time is further broken into CPU service time, queuing time, disk access time due to page faults, and networking time for job migrations or remote pagings including network contention time.

For a given workload trace, we have also measured the average/total execution time and its breakdowns. Conducting the trace-driven-simulations on a 32 node cluster, we have evaluated the performance of Network RAM supported dynamic load sharing and memory reservation schemes by comparing their slowdowns and execution times of several application workloads with dynamic load sharing without such supports.

4.3.3 The Methods to be Compared

We have compared the following five methods in this study.

- **Base**: Jobs are executed without load sharing. (As shown later in the paper, the job execution times in this environment are extremely high due to long paging and queuing times).
- **LS_RE**: Jobs are scheduled with CPU-memory-based load sharing with remote executions.
- **Net_RAM**: Jobs are executed in a Network RAM system.
- **LS_Net_RAM**: Jobs are scheduled by the Network RAM supported load sharing scheme.
- **M-Reservation**: The system adaptively reserves a small set of workstations to provide special services to the jobs demanding large memory allocations when the blocking problem occurs.

5 Performance Evaluation

We target evaluating and comparing the performance merits and limits for a given workload scheduled by a job-migration-based load sharing policy, supported by Network RAM, supported by the combined policy, or without load sharing or Network RAM, under various system and workload conditions. Our performance evaluation targets understanding the effects of network bandwidth changes to both the job-migration-based load sharing scheme and the Network RAM supported by remote paging. We have quantitatively evaluated two performance trade offs for comparing the two schemes: 1) the trade off between reducing local disk accesses due to page faults and increasing the network bandwidth demand due to remote paging; 2) the trade off between reducing local disk accesses by Network RAM and balancing job execution among workstations by job migrations. We will show the effectiveness of the Network RAM supported load sharing. We first use trace "App-Trace-1" for evaluation to show performance insights of different schemes. The reason we choose the "App-Trace-1" is that this workload trace is in a light job submission mode where the blocking problem is not serious, which does not favor our proposed scheme. After that, we present the average execution performance of all the traces.

5.1 Impact of Available Network Bandwidths

Both job migration and remote paging rely on the cluster network for data transfers. However, the performance of each scheme is affected differently by changes of the

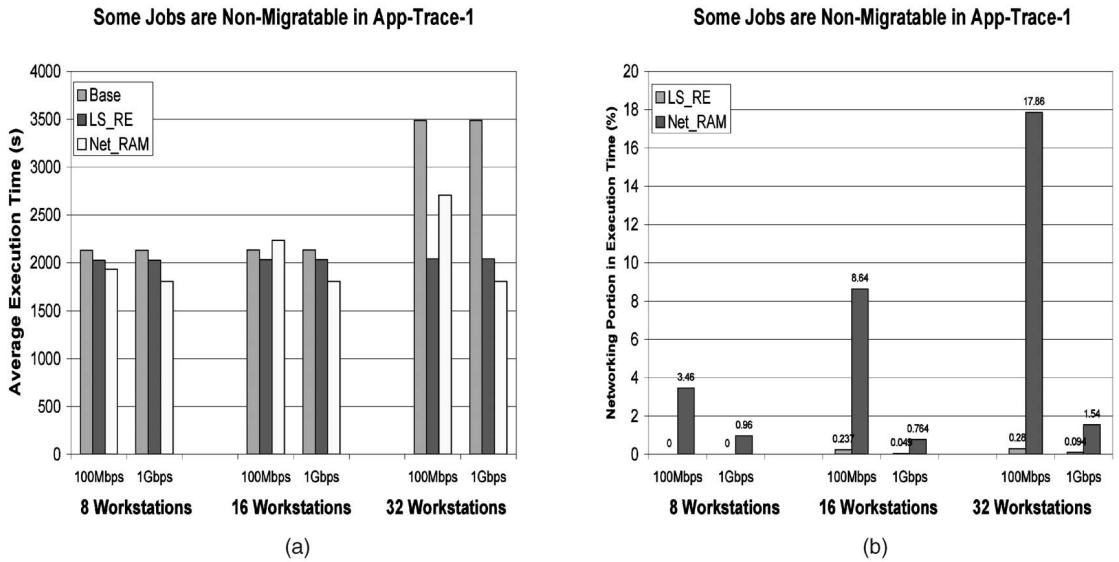

Fig. 1. The average execution times per job (a) and the networking latency portions in the execution times (b) of "App-Trace-1" with some job migration restrictions running on clusters of 8, 16, and 32 workstations.

network speed. Fig. 1 presents the average execution time per job (Fig. 1a) and the network delay portion in the execution times (Fig. 1b) of "App-Trace-1" running on clusters of 8, 16, and 32 workstations, where the jobs are executed in "Base," scheduled by "LS_RE," and executed on "Net_RAM." The network speed varies from 100 Mbps to 1 Gbps.

We have the following observations based on the experimental results in Fig. 1. First, the performance of "LS_RE" is not significantly affected as the cluster scales from 8 to 16, and from 16 to 32 workstations. The performance improves only slightly as the network speed increases from 100 Mbps to 1 Gbps. This is because the data communication via the network by remote executions is a small portion in the total execution time (0 to 0.094 percent, see Fig. 1b). Second, the performance of "Net_RAM," supported by remote paging, is quite sensitive to the network speed and the number of workstations in the cluster. For example, the average execution time of "Net_RAM" is 5 percent lower than that of "LS_RE" on the cluster of eight workstations where the network speed is 100 Mbps. As the network speed increases to 1 Gbps, the average execution time of "Net_RAM" is further reduced by 10 percent. However, as the cluster of 10 Mbps increases to 16 workstations, the average execution time of "Net_RAM" increases (to about 10 percent higher than that of "LS_RE"). As the cluster speed increases to 1 Gbps, the execution time is significantly reduced, and is 20 percent lower than that of "LS_RE." We have observed more sensitive changes of execution times as the number of workstations increases to 32. Our experiments show that the cluster scalability and workload performance when using Network RAM are highly dependent on the network speed of the cluster because the network latency due to data transfer and contention is a significant portion in the total execution time (0.96 percent to 17.86 percent, see Fig. 1b). Finally, in the workload of "App-Trace-1," some jobs are marked as nonmigratable. Therefore, the power and benefits of job migrations may be limited.

In order to fully take advantage of job migrations, we released the restrictions on the nonmigratable jobs so that remote executions can be applied to all the jobs in "App-Trace-1." Fig. 2 presents the average execution time per job (Fig. 2a) and the networking latency portions in the execution times (Fig. 2b) of the modified "App-Trace-1" scheduled by "LS_RE" in comparisons with "Base" and "NET_RAM" on the clusters of 8, 16, and 32 workstations. We show that the performance of "LS_RE" is indeed improved. In this case, the remote-execution-based load sharing policy not only outperforms the Network RAM, but is also more cost-effective without a need of a Network RAM facility.

From the scalability point of view, "LS_RE" demands less network bandwidth in order to scale the cluster by connecting more workstations than "Net_RAM" does. For example, the execution times of "LS_RE" almost remain the same when the number of workstations changes from 8 to 16, and 32 workstations for both 100 Mbps and 1 Gbps networks, which means it is scalable. (Recall that the number of jobs submitted to the cluster is proportional to the number of workstations.) In comparison, "Net_RAM" is only scalable for the 1 Gbps network.

5.2 Trade Offs between Page Fault Reductions and Load Sharing

Page faults in the Network RAM are reduced at the cost of additional network contention and delay. Although page fault reductions may be limited by the remote-execution-based load sharing scheme for large data-intensive jobs, the scheme requires less additional network support compared with the Network RAM. In order to provide insights into the trade offs between the two schemes, we present the execution time breakdowns of "APP-Trace-1" where all jobs are migratable in Figs. 3 and 4. The execution time of a workload consists of "CPU," "networking," "page faults," and "queuing," portions. "CPU" is the execution time by the CPU for the workload. "Networking" is the time spent on the cluster network, which is used for remote pagings by the Network RAM, or for remote executions by the load

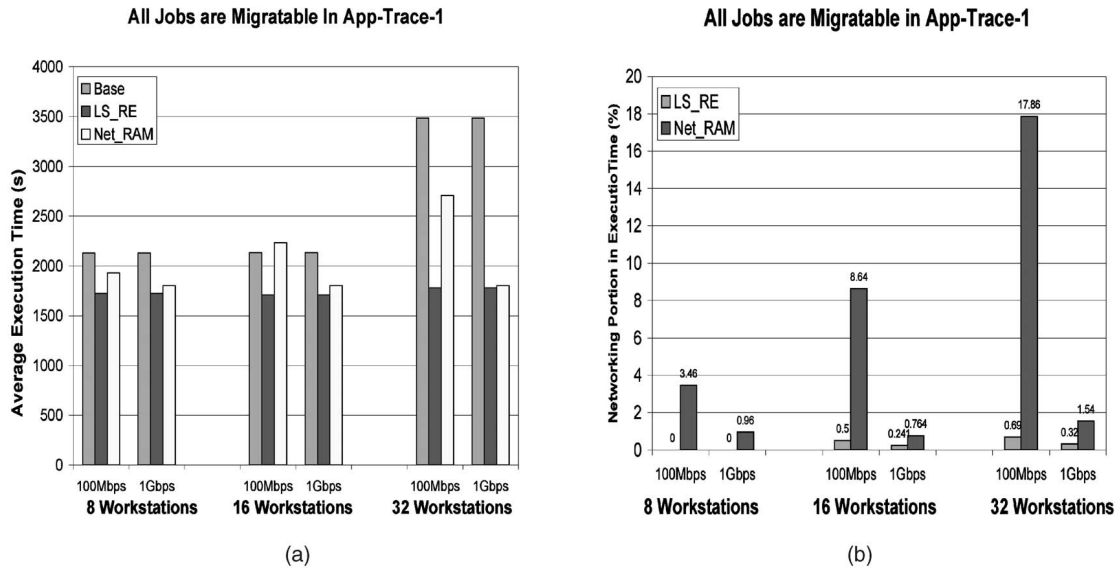

Fig. 2. The average execution times per job (a) and the networking latency portions in the execution times (b) of "App-Trace-1" without job migration restricitons running on clusters of 8, 16, and 32 workstations.

sharing scheme (including network contention time). "Page faults" is the local disk delay time for both schemes. "Queuing" is the average waiting time for a job to be executed on a workstation. When the workload is executed on a 100 Mbps cluster of 8 and 16 workstations, the networking time for remote pagings by the Network RAM is a distinguishable portion in the execution time. For example, the networking times contribute 4 percent and 9 percent to the execution times on the 8-workstation cluster and the 16-workstation cluster (see Figs. 3a and 4a), respectively. In contrast, the networking time for remote executions by the load sharing scheme is insignificant (0.07 percent and 0.5 percent). Consequently, the queuing time for each job in the Network RAM is significantly increased by networking delay, causing much longer execution times than for the remote-execution-based load sharing scheme.

We have also shown that the networking time portions in the executions of the workload by the Network RAM are significantly reduced by increasing the bus speed from 100 Mbps to 1 Gbps. Consequently, the queuing time for each job is also significantly reduced (see Figs. 3b and 4b).

Another trade off of the two schemes is between page fault reductions and load sharing. Without job migrations, job executions may not be evenly distributed among the workstations by the Network RAM although page faults can be significantly reduced through remote pagings. The unbalanced loads among workstations when using Network RAM

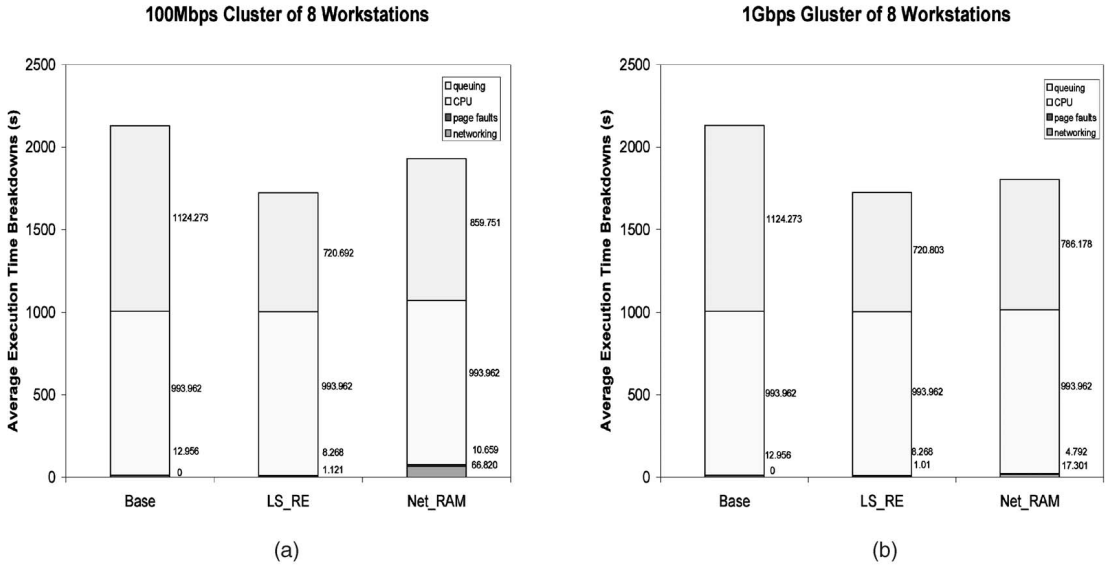

Fig. 3. The average execution time per job of "App-Trace-1" without job migration restrictions running on a 100 Mbps cluster (a) and a 1 Gbps cluster (b) of eight workstations.

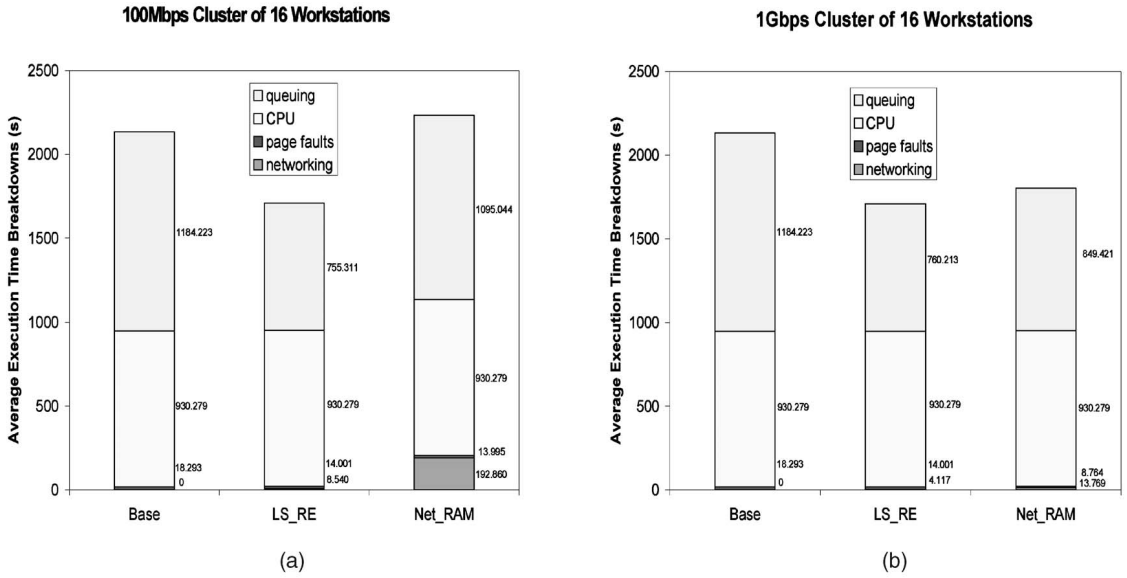

Fig. 4. The average execution times per job of "App-Trace-1" without job migration restrictions running on a 100 Mbps cluster (a) and a 1 Gbps cluster (b) of 16 workstations.

is another reason for the long queuing times for the workload executed on the 100 Mbps clusters of 8 and 16 workstations.

5.3 Performance of Network RAM Supported Load Sharing

The trade offs presented in the previous section motivate us to propose the Network RAM supported load sharing scheme ("LS_Net_RAM"). This scheme has been evaluated on a cluster of 32 workstations with 1 Gbps network. Each workload trace is again divided into two types: 1) some jobs are restricted for migrations in a trace and 2) all the jobs in a trace are migratable. Figs. 5 and 6 present the total execution times of all the five application traces of both type 1 (Figs. 5a and 6a) and type 2 (Figs. 5b and 6b), and Figs. 5 and 6 present the total execution times of all the five SPEC traces of both type 1 (Figs. 5a and 6a) and type 2 (Figs. 5b and 6b). Our experiments show that "LS_Net_RAM" outperforms all other schemes for both groups of traces of both types, particularly when the job submission rate is high (for traces of APP-Trace-5 and SPEC-Trace-5).

6 Performance Evaluation of Memory Reservation

Using the same trace-driven simulation, we have evaluated the memory reservation scheme. Since this scheme is not as sensitive as the Network RAM supported load sharing to the network speed, we only present the performance on a 1 Gbps cluster of 32 workstations. The performance data will also be used for performance comparisons between the two schemes.

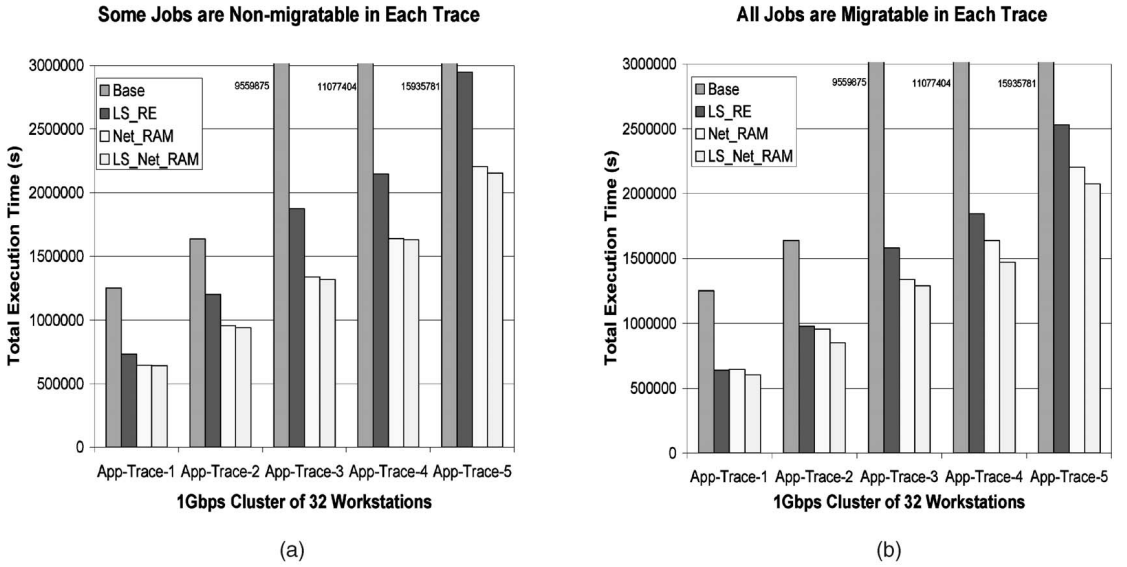

Fig. 5. The total execution times of all the five application traces ((a) for the five traces where some jobs are nonmigratable and (b) for the five traces where all the jobs are migratable), running on a 1 Gbps cluster of 32 workstations.

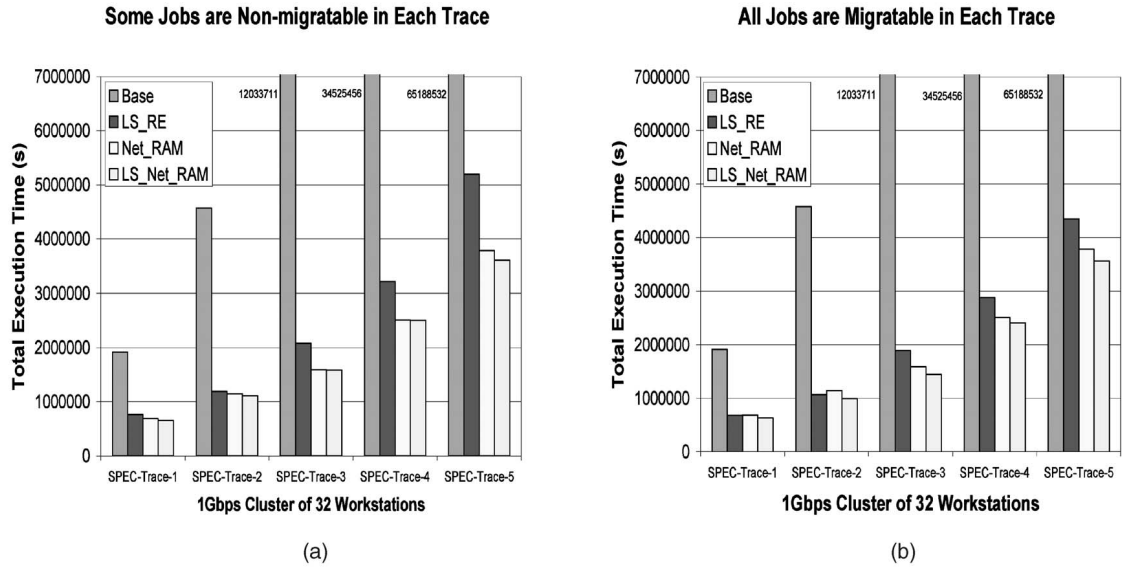

Fig. 6. The total execution times per job of all the five SPEC traces ((a) for the five traces where some jobs are nonmigratable and (b) for the five traces where all the jobs are migratable), running on a 1 Gbps cluster of 32 workstations.

6.1 Effective to Heavy Workloads with Large Working Sets by Improving Memory Utilization

We first evaluate the memory reservation scheme by the SPEC workloads that have large working sets of mean memory size demand of 155 MBytes. Fig. 7 presents the total execution times (Fig. 7a) and the queuing times (Fig. 7b) during the executions on a 32-node cluster, where the jobs in workload SPEC are scheduled by either the dynamic load sharing scheme (LS_RE) or the dynamic load sharing supported by the memory reservation scheme (M-Reservation). The trace-driven simulation results show the memory reservation significantly reduced the total execution times and the queuing times and is particularly effective to the workloads with higher job arrival rates. For example, applying memory reservation, we were able to reduce the execution times and the queuing times by 29.3 percent and 24.8 percent, respectively, for workload SPEC-Trace-1, by 32.4 percent and 35.8 percent, respectively, for workload SPEC-Trace-2, by 32.4 percent and 36.7 percent, respectively, for workload SPEC-Trace-3, by 30.3 percent and 34.0 percent, respectively, for workload SPEC-Trace-4, and 27.4 percent and 38.2 percent, respectively, for workload SPEC-Trace-5.

The reduction of the total execution times caused mainly by the reduction of the queuing times effectively reduces the average slowdowns for each trace in the SPEC workload. Fig. 8a presents the comparative average slowdowns

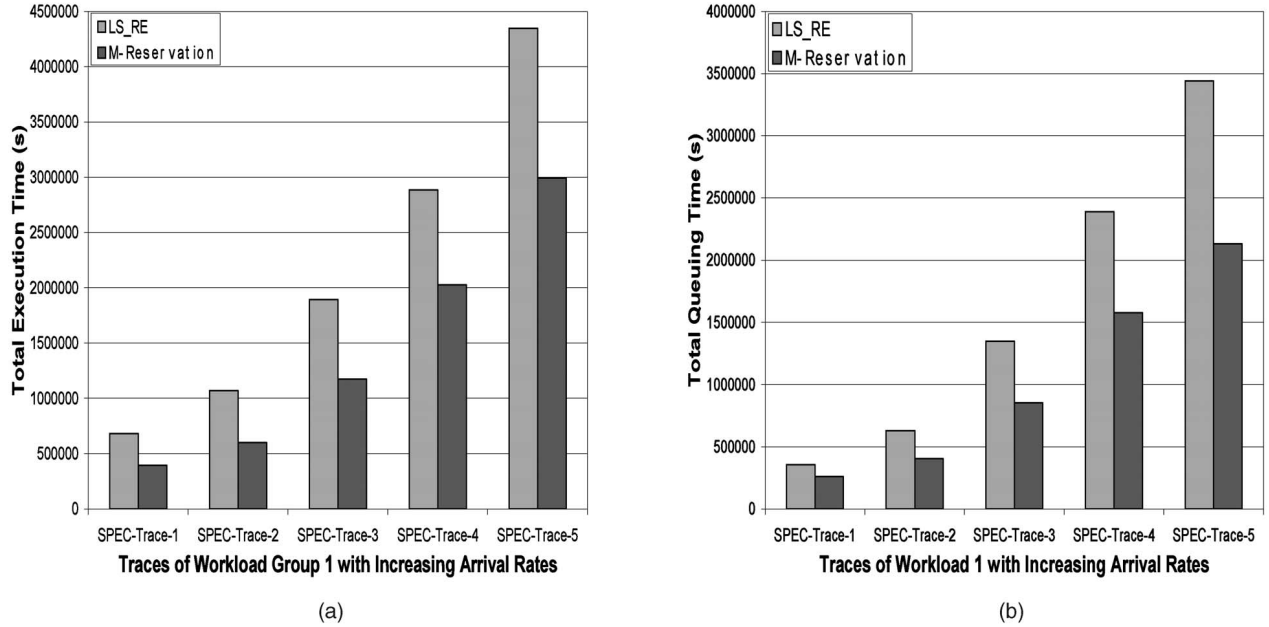

Fig. 7. The total execution times (a) and queuing times (b) of the five traces of workload SPEC on a 32 workstation cluster scheduled LS_RE and M-Reservation.

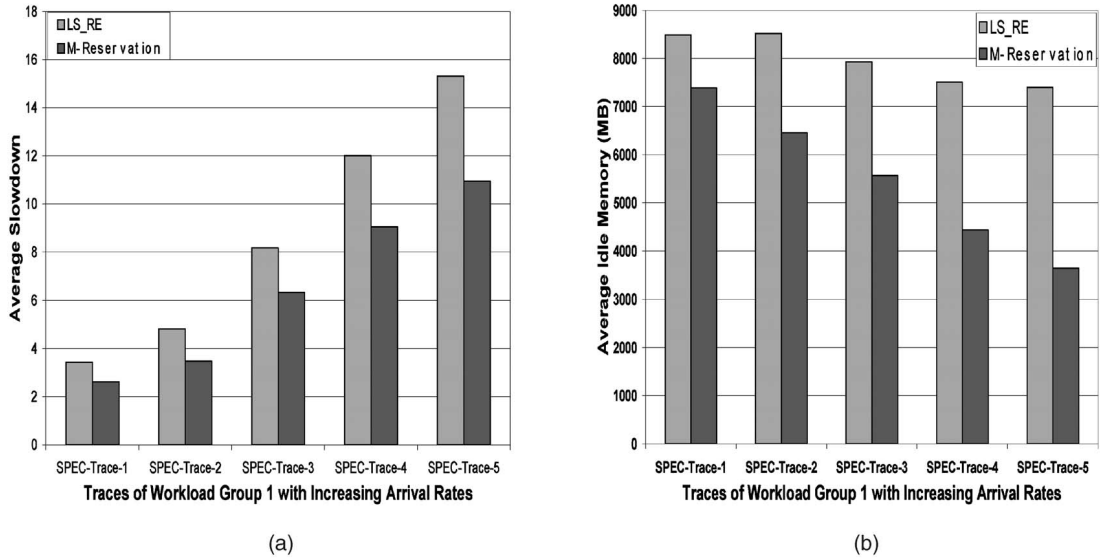

Fig. 8. The average slowdowns (a) and the average idle memory volumes (b) for executing the five traces of the SPEC workload on a 32 workstation cluster scheduled by LS_RE and M-Reservation.

of job executions using LS_RE and M-Reservation. The memory reservation scheme reduced the average slowdowns by 23.4 percent, 27.7 percent, 22.6 percent, 24.6 percent, and 28.46 percent for workloads SPEC-Trace-1, SPEC-Trace-2, SPEC-Trace-3, SPEC-Trace-4, and SPEC-Trace-5, respectively.

We have also observed the average total idle memory volumes during the lifetime of job executions in each workload trace. We collect the total idle memory volume in the cluster every second to calculate the average amount of idle memory space during the entire lifetime. We have repeated the measurements by using several other time intervals, such as 10 seconds, 30 seconds, and 1 minute, and obtained almost identical average values. This implies that the average total idle memory volume is not sensitive to different measurement time intervals. Fig. 8b presents the comparative average idle memory volumes during lifetimes of the 5 SPEC workload traces using LS_RE and M-Reservation. The memory reservation scheme reduced the average idle memory volumes by 12.9 percent, 24.2 percent, 29.7 percent, 40.9 percent, and 50.8 percent for workloads SPEC-Trace-1, SPEC-Trace-2, SPEC-Trace-3, SPEC-Trace-4, and SPEC-Trace-5, respectively. This group of results confirms that the memory reservation scheme can further utilize idle memory space so that the cluster is able to accept (migrate and submit) more jobs and to speed up the job flow in clusters. This is the main reason for significant reductions of average slowdowns in this workload.

6.2 Effective to Heavy Workloads with Moderate Working Set Workloads by Improving Job Balancing for High CPU Utilizations

Fig. 9 presents the total execution times (Fig. 9a) and the queuing times (Fig. 9b) during the executions on a 32-node cluster, where the jobs from the application workload are scheduled by either LS_RE or M-Reservation. The trace-driven simulation results show that M-Reservation can still reduce the total execution times and the queuing times, and is noticeably effective to the workloads of App-Trace-2 and App-Trace-3. For example, applying memory reservation, we were able to reduce the the execution time and the queuing time by 13.4 percent and 16.3 percent, respectively, for workload App-Trace-2, and by 14.0 percent and 16.8 percent, respectively, for workload App-Trace-3. The reductions to other three traces are modest.

The reduction of the total execution times caused mainly by the reduction of the queuing times reduces the average slowdowns for each trace in workload group 2. Fig. 10a presents the comparative average slowdowns of job executions using LS_RE and M-Reservation. M-Reservation effectively reduces the average slowdowns by 33.7 percent, 46.7 percent, and 23.6 percent for workloads App-Trace-2, App-Trace-3, and App-Trace-4, respectively. The average slowdown reductions for workloads App-Trace-1 and App-Trace-5 are modest. Our experiments show that the performance gains mainly come from job balancing from M-Reservation. In fact, the total idle memory volumes are almost the same as those without using M-Reservation. We collect the number of active jobs in each workstation every second to calculate the standard deviation of the number of active jobs among all nonreserved workstations at this moment. This standard deviation gives the job balance skew in each workstation. We further calculate the average job balance skew during the entire lifetime among all nonreserved workstations by first summing all the individual skews and then dividing the sum by the total number of time units. We have repeated the measurements by using several other time intervals, such as 10 seconds, 30 seconds, and 1 minute, and obtained almost identical average values. This implies that the average job balance skew is not sensitive to different measurement time intervals. Fig. 10b presents the comparative average job balance skew during lifetimes of five workload executions (the application traces) using LS_RE and M-Reservation. M-Reservation reduces the average job balance skew by 10.3 percent, 16.5 percent, and 6.3 percent, for workloads App-Trace-2, App-Trace-3, and App-Trace-4, respectively. The two job balance skew differences for traces App-Trace-1 and App-Trace-5 are small. This is why little performance gains are achieved by M-Reservation for these two traces. This group

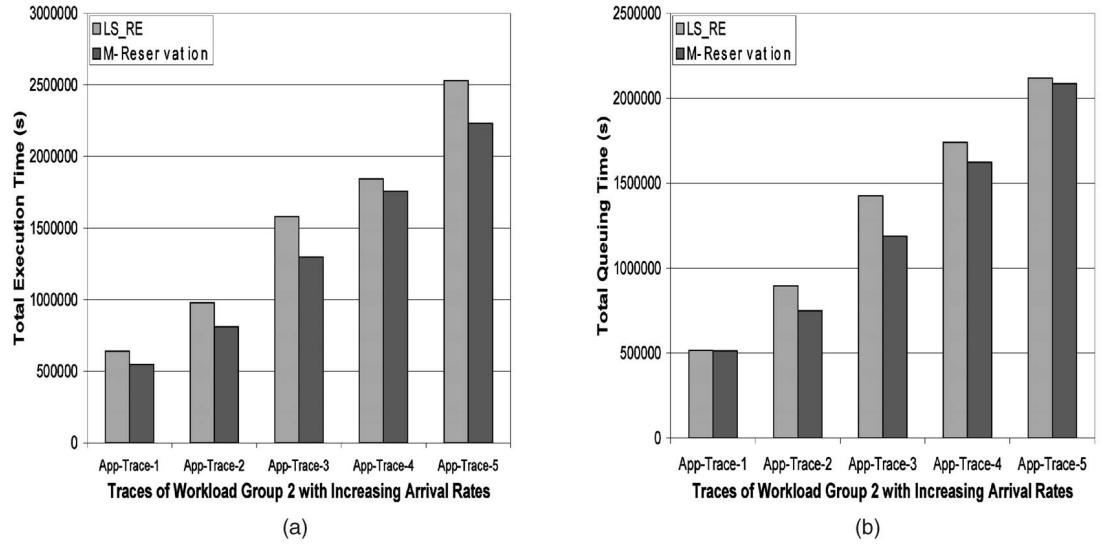

Fig. 9. The total execution times (a) and queuing times (b) of the five traces of the application workload on a 32 workstation cluster scheduled by LS_RE and M-Reservation.

of results indicates that M-Reservation is able to evenly distribute jobs among nonreserved workstations to reduce total queuing time. We have shown that overall performance gains (reductions of total execution times and slowdowns) from improving job balancing is not as significant as that from improving memory utilizations by M-Reservation.

6.3 More Performance Insights on Memory Reservation

The effectiveness and performance insights of the memory reservation scheme will be discussed by simple models in this section. The execution time of job i in a workload for $i = 1, \ldots, n$, $t_{exe}(i)$, is expressed as

$$t_{exe}(i) = t_{cpu}(i) + t_{page}(i) + t_{que}(i) + t_{mig}(i),$$

where $t_{cpu}(i)$, $t_{page}(i)$, $t_{que}(i)$, and $t_{mig}(i)$ are the CPU service time, the paging time for page faults, the queuing time waiting in a job queue, and the migration time if the job is migrated during its execution.

The total execution time of a workload with n jobs, T_{exe}, is expressed as the sum of the total CPU service time (T_{cpu}), total paging time (T_{page}), the total queuing time (T_{que}), and the total migration time (T_{mig}):

$$\begin{aligned} T_{exe} &= \sum_{i=1}^{n} t_{exe}(i) \\ &= \sum_{i=1}^{n} t_{cpu}(i) + \sum_{i=1}^{n} t_{page}(i) + \sum_{i=1}^{n} t_{que}(i) + \sum_{i=1}^{n} t_{mig}(i) \\ &= T_{cpu} + T_{page} + T_{que} + T_{mig}. \end{aligned}$$

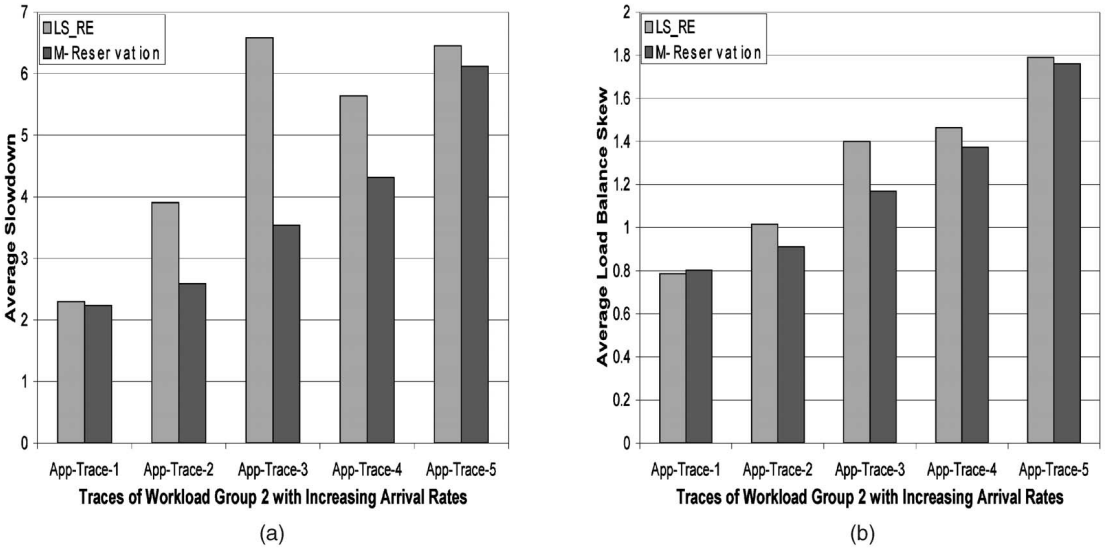

Fig. 10. The average slowdowns (a) and average job balance skews (b) for executing the five traces of the application workload on a 32 workstation cluster scheduled by LS_RE and M-Reservation.

For a given workload with n jobs running on a cluster, we compare the total execution time of the workload without memory reservation, T_{exe}, and the same quantity with memory reservation to resolve the blocking problem, denoted as $\hat{T}_{exe} = \hat{T}_{cpu} + \hat{T}_{page} + \hat{T}_{que} + \hat{T}_{mig}$. The comparison consists of the following four separate models.

1. *CPU service time*. The jobs demand identical CPU services on both cluster environment, so that $T_{cpu} = \hat{T}_{cpu}$.
2. *Paging time*. There will be three possible results: $T_{page} > \hat{T}_{page}$, $T_{page} = \hat{T}_{page}$, and $T_{page} < \hat{T}_{page}$. Paging time reduction ($T_{page} > \hat{T}_{page}$) is the objective of the memory reservation, which can be achieved by migrating jobs with large memory demands to reserved workstations.
3. *Queuing time*. In a cluster with memory reservation, the queuing time consists of two parts:

$$\hat{T}_{que} = \hat{T}_{n-que} + \sum_{k=1}^{m} g(Q_r(k)),$$

where $\hat{T}_{n-que}$ is the queuing time in nonreserved workstations, g is a FIFO queuing function mainly determined by the CPU service time of jobs in the queue of a reserved workstation, m is the number of reserved workstations, and $Q_r(k)$ is the number of jobs in reserved workstation k.

The queuing time in reserved workstation k satisfies

$$g(Q_r(k)) \leq \sum_{j=1}^{Q_r(k)} (Q_r(k) - j) w_{kj},$$

where the arrival order of jobs to workstation k is in an increasing order, i.e., job 1 is the first job arrived in workstation k, and job $Q_r(k)$ is the last arrived job. Variable w_{kj} is the waiting time of job $j+1$ for job j to complete in workstation k. In other words, w_{kj} is the time interval between the arrival time of job $j+1$ and the completion time of job j.
4. *Migration time*. There will be again three possible results: $T_{mig} > \hat{T}_{mig}$, $T_{mig} = \hat{T}_{mig}$, and $T_{mig} < \hat{T}_{mig}$. The migration time is workload and network speed dependent. As high speed networks become widely used in clusters, the migration time in load sharing is only a small portion in the execution time, becoming less crucial for load sharing performance. When $T_{mig} < \hat{T}_{mig}$, the memory reservation needs to sufficiently reduce queuing time to trade off the increase in migration time. Our experiments show that $T_{mig} \approx \hat{T}_{mig}$ because the number of large jobs is very small in job pools.

Using the four portions in the total execution time, considering the paging time reduction ($T_{page} > \hat{T}_{page}$), and assuming the difference between T_{mig} and $\hat{T}_{mig}$ is insignificant in load sharing performance, we examine the potential execution time reduction from the memory reservation:

$$\begin{aligned} T_{exe} - \hat{T}_{exe} &= (T_{cpu} + T_{page} + T_{que} + T_{mig}) \\ &\quad - (\hat{T}_{cpu} + \hat{T}_{page} + \hat{T}_{que} + \hat{T}_{mig}) \\ &\approx (T_{page} - \hat{T}_{page}) + (T_{que} - \hat{T}_{que}) \\ &\geq (T_{page} - \hat{T}_{page}) + (T_{que} - \hat{T}_{n-que} \\ &\quad - \sum_{k=1}^{m} \sum_{j=1}^{Q_r(k)} (Q_r(k) - j) w_{kj}) \\ &> T_{que} - \hat{T}_{n-que} - \sum_{k=1}^{m} \sum_{j=1}^{Q_r(k)} (Q_r(k) - j) w_{kj}. \end{aligned}$$

The above model gives conditions for the memory reservation to effectively reduce the total execution time by resolving the blocking problem. A key condition for performance gains (i.e., the above difference is larger than 0) is that the queuing time in nonreserved workstations ($\hat{T}_{n-que}$) is significantly smaller than the queuing time in all workstations without memory reservation (T_{que}) because jobs are more evenly distributed with memory reservation. Since time quantum $\sum_{k=1}^{m} \sum_{j=1}^{Q_r(k)} (Q_r(k) - j) w_{kj}$ in reserved workstations include CPU service times, and no page faults due to memory shortage will be conducted, the queuing time in the reserved workstations are minimized if $w_{k1} < w_{k2} \ldots < w_{kQ_r(k)}$. This is likely to be achieved because only a small portion of jobs are large ones.

The model also indicates that memory reservation can be potentially unsuccessful with the following conditions:

1. The cluster is lightly loaded, and moderate page faults in each node can be effectively reduced by dynamic load sharing. This is because the possibility to trigger the memory reservation due to the blocking problem is low.
2. Majority jobs in the workload are equally sized in their memory demands. Again, possibility to trigger the memory reservation due to the blocking problem is low, assuming that the accumulated global memory storage is sufficiently large.
3. If the memory allocation size of a migrated job to a reserved workstation is larger than the available user space in the reserved workstation, page faults may increase the job queuing time in a reserved node. If this makes $T_{exe} > \hat{T}_{exe}$, memory reservation fails.

The concern in the first condition has been addressed by adaptively reserving workstations, where the memory reservation is not initiated until the blocking problem is detected. In practice, our experiments have shown that the memory demands of jobs in a workload are seldom equally sized. Thus, the second concern should not be a base for only using a general purpose load sharing system to cover all the cases. The third concern can be addressed by selecting the workstations with large user memory space as the reserved workstations, and by applying Network RAM to a reserved workstation to further enhance its memory size.

7 Comparisons between the Two Schemes

The two group workloads have different characteristics: The SPEC workloads have large working sets (the mean memory demand is 155 MBytes) with dynamic request changes, while the application workloads have moderate working sets (the mean memory demand is 37 MBytes) with relative stable memory allocation requests. Both job

submission rates of both group workloads vary from low, moderate, to high. Our trace-driven simulations show that the Network RAM supported load sharing scheme outperforms the memory reservation scheme for the application workloads, and the memory reservation scheme outperforms the other scheme for the SPEC workloads. Comparing Fig. 5b with Fig. 9a, we show that the two schemes perform almost the same for workloads APP-Trace-1, APP-Trace-2, and APP-Trace-3 (low and moderate submission rates). However, the Network RAM supported load sharing scheme has execution time reductions of 17 percent and 8 percent for App-Trace-4 and APP-Trace-5 (traces with high job submission rates), respectively, normalized from the execution times of the memory reservation scheme. On the other hand, comparing Fig. 6b with Fig. 7a, we show that the memory reservation scheme has execution time reductions of 38 percent, 40 percent, 19 percent, 17 percent, and 16 percent for SPEC-Trace-1, SPEC-Trace-2, SPEC-Trace-3, SPEC-Trace-4, and SPEC-Trace-5, respectively, normalized from the execution times of the Network RAM supported load sharing scheme. These results are consistent with our analysis in previous sections. The memory reservation scheme is more effective to resolve the blocking problem than the Network RAM supported load sharing scheme, which can be easily caused by the heavy workloads of large working sets with dynamically changing memory demands. In contrast, the Network RAM supported load sharing scheme can efficiently handle a large number of jobs with moderate working sets, where the blocking problem may not be a serious concern.

Here, we further summarize the merits and limits of each scheme to show how they are mutually complimentary to each other. The Network RAM supported load sharing scheme has two advantages. First, since a job migration can vary timely, the rest of the jobs in an overloaded workstation can be relieved quickly. Second, if the memory space is sufficiently large in the identified workstation for a job migration, the migrated job will be efficiently executed there. If the memory space is not enough, the network RAM will take care of it. A major limit of this scheme is that a job migration with a large memory demand can seriously affect the current jobs in the identified workstation, although Network RAM can utilize the global memory of a cluster. This is because the workstation is not dedicated to the migrated large job. It is still open to CPU intensive job submissions and migrations.

The memory reservation scheme has the advantage of utilizing a few memory servers to provide a dedicated environment for unexpectedly large data-intensive jobs. However, since the "trouble-maker job" cannot be immediately migrated during the "transition period," the performance of other jobs in the heavily loaded workstation can be seriously affected.

8 Conclusion

Accommodating expected and unexpected workload fluctuations of service demands is highly desirable in cluster computing. Existing studies indicate that even load sharing schemes that dynamically assign and schedule resources are not able to fully utilize the available resources. We make three contributions in this study.

1. We present the conditions to cause the job blocking problem.
2. We present two schemes to address the job blocking problem for workloads with different resource demands under different system conditions.
3. Our trace-driven simulation experiments and analysis show that the proposed schemes effectively improve cluster resource utilization to resolve job blocking problems, resulting in significant performance gain.

The deployment of the two schemes in cluster systems relies on available system environment. If global memory space in the cluster can be widely shared by Network RAM and the increased network traffic is not a concern for applications, the network RAM supported load sharing scheme is an optimal choice. On the other hand, if a few workstations with large memory space are available in the cluster and jobs are sensitive to network bandwidths, the memory reservation scheme should be used. Since the two schemes are complimentary to each other, we can also keep the both schemes in the system, and switch between them adaptively in practice.

Acknowledgments

The authors appreciate the critical and constructive comments from the three anonymous referees. They thank Phil Kearns for providing them a kernel programming environment. They appreciate Bill Bynum for reading the paper and for his suggestions. This work is supported in part by the US National Science Foundation under grants CCR-0098055 and ACI-0129883, and by a Usenix Association Research Fellowship. The work is also a part of an independent research project sponsored by the US National Science Foundation for its program directors and visiting scientists. The preliminary results of this work have been presented in [5] and [22].

References

[1] A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Workstation Clusters," *Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems,* pp. 35-46, May 1999.

[2] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, and A. Keren, "An Opportunity Cost Approach for Job Assignment and Reassignment in a Scalable Computing Cluster," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 7, pp. 760-768, July 2000.

[3] A. Barak and A. Braverman, "Memory Ushering in a Scalable Computing Cluster," *J. Microprocessors and Microsystems,* vol. 22, nos. 3-4, pp. 175-182, Aug. 1998.

[4] A. Batat and D.G. Feitelson, "Gang Scheduling with Memory Considerations," *Proc. 14th Int'l Parallel and Distributed Processing Symp.,* pp. 109-114, May 2000.

[5] S. Chen, L. Xiao, and X. Zhang, "Adaptive and Virtual Reconfigurations for Effective Dynamic Job Scheduling in Clusters," *Proc. 22nd Int'l Conf. Distributed Computing Systems,* pp. 35-42, July 2002.

[6] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* pp. 63-72, May 1988.

[7] Y. Etsion, D. Tsafrir, and D.G. Feitelson, "Effects of Clock Resolution on the Scheduling of Interactive and Soft Real-Time Processes," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* pp. 172-183, June 2003.

[8] M.J. Feeley et al., "Implementing Global Memory Management Systems," *Proc. 15th ACM Symp. Operating System Principles,* pp. 201-212, Dec. 1995.

[9] M.D. Flouris and E.P. Markatos, "Network RAM," *High Performance Cluster Computing*, chapter 16, vol. 1, pp. 383-508, R. Buyya, ed., Prentice Hall, 1999.
[10] D.G. Feitelson and B. Nitzberg, "Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860," *Job Scheduling Strategies for Parallel Processing*, Springer, pp. 337-360, 1995.
[11] M. Harchol-Balter and A.B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," *ACM Trans. Computer Systems*, vol. 15, no. 3, pp. 253-285, 1997.
[12] S. Jiang and X. Zhang, "TPF: A System Thrashing Protection Facility in Linux," *Software: Practice and Experience*, vol. 32, no. 3, pp. 295-318, 2002.
[13] V. Karamcheti and A. Chien, "A Hierarchical Load-Balancing Framework for Dynamic Multithreaded Computations," *Proc. Supercomputing Conf.*, Nov. 1998.
[14] E.P. Markatos and G. Dramitinos, "Implementation of a Reliable Remote Memory Pager," *Proc. 1996 Usenix Technical Conf.*, pp. 177-190, Jan. 1996.
[15] L.E. Schrage, "A Proof of the Optimality of the Shortest Processing Remaining Time Discipline," *Operational Research*, vol. 16, pp. 678-690, 1968.
[16] S. Setia, "The Interaction between Memory Allocation and Adaptive Partitioning in Message-Passing Multicomputers," *Proc. IPPS Workshop Job Scheduling Strategies for Parallel Processing*, pp. 146-164, 1995.
[17] S. Setia, M.S. Squillante, and V.K. Naik, "The Impact of Job Memory Requirements on Gang-Scheduling Performance," *Performance Evaluation Rev.*, Mar. 1999.
[18] A. Silberschatz and P.B. Galvin, *Operating Systems Concepts*, fourth ed. Addison-Wesley, 1994.
[19] M.S. Squillante, D.D. Yao, and L. Zhang, "Analysis of Job Arrival Patterns and Parallel Scheduling Performance," *Performance Evaluation*, vols. 36-37, pp. 137-163, 1999.
[20] A. Wierman and M. Harchol-Balter, "Classifying Scheduling Policies with Respect to Unfairness in an M/GI/1," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pp. 238-249, June 2003.
[21] L. Xiao, S. Chen, and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 3, pp. 223-240, 2002.
[22] L. Xiao, X. Zhang, and S.A. Kubricht, "Incorporating Job Migration and Network RAM to Share Cluster Memory Resources," *Proc. Ninth IEEE Int'l Symp. High Performance Distributed Computing*, pp. 71-78, Aug. 2000.
[23] X. Zhang, Y. Qu, and L. Xiao, "Improving Distributed Workload Performance by Sharing Both CPU and Memory Resources," *Proc. 20th Int'l Conf. Distributed Computing Systems*, pp. 233-241, Apr. 2000.

Li Xiao received the BS and MS degrees in computer science from Northwestern Polytechnic University, China, and the PhD degree in computer science from the College of William and Mary in 2002. She is an assistant professor of computer science and engineering at Michigan State University. She is a recipient of the USENIX Fellowship for her PhD dissertation research from 2001 to 2002. Her research interests are in the areas of distributed and Internet systems, system resource management, and design and implementation of experimental algorithms. She is a member of ACM and the IEEE Computer Society.

Songqing Chen received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1997 and 1999, respectively. He is a PhD candidate in computer science at the College of William and Mary. His research interests are distributed and Internet systems, and kernel systems programming. He is a student member of IEEE Computer Society and ACM.

Xiaodong Zhang received the BS degree in electrical engineering from Beijing Polytechnic University in 1982, and the MS and PhD degrees in computer science from the University of Colorado at Boulder in 1985 and 1989. He is the Lettie Pate Evans Professor of computer science and the department chair at the College of William and Mary. He was the program director of advanced computational research at the US National Science Foundation from 2001 to 2003. He is a past member of the editorial board of *IEEE Transactions on Parallel and Distributed Systems*, and currently serves as an associate editor of *IEEE Micro*. His research interests are in the areas of parallel and distributed computing and systems, and computer architecture. He is a senior member of the IEEE and the IEEE Computer Society.