Proceedings of 2000 International Parallel and Distributed Processing Symposium (IPDPS'2000), Cancun, Mexico, May 1-5, 2000

Effective Load Sharing on Heterogeneous Networks of Workstations *

Li Xiao[1] Xiaodong Zhang[1] Yanxia Qu[2]

[1]Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795
{lxiao, zhang}@cs.wm.edu

[2]Bell Laboratories, Lucent Technologies
Holmdel, NJ 07733-3030
yqu@dnrc.bell-labs.com

Abstract

We consider networks of workstations which are not only time-sharing, but also heterogeneous with a large variation in the computing power and memory capacities of different workstations. Many load sharing schemes mainly target sharing CPU resources, and have been intensively evaluated in homogeneous distributed environments. However, the penalties of data accesses and movement in modern computer systems, such as page faults, have grown to the point where the overall performance of distributed systems cannot be further improved without serious considerations concerning memory resources in the design of load sharing policies. Considering both system heterogeneity and effective usage of memory resources, we design and evaluate load sharing policies in order to minimize both CPU idle times and the number of page faults in heterogeneous distributed systems. Conducting trace-driven simulations, we show that load sharing policies considering both CPU and memory resources are robust and effective in heterogeneous systems. We also show that the functionality and the nature of load sharing policies are quite independent on several memory demand distributions of workloads.

1 Introduction

Most load sharing schemes proposed for distributed systems (e.g., [5], [6], [7], [12], [13] [14], [19]) mainly consider effectively sharing CPU cycles. We have designed CPU-Memory-based load sharing schemes and shown their effectiveness by trace-driven simulations in a homogeneous environment [17]. In this paper, we present our continued effort and report performance results of extending the load sharing design on heterogeneous distributed systems.

Some work has been reported on global memory resource management to reduce page faults by paging remote memory space in a distributed system (e.g., [1], [8], and [16]). In contrast, our approach is to use migrations to find a node with sufficient memory space and CPU cycles for a job to execute. A load sharing scheme in MOSIX also uses the migration approach [2] for memory load sharing. However, it uses preemptive migrations only, and is evaluated on a homogeneous environment.

A static scheduling scheme which considers only CPU heterogeneity to coordinate a parallel job and local user processes is proposed in [6]. Researchers have also made efforts to schedule parallel programs, to conduct efficient communications, and to predict performance on heterogeneous systems (e.g., [3], [9], [10], [18]).

In this study, we address the following four questions: (1) what is an effective way to quantitatively incorporate system information related to processor and memory heterogeneity in the design of load sharing policies? (2) what are the effects of system heterogeneity on performance for different load sharing policies? (3) Why are some load sharing policies sensitive to the system heterogeneity? and (4) Do memory demand distributions of workloads fundamentally affect the performance of load sharing policies? These considerations are important additions in designs and evaluation of load sharing policies for memory-intensive workloads on networks of heterogeneous workstations.

Applying a concept of CPU and memory weights, we quantitatively distinguish the computing capability of CPUs and memory capacities among heterogeneous workstations. The load index of each computing node is uniquely represented by its CPU weight and its memory size. We have conducted trace-driven simulations to compare load sharing policies for CPU load sharing and our policies for both CPU and memory sharing in heteroge-

*This work is supported in part by the National Science Foundation under grants CCR-9400719 and CCR-9812187, and EIA-9999030, by the Air Force Office of Scientific Research under grant AFOSR-95-1-0215, and by Sun Microsystems under grant EDUE-NAFO-980405.

neous systems. We show that the overall performance of a distributed system and individual jobs on the system can be further improved by considering memory resources as an additional and important factor in load sharing policies. We show that our CPU-Memory-based load sharing policies are robust and effective in heterogeneous systems. We also show that the functionality and the nature of load sharing policies are quite independent on several memory demand distributions of workloads.

2 CPU/Memory Weights and Heterogeneity

In this paper, heterogeneity only refers to the variations of CPU powers and memory capacities, but not the variations of operating systems, network interfaces and hardware organizations among the workstations. In this section, we quantitatively characterize heterogeneous CPU powers and memory capacities in a network of workstations. The simple models to be discussed here will be used in the designs and evaluation of load sharing policies in the rest of the sections. We use node index j to represent one of the nodes in a heterogeneous network of workstations. We also use variable P to represent the total number of nodes in the system.

The CPU weight of a workstation refers to its computing capability relative to the fastest workstation in a distributed system. The value of the CPU weight is less than or equal to 1. Since the CPU weight is a relative ratio, it can also be represented by the CPU speed of each node measured by millions of instructions per second (MIPS). If $V_{cpu}(j)$ is the speed of workstation M_j in MIPS, $j = 1, ..., P$, the CPU weight can be expressed as follows:

$$W_{cpu}(j) = \frac{V_{cpu}(j)}{\max_{i=1}^{P} V_{cpu}(i)}, \tag{1}$$

The total CPU power of a system is defined as the sum of the CPU speed of each workstation, which represents the accumulated computing capability of the system:

$$TP_{cpu} = \sum_{j=1}^{P} V_{cpu}(j). \tag{2}$$

Similarly, the memory weight is calculated by comparing the memory sizes among the computing nodes:

$$W_{mem}(j) = \frac{RAM_j}{\max_{i=1}^{P} RAM_i}, \tag{3}$$

where RAM_j is the amount of user available memory space on node j for $j = 1, ..., P$.

The total memory capacity of a system is defined as

$$TP_{mem} = \sum_{j=1}^{P} MS_j. \tag{4}$$

where MS_j is the memory size of node j.

The system heterogeneity can be quantified as the variance of computing powers and memory capacities among the workstations. Using standard deviation and CPU weights, we define the CPU heterogeneity as follows:

$$H_{cpu} = \sqrt{\frac{\sum_{j=1}^{P}(\overline{W}_{cpu} - W_{cpu}(j))^2}{P}}, \tag{5}$$

where $\overline{W}_{cpu} = \frac{\sum_{j=1}^{P} W_{cpu}(j)}{P}$, is the average CPU weight. Similarly, we define memory heterogeneity as follows:

$$H_{mem} = \sqrt{\frac{\sum_{j=1}^{P}(\overline{W}_{mem} - W_{mem}(j))^2}{P}}, \tag{6}$$

where $\overline{W}_{mem} = \frac{\sum_{j=1}^{P} W_{mem}(j)}{P}$, is the average memory weight in the system. Higher values of H_{cpu} and H_{mem} in a distributed system correspond with a higher variation in the CPU capability and memory capacity among different nodes. A homogeneous system is characterized by $H_{cpu} = H_{mem} = 0$.

3 Load sharing in heterogeneous systems

Here are two terms to be used in a multiprogrammed environment on the jth node for $j = 1, ...P$:

- MT_j: the memory threshold in bytes is the total amount of memory thresholds accumulated from the running jobs on the computing node. (After a job executes in its stable stage, its working set size should also be stable. The memory space for the stable working set is called the memory threshold of the job [15]. If $RAM_j > MT_j$, page faults would rarely occur, otherwise, paging would be frequently conducted during the executions of jobs in the node.)
- σ_j: the average page fault rate caused by all jobs on a computing node is measured by the number of page faults per million instructions when the allocated memory space equals the memory threshold.

When a job migration is necessary in load sharing, the migration can be either a remote execution which makes jobs be executed on remote nodes in a non-preemptive way, or a preemptive migration which may suspend the selected jobs, move them to a remote node, and then restart them. We have compared the performance between the remote executions and preemptive migrations for load sharing in a homogeneous environment [17]. Our study indicates that an effective preemptive migration for a memory-intensive workload is not only affected by the workload's lifetime length, but also by its data access patterns. Without a thorough understanding of workloads' execution patterns interleaving among the CPU, the memory and the I/O, it is difficult to effectively use preemptive migrations in load sharing policies. For this reason, we have decided to only use the remote execution strategy in this study. Our study in this paper focuses on the three policies using remote executions: the first based on CPU resource information, the second using information on memory usage, and the third based on data concerning both CPU and memory resources. Brief descriptions of the three policies are given as follows.

CPU-based load sharing. The load index in each computing node is represented by the length of the CPU waiting queue, L_j. A CPU threshold on node j, denoted as CT_j, is set based on the CPU computing capability. For a new job requesting service in a computing node, if the waiting queue is shorter than the CPU threshold ($L_j < CT_j$), the job is executed locally. Otherwise, the load sharing system finds the remote node with the shortest waiting queue to remotely execute this job. This policy is denoted as CPU_RE.

memory-based load sharing. Instead of using L_j, we propose to use the memory threshold, MT_j to represent the load index. For a new job requesting service in a computing node, if the node memory threshold is smaller than the user memory space ($MT_j < RAM_j$), the job is executed locally. Otherwise, the load sharing system finds the remote node with the lightest memory load to remotely execute this job. This policy is denoted as MEM_RE.

CPU-memory-based load sharing. We have proposed a load index which considers both CPU and memory resources. The basic principle is as follows. When a computing node has sufficient memory space for both running and requesting jobs, the load sharing decision is made by a CPU-based policy. When the node does not have sufficient memory space for the jobs, the system will experience a large number of page faults, resulting in long delays for each job in the node. In this case, a memory-based policy makes the load sharing decision to either migrate jobs to suitable nodes or to hold the jobs in a waiting pool if necessary.

The load index of node j ($j = 1, ..., P$) combining the resources of CPU cycles and memory space is defined as

$$Index(j)(L_j, MT_j) = \begin{cases} L_j, & MT_j < RAM_j, \\ CT_j, & MT_j \geq RAM_j. \end{cases}$$

When $MT_j < RAM_j$, CPU-based load sharing is used. When $MT_j \geq RAM_j$, the CPU queue length (the load index) is set to CT_j as if the CPU were overloaded so that the system refuses to accept jobs. In our implementation, when $MT_j \geq RAM_j$, the local scheduler immediately searches for the most lightly loaded node in the system as the job's destination. The load sharing policy can be expressed as follows:

$$LS(Index(j)) = \begin{cases} \texttt{local execution}, & Index(j) < CT_j, \\ \texttt{remote execution}, & Index(j) \geq CT_j. \end{cases}$$

This policy is denoted as CPU_MEM_RE.

4 Performance Evaluation Methodology

4.1 A simulated heterogeneous distributed system

Harchol-Balter and Downey [12] developed a simulator of a homogeneous distributed system with 6 nodes, where each local scheduler is CPU-based only. We expanded the capability of this simulator by adding the function of system heterogeneity and by implementing the CPU-based, Memory-based, and CPU-Memory-based load sharing policies using remote executions. The simulated heterogeneous system is configured with parameters listed in Table 1.

The parameter values in Table 1 are similar to the ones of Sun SPARC-20, Sun Ultra 5 and Sun Ultra 10 workstations. The remote execution overhead matches the 10 Mbps Ethernet network service times for the Sun workstations. We have also conducted experiments on an Ethernet network of 100 Mpbs. However, compared with the experiments on the 10 Mbps Ethernet, we found that little performance improvement was gained for all policies. Although the remote execution overhead is reduced by the faster Ethernet, it is still an insignificant factor for performance degradation in our experiments. The page fault overhead is the dominant factor.

CPU speeds	100 to 500 MIPS
memory sizes (MS_j)	32 to 256 MBytes
kernel and file cache usage (U_{sys})	16 MBytes [4]
user available space (RAM_j)	$MS_j - U_{sys}$
memory page size	4 KBytes
Ethernet speed	10 Mbps
page fault service time	10 ms
CPU time slice for a job	10 ms
context switch time	0.1 ms
overhead of a remote execution	0.1 sec

Table 1: Parameters used for the simulated heterogeneous network of workstations where the CPU speed is represented by Millions Instruction Per Second (MIPS).

The CPU local scheduling uses the round-robin policy. Each job is in one of the following states: "ready", "execution", "paging", "data transferring", or "finish". When a page fault happens in the middle of a job execution, the job is suspended from the CPU during the paging service. The CPU service is switched to a different job. When page faults happen in executions of several jobs, they will be served in FIFO order.

4.2 System conditions

We have the following conditions and assumptions for evaluating the load sharing policies in the distributed system:

- The CPU load index in each node is calculated by

$$L_j \times \frac{1}{W_{cpu}(j)}, \quad j = 1, ..., P,$$

  where L_j is the number of jobs queued in node j, $W_{cpu}(j)$ is the CPU weight of the node, and P is the total number of nodes in the system.

- Each computing node maintains a global load index file which contains both the CPU and memory load status information of other computing nodes. The load sharing system periodically collects and distributes the load information among the computing nodes. Since only 6 nodes are used in our system, the global information collection cost is negligible. The acceptance of this assumption has also been experimentally validated in [20].

- The location policy determines which computing node is selected for a job execution. The policy we use is to find the most lightly loaded node in the distributed system.

- We assume that the memory allocation for a job is done at the arrival of the job.

- Similar to the assumptions in [11] and [16], we assume that page faults are uniformly distributed throughout job executions.
- We assume that the memory threshold of a job is 40% of its requested memory size. The practical value of this threshold assumption has also been confirmed by the studies in [11] and [16].

4.3 Workloads

The workloads we have used are the 8 traces from [12]. Each trace was collected from one workstation on different daytime intervals. The jobs in each trace were distributed among 6 homogeneous workstations. We have done the following modifications of the traces for our study. We converted the job duration time into Million Instructions according to the CPU speed. The duration time only represents the CPU service time. The requested memory size of each job in the traces is generated from a Pareto distribution with the mean sizes of 1 MBytes, 2 MBytes, 4 MBytes, and 8 MBytes. Each job has the following 4 items:

$<$ `arrival time, arrival node, requested memory size, duration time` $>$

The page faults in each node are conducted in our simulation as follows. When the memory threshold of jobs in node j is equal to or larger than the available memory space of the node ($MT_j \geq RAM_j$), each job in the node will cause page faults at a given page fault rate. The page fault rates of jobs range from 0 to 4.0 per million instructions in our experiments. In practice, application jobs have page fault rates from 1 to 10.

5 Experimental Results and Analysis

Using the trace-simulation on the distributed heterogeneous system of 6 nodes, we have evaluated the performance of the following load sharing policies: CPU_RE, MEM_RE, and CPU_MEM_RE. In addition, we have also compared execution performance of the above policies with the execution performance without using load sharing, denoted as NO_LS.

A major timing measurement we have used is the mean *slowdown*, which is the ratio between the total wall-clock execution time of all the jobs in a trace and the their total CPU execution time. Major contributions to the slowdown come from the delays of page faults, waiting time for CPU service, and the overhead of migration and remote execution.

5.1 Effects of system heterogeneity

We first examined the effects of system heterogeneity to load sharing policies, to memory demands of workloads, and to the overall performance. For a given total CPU power (TP_{cpu}) in MIPS, a given total memory capacity (TP_{mem}) in MBytes, and a given number of nodes (P), we can construct a homogeneous system and several choices of heterogeneous systems. The total CPU power and the total memory capacity we selected for a network of workstations are 1,800 MIPS and 768 MBytes. The

node	1	2	3	4	5	6	heterogeneity
MIPS	300	300	300	300	300	300	$H_{cpu}(1) = 0.0$
MB	128	128	128	128	128	128	$H_{mem}(1) = 0.0$
MIPS	300	400	300	200	300	300	$H_{cpu}(2) = 0.14$
MB	128	128	128	128	128	128	$H_{mem}(2) = 0.0$
MIPS	500	500	200	200	200	200	$H_{cpu}(3) = 0.28$
MB	256	256	64	64	64	64	$H_{mem}(3) = 0.38$
MIPS	500	100	500	100	500	100	$H_{cpu}(4) = 0.4$
MB	256	32	256	32	128	64	$H_{mem}(4) = 0.4$

Table 2: The 4 platforms based on the total power of 1,800 MIPS and the total memory capacity of 768 MBytes. Each platform consists of 6 nodes. Both CPU and memory heterogeneities, ($H_{cpu}(i)$ and $H_{mem}(i)$, $i = 1, 2, 3, 4$), are also calculated for each platform.

homogeneous system based on the total CPU power and the total memory capacity is a network of 6 workstations where each node has 300 MIPS computing capability and 128 MBytes memory capacity. Table 2 lists all the configurations we used in performance evaluation for the given total CPU power and the given total memory capacity. Each of the 6 nodes is represented by its MIPS and MBytes. The system heterogeneities represented by H_{cpu} and H_{mem} of each platform are also calculated based on the formulas in Section 2. The 4 platforms are sorted by both CPU and memory heterogeneities, from homogeneous platform 1 ($H_{cpu}(1) = H_{mem}(1) = 0.0$) to highest heterogeneous platform 4 ($H_{cpu}(4) = 0.4$, $H_{mem}(4) = 0.4$).

We first concentrate on the performance comparisons of "trace 0" in different directions, and will present performance comparisons of all the traces after this focused study.

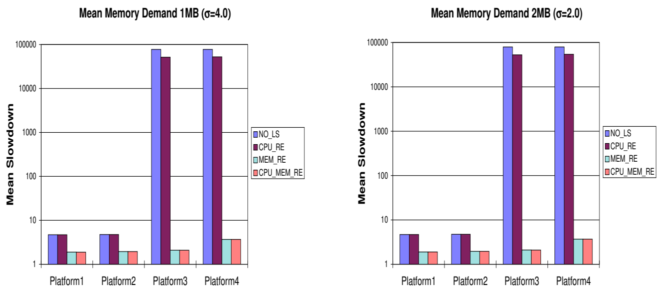


Figure 1: Mean slowdowns of "trace 0" on different heterogeneous platforms, with mean memory size of 1 MBytes (left) and 2 MBytes (right).

Figures 1 and 2 present the mean slowdowns of "trace 0" scheduled by the three load sharing policies on platforms 1, 2, 3, and 4, with the mean memory demand of 1 MBytes, 2 MBytes, 4 MBytes and 8 MBytes. We show that policies NO_LS and CPU_RE perform poorly, and policies MEM_RE and CPU_MEM_RE perform well on all the platforms with different sizes of memory demand. Load sharing decisions are made only

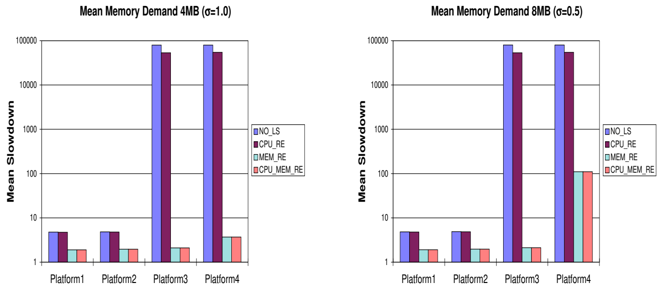


Figure 2: Mean slowdowns of "trace 0" on different heterogeneous platforms, with mean memory size of 4 MBytes (left) and 8 MBytes (right).

by considering the CPU resources in CPU_RE, which will not be beneficial to memory-intensive workloads. As the system heterogeneity increases, load sharing performance is affected. For example, as the heterogeneity moderately increases from platform 1 (homogeneous system) to platform 2 ($H_{cpu}(2) = 0.14$ and $H_{mem}(2) = 0$), the mean slowdowns of the workloads with different sizes of mean memory demand slightly increase. However, as the system heterogeneity further increases on platform 3 ($H_{cpu}(3) = 0.28$, and $H_{mem}(3) = 0.38$), the mean slowdowns for NO_LS and CPU_RE increase more than 10,000 times. In contrast, the mean slowdowns for MEM_RE and CPU_MEM_RE only increase to less than 2 times. On platform 4 ($H_{cpu}(4) = 0.4$ and $H_{mem}(4) = 0.4$), our experiments show the similar results except for the mean memory demand size of 8 MBytes. The mean slowdowns of the workloads with mean memory demand of 8 MBytes for MEM_RE and CPU_MEM_RE on platform 4 significantly increase compared with their slowdowns on platform 3. However, the slowdowns are still much lower than the slowdowns for NO_LS and CPU_RE.

Why are policies MEM_RE and CPU_MEM_RE much less sensitive to the system heterogeneity than the CPU-based policy? We give following reasons to answer this question. First, the workloads we used are memory-intensive. Thus, memory load sharing is more effective than CPU load sharing. Second, in a heterogeneous system, the job pool in a less powerful node in terms of CPU or/and memory capability may initially be assigned a larger number of jobs than a more powerful node. The local scheduler using the CPU-based policy in the less powerful node could easily make wrong decisions by scheduling more jobs than the node could allocate in its limited memory space, causing frequent page faults which would slow down the executions. In contrast, policies MEM_RE and CPU_MEM_RE will identify less powerful nodes that lack sufficient memory to serve jobs. The local scheduler on the weaker nodes can then migrate some jobs to execute in remote nodes with low memory allocations. By correctly making the migration decisions, policies MEM_RE and CPU_MEM_RE are able to offset system heterogeneity effects. Finally, compared with a homogeneous system, jobs of the same workload are likely to move around in a more dynamic manner on a heterogeneous system. However, even for policies MEM_RE and CPU_MEM_RE (which make beneficial migration decisions for memory-intensive workloads), the overhead of remote executions and page faults still increase proportionally to the increase in the heterogeneity and mean memory demand.

As the mean memory demand increases, the size of each job in a workload also increases. It is more likely for a job with a large memory demand to be scheduled for a remote execution. On the other hand, it is also possible that none of the nodes in the system could find sufficient memory space for such a job. In this case, the job will be scheduled to run in the most lightly loaded node where it will suffer the page faults. Therefore, for workloads with a large mean memory demand, increases of slowdowns for policies MEM_RE and CPU_MEM_RE on a highly heterogeneous system (such as platform 4) are mainly caused by the increased amount of remote executions and additional page faults. For example, for policies CPU_MEM_RE on "trace 0" with a memory demand of 8 MBytes at $\sigma = 0.5$, the total numbers of migrations on platforms 1, 2, and 3, are 18, 17 and 15, respectively. The number of migrations increases to 51 on platform 4. The total numbers of page faults are 8,339, 8,168, and 16,156 on platforms 1, 2, and 3 respectively. The number of page faults increases to 213,796 on platform 4.

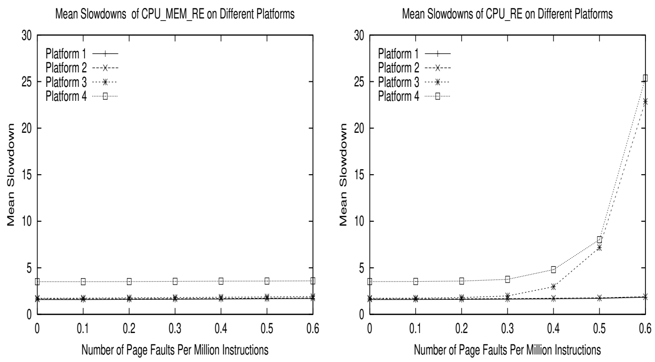
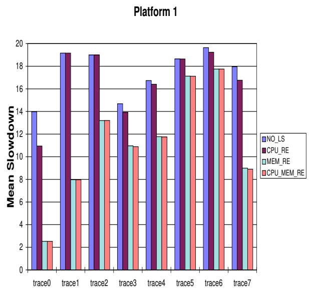


Figure 3: Mean slowdowns as the page fault rate increases on "trace 0" on different platforms, with mean memory size of 4 MBytes scheduled by policy CPU_MEM_RE (left) and CPU_RE (right).

We also evaluate the heterogeneity effects to load sharing performance as the page fault rate increases. Figure 3 compares the slowdowns for CPU_MEM_RE (left figure) and for CPU_RE (right figure) as the page fault rate increases on "trace 0" with mean memory demand of 4 MBytes on the homogeneous platform, and on three other heterogeneous platforms. The experiments show that as the heterogeneity measured by H_{cpu} and H_{mem} increases to a certain degree, the slowdown also increases for the same load sharing policies. For example, the mean slowdown of CPU_MEM_RE on platform 4 ($H_{cpu}(4) = 0.4$, and $H_{mem}(4) = 0.4$) is 2 times higher than that on the homogeneous platform (platform 1) for page fault rates ranging from 0 to 0.6. In contrast, the mean slowdown of CPU_RE on platform 4 is 2 times higher than that on platform 1 for page fault rates ranging from 0 to 0.4. But this gap increases to 14 times as the page fault rate further increases to 0.6. We show that a slight increase of page fault rates makes the CPU-based policy even more sensitive to the system heterogeneity.

5.2 Performance comparisons of all the traces

The performance comparisons of the load sharing policies on other traces are consistent with what we have presented for "trace 0" in principle. Figures 4 and 5 present the mean slowdowns of all the traces ("trace 0", ..., "trace 7") scheduled by the three load sharing policies with the mean memory demand of 4 MBytes on platforms 1 and 2, platforms 3 and 4, respectively.



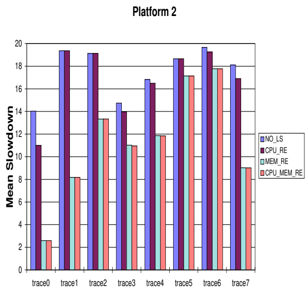
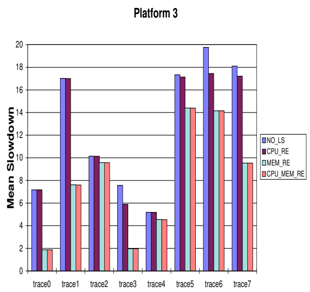


Figure 4: Mean slowdowns of all the 8 traces scheduled by different load sharing policies with the mean memory demand of 4 MBytes on platform 1 (left) and platform 2 (right).



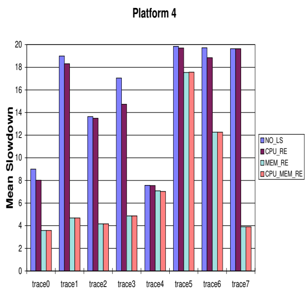


Figure 5: Mean slowdowns of all the 8 traces scheduled by different load sharing policies with the mean memory demand of 4 MBytes on platform 3 (left) and platform 4 (right).

We also adjusted the page fault rate for each trace in order to obtain reasonably balanced slowdown heights of 20 among all the traces on the 4 platforms. Table 3 lists all the page fault rates (σ's) for each trace on different platforms.

The NO_LS policy has the highest slowdown values for all the traces on all the platforms. The CPU_RE policy has the second highest slowdown values for all the traces on all the platforms. Policies MEM_RE and CPU_MEM_RE performed the best for all the traces on all the 4 platforms.

trace	0	1	2	3	4	5	6	7
platform 1	1.2	0.37	0.26	2.5	0.23	0.24	0.8	2.2
platform 2	1.2	0.37	0.26	2.5	0.23	0.24	0.8	2.2
platform 3	0.5	0.17	0.11	1.0	0.10	0.11	0.36	0.95
platform 4	0.5	0.15	0.10	0.94	0.10	0.24	0.35	0.95

Table 3: Page fault rates used for the 8 traces on the 4 platforms.

5.3 Effects of initial job pool assignment

job pools	1	2	3	4	5	6
job account	6337	28	4	1398	704	5338

Table 4: The number of jobs initially assigned to each node in the trace-driven simulation.

In our simulation, each node is initially set with a job pool where jobs are waiting for scheduling by the node. Table 4 gives the initial number of jobs in each job pool. In our simulations discussed in sections 5.1, 5.2, and the next section, job pool j is assigned to node j, for $j = 1, ..., 6$. The initial job pool assignment among the nodes in Table 4 may not match the heterogeneous computing capability and storage capacity of different nodes. Therefore, a less powerful node may be assigned more jobs, and a powerful node may get fewer jobs. The initial job pool assignments among the nodes can be rearranged in accordance with each node's capability and capacity before starting the execution of workloads. This is simply done by sorting the node set of a platform based on each node's CPU power and memory capacity, and sorting the job pool set based on the number of jobs in each job pool. Then we make the initial assignment between the node set and the job pool set in their sequential sorted orders. We call this assignment as the best initial job pool assignment. If we make the initial assignment between the node set and the job pool set in the opposite sorted order, we will get the worst initial job pool assignment by giving the largest job pool to the least powerful node and the smallest job pool to the most powerful node. Table 5 presents the mean slowdowns of "trace 0" by NO_LS on different heterogeneous platforms using both the worst and the best initial job pool assignments (the worst case and the best case). Using the worst initial job pool assignment, the slowdowns increase as the system heterogeneity increases. The slowdowns sharply increase to more than 230,000 on platform 4 from 1.71 on platform 1. In contrast, the slowdowns slightly decrease as the system heterogeneity increases using the best initial job pool assignment (from 1.71 to 1.35). Our experimental results indicate that an effective initial job pool assignment can well take advantage of the system heterogeneity. However, this may not be a realistic approach for a distributed load sharing system where migration decisions are made independently at each node. The initial job pool assignment can be done by a centralized scheduler.

Worst case	1 MBytes ($\sigma = 2.0$)	4 MBytes ($\sigma = 0.5$)
platform 1	1.71	1.71
platform 2	2.24	2.25
platform 3	7.82	9.11
platform 4	233,830	236,211
Best case	1 MBytes ($\sigma = 2.0$)	4 MBytes ($\sigma = 0.5$)
platform 1	1.71	1.71
platform 2	1.53	1.54
platform 3	1.33	1.33
platform 4	1.35	1.35

Table 5: Mean slowdowns of “trace 0” with NO_LS using the worst initial job pool assignment (worst case) and the best initial job pool assignment (best case). In the both case studies, memory demand of 1 MBytes with page fault rate of 2.0 and memory demand of 4 MBytes with page fault of 0.5 are used in the simulations.

6 Effects of Memory Demand Distributions

In our experiments, the memory demand of jobs in workloads is generated by a Pareto distribution. How is the performance of memory-based and CPU-Memory-based load sharing policies affected by other distributions of the memory demand? To address this question, we ran the simulations on the workloads with different memory demand distributions, and found that policies MEM_RE and CPU_MEM_RE are also effective on other memory demand distributions besides the Pareto distribution.

The other distributions we have used for comparisons are uniform distribution, exponential distribution, and erlang distribution. We ran the experiments for “trace 0” with these memory demand distributions on platform 1, the homogeneous configuration, and on platform 4, the most heterogeneous configuration.

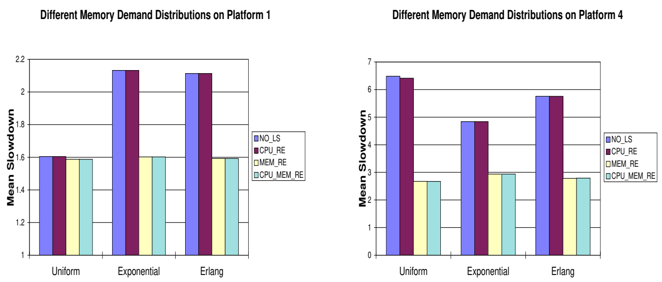


Figure 6: Mean slowdowns of “trace 0” with different memory demand distributions on platform 1 (left), and on platform 4 (right).

Figure 6 compares the mean slowdowns of “trace 0” scheduled by the three policies and NO_LS on platform 1 (left) and on platform 4 (right). In order to make the slowdowns of all the experiments on different platforms less than 10, on platform 1, the page fault rate is $\sigma = 4.14$ for the uniform memory demand distribution, is $\sigma = 1.21$ for the exponential distribution, and is $\sigma = 2.3$ for the erlang distribution. On platform 2, the page fault rate is $\sigma = 0.27$ for the uniform memory demand distribution, is $\sigma = 0.35$ for the exponential distribution, and is $\sigma = 0.30$ for the erlang distribution. The mean memory demand of the workloads on both platforms is 48MBytes.

With the exponential and erlang distributions, the mean slowdowns by CPU_MEM_RE and MEM_RE are 1.33 times lower than that by NO_LS and CPU_RE on platform 1, and more than 1.65 times lower on platform 4, respectively. Although policies CPU_MEM_RE and MEM_RE for the uniform memory demand distribution do not significantly overperform NO_LS and CPU_RE on platform 1, the performance improvement (2.4 times) by the policies is significant on platform 4.

7 Conclusion

We have experimentally examined and compared a CPU-based, a memory-based and a CPU-Memory-based load sharing policies on heterogeneous networks of workstations. Based on our experiments and analysis we have following observations and conclusions:

- The CPU and memory weights of workstations can be effectively used to characterize heterogeneity of a distributed system for designs of load sharing policies. For given total CPU power and total memory capacity, we can have different homogeneous and heterogeneous configurations with a roughly equivalent purchase cost. Under such a condition, the performance evaluation and comparisons are meaningful and useful.
- The CPU-based load sharing policy is not robust in a heterogeneous system, and performs poorly for memory-intensive workloads.
- The performance of the memory-based and CPU-Memory-based load sharing policies are quite independent of system heterogeneity changes for memory-intensive workloads. This is because the job migrations considering both memory and CPU resources offset the negative effects of the system heterogeneity. As the system heterogeneity increases to a certain degree, the remote executions and page faults also increase proportionally for the two policies, resulting a moderate degradation of the performance. However, our experiments also show that changes of the heterogeneity do not affect the functionality and nature of the two policies.
- An initial job pool assignment which uses information regarding system heterogeneity can allocate system resources effectively.
- We also show that the CPU-based, memory-based and CPU-Memory-based load sharing policies are independent on several different memory demand distributions.

The traces and simulation programs can be accessed at http://www.cs.wm.edu/hpcs/WWW/HTML/publications/abs00-2.html.

Acknowledgements: We wish to thank Stefan Kubricht for reading the paper and for his comments. The comments from the anonymous referees are insightful and constructive. The trace-driven simulator and the trace data available in the public domain from M. Harchol-Balter and A. B. Downey are helpful to our program development and testing of trace-driven simulations.

References

[1] A. Acharya and S. Setia, “Availability and utility of idle memory in workstation clusters”, *Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, May 1999, pp. 35-46.

[2] A. Barak and A. Braverman, “Memory ushering in a scalable computing cluster”, *Journal of Microprocessors and Microsystems*, Vol. 22, No. 3-4, August 1998, pp. 175-182.

[3] F. Berman, R. Wolski, S. Figueira, J. Schopf and F. Shao, “Application-level scheduling on distributed heterogeneous networks”, *Proceedings of Supercomputing’96*, November 1996.

[4] A. Cockcroft, *Sun Performance and Tuning*, Second Edition, Sun Microsystems Press, 1997.

[5] F. Douglis and J. Ousterhout, “Transparent process migration: design alternatives and the sprite implementation”, *Software — Practice and Experience*, Vol. 21, No. 8, 1991, pp. 757-785.

[6] X. Du and X. Zhang, “Coordinating parallel processes on networks of workstations”, *Journal of Parallel and Distributed Computing*, Vol. 46, No. 2, 1997, pp. 125-135.

[7] D. L. Eager, E. D. Lazowska, and J. Zahorjan, “The limited performance benefits of migrating active processes for load sharing”, *Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, May 1988, pp. 63-72.

[8] M. J. Feeley, et. al., “Implementing global memory management systems”, *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995, pp. 201-212.

[9] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke, “Multimethod communication for high-performance metacomputing applications”, *Proceedings of Supercomputing’96*, November 1996.

[10] M. Faerman, A. Su, R. Wolski, and F. Berman, “Adaptive performance prediction for distributed data-intensive applications”, *Proceedings of Supercomputing’99*, November 1999.

[11] G. Glass and P. Cao, “Adaptive page replacement based on memory reference behavior”, *Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, May 1997, pp. 115-126.

[12] M. Harchol-Balter and A. B. Downey, “Exploiting process lifetime distributions for dynamic load balancing”, *ACM Transactions on Computer Systems*, Vol. 15, No. 3, 1997, pp. 253-285.

[13] C.-C. Hui and S. T. Chanson, “Improved strategies for dynamic load sharing”, *IEEE Concurrency*, Vol. 7, No. 3, 1999, pp. 58-67.

[14] T. Kunz, “The influence of different workload descriptions on a heuristic load balancing scheme”, *IEEE Transactions on Software Engineering*, Vol. 17, No. 7, 1991, pp. 725-730.

[15] A. Silberschatz and P. B. Galvin, *Operating Systems Concepts*, 4th Edition, Addison-Wesley, 1994.

[16] G. M. Voelker, et. al., “Managing server load in global memory systems”, *Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, May 1997, pp. 127-138.

[17] X. Zhang, Y. Qu, and L. Xiao, “Improving distributed workload performance by sharing both CPU and memory resources”, *Proceedings of 20th International Conference on Distributed Computing Systems*, (ICDCS’2000), Taipei, Taiwan, April 10-13, 2000.

[18] X. Zhang and Y. Yan, “Modeling and characterizing parallel computing performance on heterogeneous NOW”, *Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing*, (SPDP’95), October, 1995, pp. 25-34.

[19] S. Zhou, “A trace-driven simulation study of load balancing”, *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, 1988, pp. 1327-1341.

[20] S. Zhou, J. Wang, X. Zheng, and P. Delisle, “Utopia: a load-sharing facility for large heterogeneous distributed computing systems”, *Software — Practice and Experience*, Vol. 23, No. 2, 1993, pp. 1305-1336.