Availability and Utility of Idle Memory in Workstation Clusters

Anurag Acharya
Dept. of Computer Science
University of California
Santa Barbara, CA 93106

Sanjeev Setia
Dept. of Computer Science
George Mason University
Fairfax, VA 22030

Technical Report TRCS98-26
Dept of Computer Science
University of California, Santa Barbara

October 1998

Abstract

In this paper, we examine the availability and utility of idle memory in workstation clusters. We attempt to answer the following questions. First, how much of the total memory in a workstation cluster can be expected to be idle? This provides an estimate of the opportunity for hosting guest data. Second, how much memory can be expected to be idle on individual workstations? This helps determine the recruitment policy – how much memory should be recruited on individual hosts? Third, what is the distribution of memory idle-times? This indicates how long guest data can be expected to survive; applications that access their data-sets frequently within the expected life-time of guest data are more likely to benefit from exploiting idle memory. Fourth, how much performance improvement can be achieved for off-the-shelf clusters without customizing the operating system and/or the processor firmware? Finally, how long and how frequently might a user have to wait to reclaim her machine if she volunteers to host guest pages on her machine? This helps answer the question of social acceptability. To answer the questions relating to the availability of idle memory, we have analyzed two-week long traces from two workstation pools with different sizes, locations, and patterns of use. To evaluate the expected benefits and costs, we have simulated five data-intensive applications (0.5 GB-5 GB) on these workstation pools.

1 Introduction

Idle workstations have traditionally been harvested for their cycles. Exploiting idle workstations for hosting guest computation has been a popular research area. Systems that utilize idle workstations for running sequential jobs have been in production use for many years [15]; there has been significant amount of research on exploiting idle workstations for hosting parallel computations (for e.g. [1, 4, 5]). With the current growth in the number and the size of data-intensive tasks, exploiting idle workstations for their *memory* is an attractive option, especially for programs with transient peaks in their memory requirements or for data-intensive programs that are run infrequently.

Although the price of memory has fallen dramatically in recent years, installing large amounts of memory on a single machine remains expensive for two reasons. First, memory is cheap only in commodity sizes (32/64/128 MB); there is a large premium for larger sizes. For example, a 512 MB PC100 SDRAM DIMM costs $3900 [17] whereas four 128 MB PC100 SDRAM DIMMs cost $600 [21]. Second, commodity-priced machines have a limit on the amount of memory; server-class machines that can have larger memories come at a substantial premium.

Several research projects have proposed using memory of idle workstations for hosting guest data [7, 8, 11, 12, 14, 20, 22, 27]. Iftode et al [14] propose extending the memory hierarchy of multicomputers by introducing a remote memory server layer; Felten and Zahorjan [12] examined the idea of using remote client memory instead of disk for virtual memory paging; Schilit and Duchamp [23] investigated the use of remote memory paging for diskless portable machines; Dahlin et al [8] and Sarkar et al [22] propose schemes to use idle memory to increase the effective file cache size; Feeley et al [11] describe a low-level global memory management system that uses idle memory to back up both file pages and virtual memory page.

These efforts have focused on developing efficient mechanisms for hosting guest data. In this paper, we examine the *opportunity* for hosting guest data in real workstation clusters. We attempt to answer the following questions. First, how much of the total memory in a workstation cluster can be expected to be idle? This provides an estimate of the opportunity for hosting guest data. Second, how much memory can be expected to be idle on individual workstations? This helps determine the recruitment policy – how much memory should be recruited on individual hosts? Third, what is the distribution of memory idle-times? This indicates how long guest data can be expected to survive; applications that access their data-sets frequently within the expected life-time of guest data are more likely to benefit from exploiting idle memory. Fourth, how much performance improvement can be achieved for off-the-shelf clusters without customizing the operating system and/or the processor firmware? While the performance improvement for off-the-shelf clusters is likely to be smaller than that for customized clusters, it would be available to a much larger user community. Finally, how long and how frequently might a user have to wait to reclaim her machine if she volunteers to host guest data on her machine? This helps answer the question of social acceptability – whether users will feel comfortable enough with the performance of the system to agree to participate.

To answer the questions relating to the availability of idle memory, we have analyzed two-week long traces from two workstation clusters with different distributions of memory and different patterns of use. To evaluate the expected benefits and costs, we have simulated the execution of three real data-intensive applications (out-of-core LU decomposition, decision support queries on a sales database, data-mining for buying patterns in retail transactions and searching a large set of files) and two synthetic applications with memory footprints between 0.5 GB and 5 GB.

We first examine the memory availability questions. Section 2 describes the methodology used to monitor the memory usage in a workstation cluster, and the metrics computed to estimate the opportunity for hosting guest memory. Next, we evaluate how much performance improvement can be achieved without customizing the operating system and/or the processor firmware? In Section 3, we describe the framework assumed by our simulation experiments, the applications used in the study, and the system and workload models used in our simulations. Section 4 presents our results. Finally, we examine the impact of hosting guest data on the primary users of the workstations.

Cluster	32MB	64MB	64-128MB	128-256MB	256-512MB	1024MB	mean	median
`clusterA`	–	6	15	5	2	1	179MB	128MB
`clusterB`	8	13	1	1	–	–	62MB	64MB

Table 1: Distribution of physical memory on the hosts in the traced clusters.

2 Memory availability

To determine the availability of idle memory for hosting guest data, we have analyzed detailed two-week long traces (Sep 7-21) from two workstation clusters. These traces were gathered over from 52 workstations in two different clusters. We used these traces to determine a detailed breakdown of memory usage and availability in individual workstations, in machines with equal amounts of memory and in both the clusters. We first describe the traces and how they were collected. We then present memory usage and availability metrics for these traces. Finally, we present the distribution of memory idle-times.

2.1 Traced clusters

The first cluster, `clusterA`, consists of 29 Sun workstations running Solaris 2.5.1/2.6 at the University of ___. These workstations are used by faculty and graduate students for personal purposes as well as for running large compute-intensive jobs. Most of the workstations in this cluster had 128 MB or more (Table 1 presents the memory distribution). The total memory in this cluster was 5.2 GB. We use this cluster as an example of clusters with large memories.

The second cluster, `clusterB`, consists of 23 Sun workstations running Solaris 2.6 at __ University. Most of these workstations are used by faculty and researchers for personal purposes; a couple of the larger machines are also used for running compute-intensive jobs. Most of the workstations in this cluster had 64 MB or less (Table 1 presents the memory distribution). The total memory in this cluster was 1.4 GB. We use this cluster as an example of clusters with small memories.

2.2 Trace collection

For each workstation, we monitored seven kinds of information: (1) user activity, (2) CPU load, (3) kernel memory usage, (4) list of active processes, (5) process memory usage, (6) list of open files, (7) contents of the file cache.

- **User activity:** we tracked user activity by checking the access times for the Solaris keyboard and mouse devices files.[1]
- **CPU load:** we tracked CPU load using `/usr/bin/w`. We use the average number of jobs over the last 5 minutes as the measure of load.
- **Kernel memory usage:** we tracked kernel memory usage using two commands: `/usr/sbin/prtconf` and `/usr/bin/netstat -k | grep pp_kernel`.[2] The first command is used to determine the

[1] The keyboard device file is /devices/pseudo/conskbd0:kbd and the mouse device file is /devices/pseudo/consms0:mouse. Solaris updates the access times of these files to reflect user activity.

[2] The `-k` option to `netstat` is not documented on its man page. We thank Adrian Cockcroft for describing it in his column in *SunWorld Online*.

total memory installed in the system. The second command is used for two purposes. First, it reports the amount of memory available for allocation (reported as `totalpages`). The difference between the total physical memory and memory available for allocation is used for the initial load of the kernel [6]. Second, it reads the `system_pages` kernel statistics structure to determine the number of pages used by the kernel from this available pool [6] (see Figure 1 for sample output from this command).

- **List of active processes:** we used `ps` to determine the list of *active* processes. Every process that is reported by `ps` to consume non-zero CPU time was marked as an *active* process. In addition, all periodic daemons and the X server are assumed to be always be active and are marked as such. Processes created for capturing information for these experiments were not considered active.

- **Physical memory used by active processes:** we determine the physical memory used by active processes using the `memps` and `pmem`[3] utilities provided by RMCmem, the experimental kernel module made available by Richard McDougal [16]. The `memps` utility provides information about the total physical memory used by a list of processes. For each process in the list, it provides a breakdown of the resident set size into shared and private regions. The `pmem` utility provides a detailed breakdown of the resident set of a process into individual segments (see Figure 2 for a snippet of `pmem` output). Based on the information provided by these processes, we were able to compute a fairly accurate estimate of the total memory used by all active processes.

- **List of open files:** we used the `lsof`[4] utility to obtain the list of open files for each active process.

- **Contents of the file cache:** we used the `memps -m` utility from the RMCmem package to determine the files being cached by the OS (see Figure 3 for a snippet of output from this utility). To limit the size of this trace, only the changes in the file cache content since the last snapshot were saved.

We coordinated the monitoring operations using a perl script that woke up every 90 seconds[5] and spawned the necessary processes. To limit the impact of tracing on the memory usage patterns, we filtered the larger traces before writing them out to trace files: (1) we filtered inactive processes from the output of `ps`; (2) we determined the resident set breakdown only for active processes; (3) we saved only incremental snapshots for the contents of the file cache. As a result, we were able to limit the footprint of the trace files to about 100 KB per hour for 128 MB machines.

```
pp_kernel 3957 pagesfree 841 pageslocked 4100 pagesio 0 pagestotal 39526
```

Figure 1: Sample output from `/usr/bin/netstat -k | grep pp_kernel`

[3] This utility has been integrated into Solaris 2.6 as `/usr/proc/bin/pmap`.
[4] Available at *ftp://vic.cc.purdue.edu/pub/tools/unix/lsof*.
[5] We found that intervals shorter than 90 seconds did not change the results significantly, while increasing both the overheads associated with tracing and the size of the trace files.

```
Address   Kbytes Resident Shared Private Permissions       Mapped File
00010000     760      680    656      24 read/exec         emacs
000DC000    1232     1192    344     848 read/write/exec   emacs
00210000     296      216      -     216 read/write/exec    [ heap ]
EF350000      16       16     16       - read/exec         libc_psr.so.1
EF360000      16       16     16       - read/exec         libmp.so.2
EF372000       8        8      8       - read/write/exec   libmp.so.2
```

Figure 2: Snippet of `pmem` output.

```
SunOS play 5.6 Generic sun4u    10/13/98

23:08:15
   Size  Filename
  2024k  /usr/local/sbin/cfengine
  1936k  /fs/net/solaris/bin/emacs
  1464k  /usr/dt/lib/libXm.so.3
   616k  /usr/lib/libc.so.1
   560k  /usr/bin/perl
   464k  /cachefs/cachedir/0000000000005901/.cfs_attrcache/0000000000069400
   464k  /usr/lib/libnsl.so.1
   448k  /usr/openwin/bin/Xsun
   440k  /usr/openwin/lib/libX11.so.4
```

Figure 3: Snippet of output from `memps -m`.

2.3 Memory usage and availability

We divided the memory on each workstation into six categories based on its usage pattern: (1) *static kernel memory* which is used for the initial kernel load; (2) *dynamic kernel memory* which is used for various kernel data structures and modules; (3) *lotsfree* which is the list of free pages maintained by the paging daemon;[6] (4) *file cache* which is used to cache disk files; (5) *process memory* which holds the resident sets of active processes; and (6) *available memory* which is the remaining memory. Of these, *static kernel memory* and *lotsfree* are fixed at boot time.[7] We computed *dynamic kernel memory* using the information provided by `/usr/bin/netstat`. We computed process memory using the resident set breakdown provided by `pmem` and `memps`. Note that, in Solaris, the virtual memory and the file cache are unified: pages corresponding to files are classified as process memory if they are `mmap`'ed into the address space of an active process; otherwise they are considered as part of the file cache.

Estimating the memory footprint of the file cache was a bit more complex. We considered the set of files that were opened by a process (reported by `lsof`) to be active at the instants where the process was active (cpu usage > 0). Note that this set of files also includes the executable file for the process and all the library files that are mapped into the process address space. The

[6]If the number of free pages drops to less than `lotsfree`, the paging daemon steals previously allocated pages.
[7]For Solaris 2.5.1/2.6, `lotsfree` = (TotalMemory - StaticKernelMemory)/64.

memory footprint of these active files was determined using the information provided by `memps -m`. However, since the file cache is used to cache files that will be used in the future, the challenge was to determine the memory footprint of *live* files, i.e., files which will be accessed again in the near future. To do this, we performed two kinds of lookahead while processing our traces. When a workstation was busy, we looked ahead to find the point when it next became idle; all files that were accessed during this "busy" period were assumed to be *live* during the whole "busy" period. When a workstation was idle, we looked ahead to the next "busy" period; all files accessed in the "idle" period as well as the following "busy" period were assumed to be *live* during the "idle" period. Note that the average length of a "busy" period was about 25 minutes. Thus, our computation of the memory used for the file-cache at a given instant includes not only the memory used for caching files that are active at that instant but also the memory used for caching files that will be active in the not-too-distant future.

A workstation was considered to be idle if there was no user activity (keyboard or mouse) *and* the load (as reported by `w`) was less than 0.3 for five minutes or more.[8] We would like to caution the reader that since our traces have a granularity of 90 seconds and since we do not directly track file open/close operations, we probably miss some bursts of file-cache activity, especially during busy periods.

Figure 4 presents the variation (over two weeks) in memory usage and availability for workstations of different size. These results have been averaged over all machines of the same size. We grouped machines by memory size to avoid computing averages of quantities with very large differences. We summarize this information in Table 2 which presents the average size of each of these divisions over the two-week period. In this table, `kernel memory` includes both *static kernel memory* and *dynamic kernel memory*. We make the following observations:

- The total memory used by the operating system is a substantial fraction of the total memory in use. It ranges from 10 MB on 32 MB machines to 49 MB on 256 MB machines and is relatively insensitive to time-of-day/day-of-week effects. In general, the average memory used by the kernel is significantly more (1.6-2.8 times) than the average memory used by user processes. Note that the number for the kernel memory includes the page daemon free list and the memory used for initial kernel load and does not include the file-cache. The number for process memory includes all active processes, the X server and all periodic daemons.

- The average amount of memory used by the file-cache to hold files that will be accessed in the *not-too-distant* future is small compared to the total memory on the machine. The fraction of the memory used to cache *live* files varies significantly with time and day but the mean+std_dev for this fraction is within 15% of the total memory for all memory sizes.

- On the average, a substantial fraction of the total memory of a machine is not in active use. This *fraction* grows as the total amount of memory on a machine grows – from about 12-14 MB for a 32 MB machine to about 180-192 MB on a 256 MB machine.

- The active process memory and file cache memory increases slowly with the physical memory on the host (see Table 2). We find that the resident set sizes of processes and that the sizes of cached files tend to be larger on machines with larger memories.

We would like to emphasize that these numbers are *averages* and as such they smooth over sharper variations in memory usage that occur on individual machines. To some extent, they appear

[8]This measure of workstation idleness has been used in several studies of workstation utilization.

Host type	kernel memory (KB)	file-cache memory (KB)	process memory (KB)	available memory (KB)
32MB hosts	10310 (1133)	2402 (2257)	3746 (2686)	16310 (3844)
64MB hosts	16347 (2081)	4093 (3776)	10017 (6982)	35079 (8030)
128MB hosts	25512 (3257)	8216 (10271)	12583 (12621)	84761 (17623)
256MB hosts	50109 (8625)	7384 (7821)	17606 (23335)	187045 (47535)

Table 2: Average amount of memory used for different purposes. The numbers in parentheses are the corresponding standard deviations.

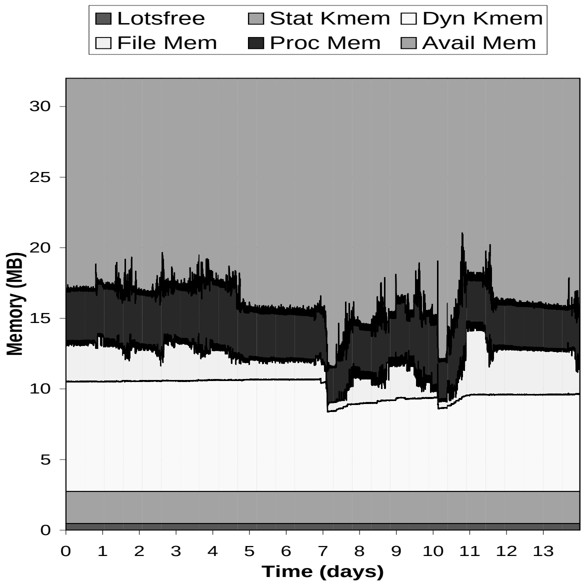
to run contrary to a widely held perception that "my workstation pages a lot". To understand memory availability from the point of view of users sitting in front of individual machines, we studied the variation in memory usage for individual workstations. Figure 5 presents the variation in available memory in individual workstations with different amounts of memory. We note that for each workstation, there are several points at which the amount of memory available is very low – these are points at which the workstation is likely to page and the user is likely to perceive the memory of her workstation to be running short. Table 3 quantifies this perception. It indicates the frequency with dips in memory availability occur. We note that half the hosts in this study (26) had at least one interval in which the memory availability dropped below 6 MB; one-fourth of the hosts had 20-30 such intervals over the period of two weeks.

The graphs in Figure 5 also show that even though such dips in memory availability occur frequently enough to be noticed, a substantial fraction of the memory is available on individual workstations most of the time. We conclude the perception of memory being short is based on infrequently occurring worst-case memory requirements and that a substantial fraction of memory on a workstation is available if we look at day-long/week-long intervals.

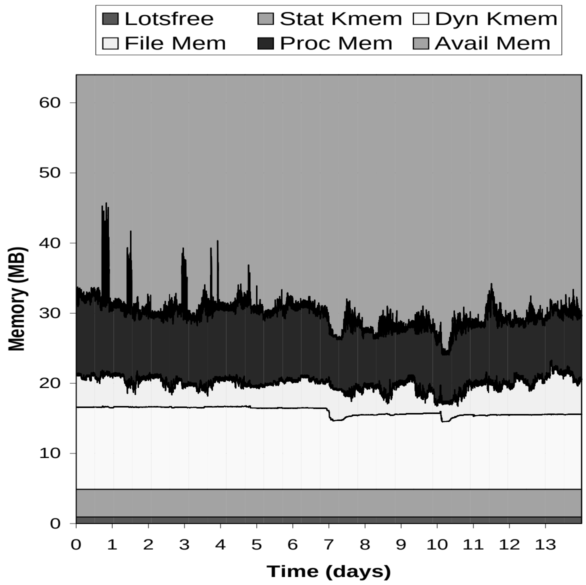
Number of intervals	0	≥ 1	≥ 6	≥ 11	≥ 21	≥ 31
8 MB	24	28	19	18	15	14
6 MB	26	26	16	16	14	12
4 MB	32	20	14	14	13	10

Table 3: Dips in memory availability. The box in the row labeled M MB and the column labeled $\geq k$ dips, contains the number of hosts for which the memory availability drops below M MB for k or more intervals over the two week period. For example, half the hosts (26) had at least one interval in which the memory availability dropped below 6 MB.

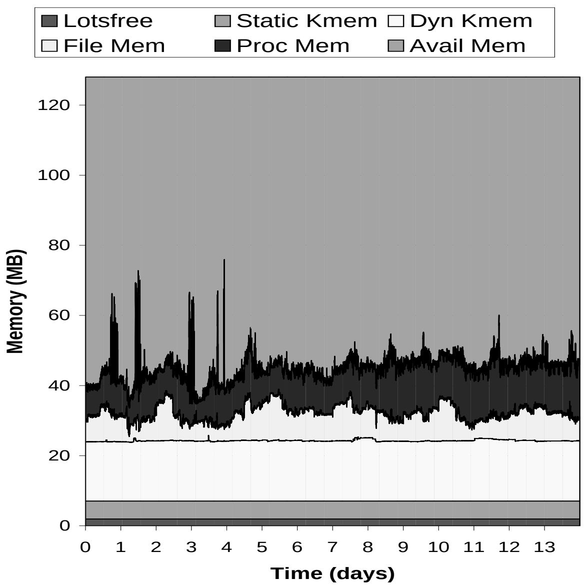
Figure 6 presents the variation in total memory available in both the clusters. It differentiates between the memory available on workstations that are in use and the memory available on workstations that are idle. We make two observations. First, for `clusterB`, most of the memory available is on idle workstations whereas for `clusterA`, substantial amounts of available memory is on workstations that are in use. Second, on the whole, a larger fraction of the total memory in `clusterA` is available than in `clusterB`. These differences are to be expected as the machines in `clusterB` are relatively small (average memory: 62 MB) whereas the machines in `clusterA` are relatively large (average memory: 179 MB). Moreover, many of the machines in `clusterA` are much larger (8 out of 29 machines have more than 220 MB) and can contribute a larger fraction of their



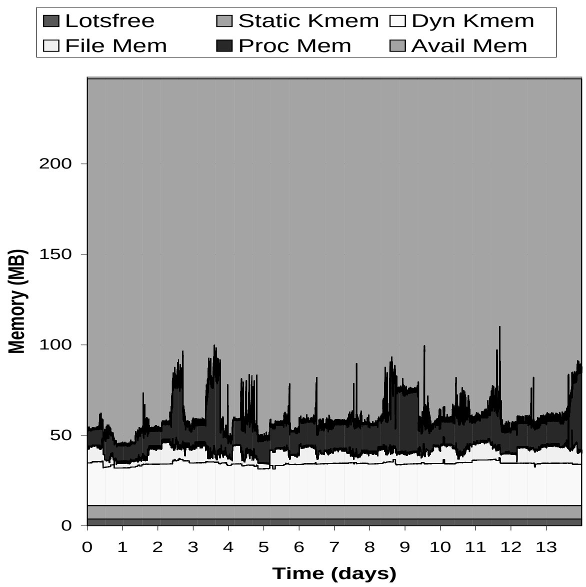
(a) 32 MB machines



(b) 64 MB machines

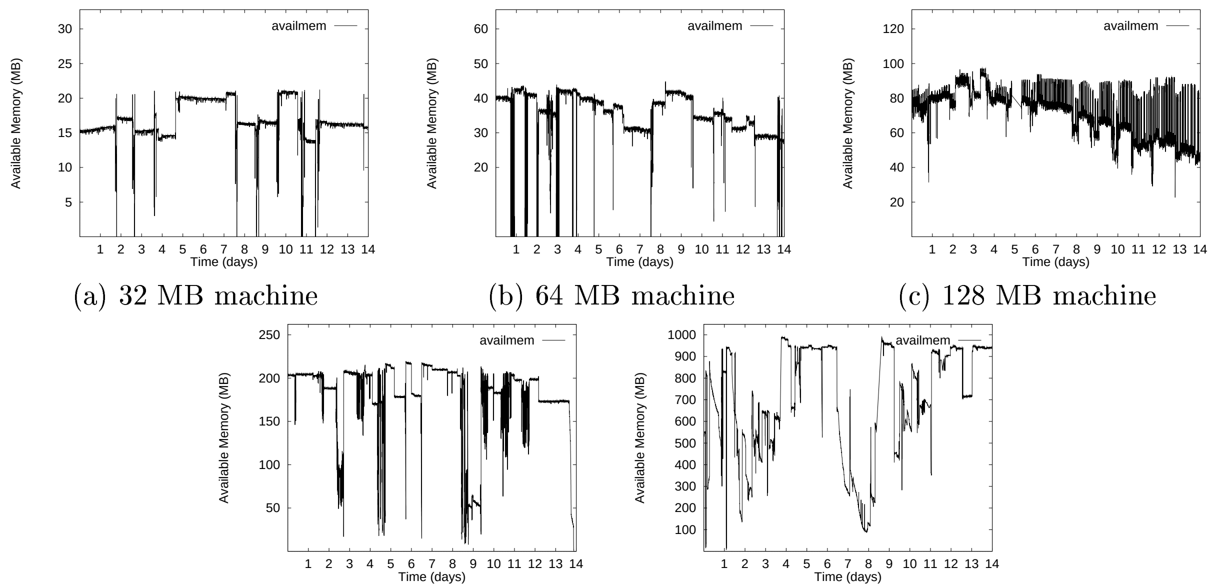


(c) 128 MB machines



(d) 256 MB machines

Figure 4: Variation in memory usage and availability. In each graph, the shaded portion of the figure at the top denotes the average amount of memory available for machines of that size. For these graphs, we consider all machines in both the clusters. We have not presented the corresponding graph(s) for machines with memories larger than 256 MB as there are only three such machines and their size differs too much (320MB-1024MB) to be averaged in a meaningful way.



(a) 32 MB machine (b) 64 MB machine (c) 128 MB machine

(d) 256 MB machine (e) 1 GB machine

Figure 5: Variation in available memory for individual workstations. Note that the range of the y-axis is different for each of the graphs. For each graph, the y-range corresponds to the total memory on the machine. These graphs show that while there are "dips' in memory availability, large fractions of a workstation's memory is available most of the time.

memory.

Figure 7 presents the breakdown of the available memory contributed by machines of different size for both clusters. We observe that in `clusterA`, about 53-60% of the available memory is contributed by the larger hosts (with $\geq$ 256 MB): when idle hosts are considered, this contribution consists of roughly equal shares from 256 MB hosts and larger hosts; when all hosts are considered, the contribution of the larger hosts dominates. This is because even when busy, these hosts often use only a fraction of their total memory. The figures for `clusterB` do not show this variation as most hosts in this cluster are relatively small and have relatively small amounts of memory available when they are busy.

2.4 How long is a memory region available for?

To understand how long guest data can be expected to survive, we determined the length of time memory regions of different size were available on each workstation. Applications that access their data-sets frequently within the expected life-time of guest data are more likely to benefit from exploiting idle memory. For all workstations with a given amount of memory, we computed the average length of periods for which memory regions of different size were available. To eliminate very short periods (which a practical memory harvesting system is unlikely to exploit), we eliminated all periods shorter than five minutes.[9]

Table 4 presents the average length of idle periods for memory regions. If you consider all hosts (not just the ones that are idle), memory regions as large as half the total memory on

[9]A similar limit is used by most workstation resource harvesting systems including Condor.

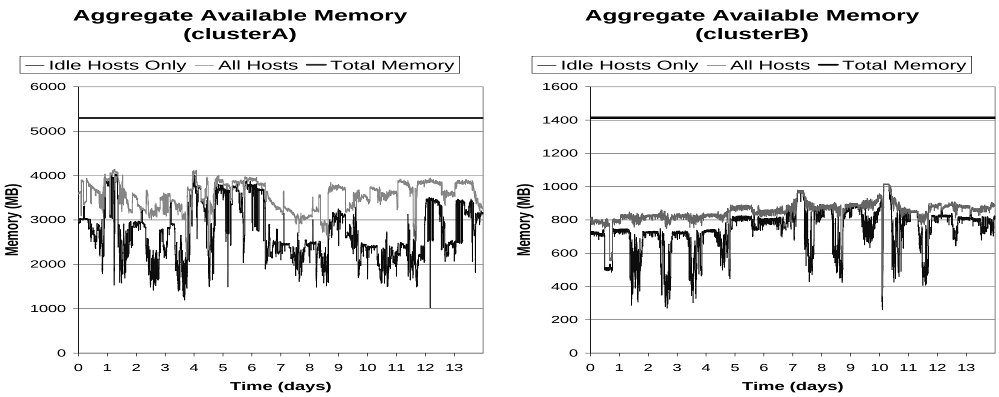


Figure 6: Variation of total memory available in the two clusters. The graphs marked "All Hosts" correspond to memory available on both idle and busy hosts; the graphs marked "Idle Hosts only" correspond to memory available on idle hosts. For `clusterA`, the average available memory for all hosts is 3549 MB and the average available memory for idle hosts is 2747 MB. The corresponding numbers for `clusterB` are 852 MB and 742 MB.

reasonably configured workstations (those with $\geq$ 64 MB physical memory) can be expected to be available for 39 minutes or more; and memory regions that are a quarter of the total memory on a machine can be expected to be available for 6 hours or more. If you consider only memory that is on idle workstations, the corresponding average idle periods are 12 minutes or more for half the workstation's memory and 30 minutes or more for a quarter of the workstation's memory. These results suggest that even moderately long-running applications might be able to derive benefit from exploiting idle memory.

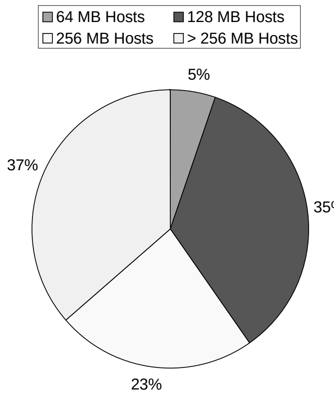
3 Experiments

To evaluate the expected benefits and costs of exploiting idle memory in off-the-shelf workstation clusters, we have simulated the execution of three real data-intensive applications (out-of-core LU decomposition, decision support queries on a sales database and data-mining for buying patterns in retail transactions) and two synthetic applications on the traced workstation clusters. The goal of our simulations was to assess both the benefit that seen by an application from the use of idle memory for caching, as well as the cost seen by the owner of a workstation that is used in this manner.

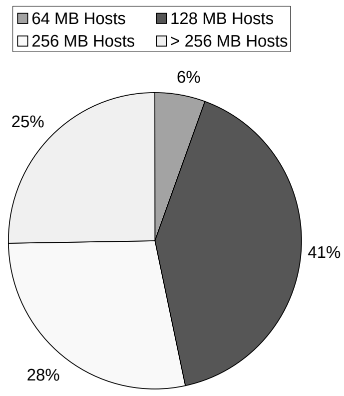
We first describe the infrastructure we assume for managing idle memory in a cluster. We then describe the applications that we used for our experiments, as well as the system and workload models. Finally, we describe the metrics used to assess the cost/benefit of exploiting idle memory.

3.1 Infrastructure for managing idle memory on off-the-shelf clusters

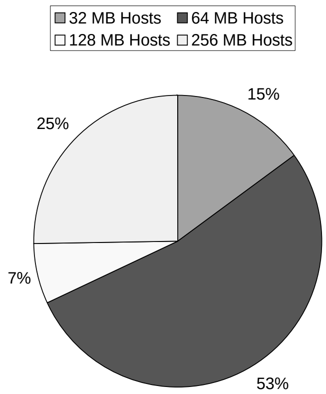
For these experiments, we assumed a simple portable runtime system that is suitable for managing idle memory in an off-the-shelf workstation cluster. The design of this system, known as *Dodo* [3], is based on that of Condor [15] which harvests idle processors. Each application is linked to a library that implements the functionality needed by the application in order to create, read, write, and delete remote memory regions. In contrast to global memory systems such as GMS [11] where



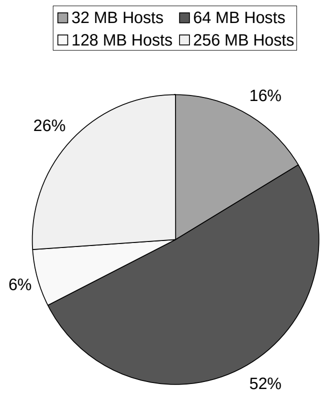
(a) all `clusterA` hosts



(b) idle `clusterA` hosts



(c) all `clusterB` hosts



(d) idle `clusterB` hosts

Figure 7: Breakdown of memory contribution from hosts of different sizes.

Host type	4MB	8MB	16MB	32MB	64MB	128MB	256MB	512MB
32MB hosts (all)	32hrs	6hrs	1hr	–	–	–	–	–
32MB hosts (idle)	24min	16min	5min	–	–	–	–	–
64MB hosts (all)	38hrs	25hrs	9hrs	39min	–	–	–	–
64MB hosts (idle)	58min	55min	39min	12min	–	–	–	–
128MB hosts (all)	16hrs	15hrs	14hrs	10hrs	90min	–	–	–
128MB hosts (idle)	35min	35min	34min	30min	15min	–	–	–
256MB hosts (all)	10days	8days	3days	50hrs	26hrs	22hrs	–	–
256MB hosts (idle)	48min	47min	43min	40min	31min	28min	–	–
All hosts (all)	29hrs	16hrs	6hrs	2hrs	2hrs	25hrs	5hrs	8hrs
All hosts (idle)	48min	44min	38min	23min	26min	58min	2hrs	90min

Table 4: Average length of idle periods for different memory regions and host groups. The increase in average length of idle periods for large memory regions ($\geq$ 128MB) in the last two rows is due to a small number of hosts with large amounts of memory (320-1024 MB).

applications use remote memory pages implicitly, *Dodo* requires applications to make explicit use of remote memory using an API similar to the *stdio* API provided by the C runtime library (the main routines being mopen(), mread(), mwrite(), and mclose()). We chose an explicit and synchronous interface for portability and simplicity. *Dodo* has been designed to run on Unix platforms; our implementation of *Dodo* currently runs on Linux and Solaris. Note that in *Dodo*, remote memory is used for read-only caching. Writes to remote memory are propagated to disk in parallel to being sent to the remote host.

In addition to the runtime library mentioned above, *Dodo* has three other components: (i) resource monitors, (ii) idle memory daemons, and (iii) a central memory manager and scheduler. Resource monitors execute on each participating workstation and notify the central manager when a workstation becomes idle. The central manager keeps track of the idle workstations in the cluster as well as the amount of memory available on each workstation. An idle memory daemon is started on a workstation when it is recruited and is terminated as soon as the workstation becomes busy. It stores remote memory regions in its address space and supplies them on request.

To allocate remote memory, the Dodo runtime library makes a request to the central manager. The central manager (randomly) selects a remote host from the list of hosts it knows have enough memory to satisfy the request. If the allocation is successful, a descriptor is returned to the library which includes the identity of the remote host and a memory-region identifier on that host. The library returns a handle to the application which it uses in all subsequent operations on that region. The allocation fails if sufficient memory is not available. When this happens, the library refrains from making allocation calls for a fixed time period, called the *refraction period*. The motivation behind this is to reduce the cost of allocation attempts in a period when the allocation attempts are not likely to succeed. When an attempt to access a memory region on a particular node fails (either because the node has crashed, or because it is no longer available for hosting remote memory, or because it has previously dropped the remote memory region that was requested), the library drops all the descriptors for memory regions stored on that node.

Memory recruitment policy: In our simulations, we assume that a workstation's memory is recruited only if the workstation has been idle for five minutes. We make this assumption for social acceptance reasons. Workstation owners are usually reluctant to have their resources utilized by guests while it is in use. As shown by a recent study [28], serving remote memory requests can potentially degrade the performance of local jobs.

In order to further limit the impact of hosting guest data on workstation owners, we restrict the maximum amount of memory that is recruited from a workstation. The idle memory daemon uses `netstat`, `ps`, `memps` and `pmem` to determine the memory in use. To this, it adds (i) the memory in the paging free list ($lotsfree$), and (ii) a "headroom" of 15% of the total memory to account for the files in the file cache that are not currently open but might be opened in near future. Recall from Section 2 that 15% of total memory is usually enough to hold the live files in the file-cache. The idle memory daemon recruits the remaining memory.

3.2 Applications

We used three real applications and two synthetic benchmarks in our study. For the real applications, we used traces captured by running these applications on realistic workloads [26]. These traces were obtained using AIX `trace` utility on eight processors of an IBM SP-2. They contain information about the compute time between successive I/O requests, the operation performed (read, write, open, seek, close), the file that is accessed, the size of the request, and the offset of the request within the file. The datasets vary from 0.5 GB to about 5 GB.

db2: the workload for `db2` consisted of five consecutive queries against a simulated database containing records for one year's operation of a large department store (one million customers, 100 thousand products, 10 thousand employees, 200 departments and 100 million transactions). These queries perform complex join, select, and aggregate operations on indexed and non-indexed relations. The data was stored in the DB2 database from IBM. The total database size was 5.2 GB (including the indices) and it was stored in 831 files.

dmine: this application tries to extract association rules from retail data [2, 19]. The dataset consists of 50 million transactions, with an average transaction size of 10 items and a maximal potentially frequent set size of 3 (see [2, 19] for details). The dataset size for this program was 4 GB and was partitioned into 8 files.

lu: this application computes the dense LU decomposition of an out-of-core matrix [13]. The dataset consisted of an 8192×8192 double precision matrix (total size 536 MB) with a slab size of 64 columns. The data is stored in 8 files.

We chose these three applications as representatives of three classes of applications. We chose `lu` to represent the applications whose dataset size fits in the amount of idle memory available in the cluster. We chose `dmine` to represent the class of applications whose dataset does not entirely fit into the available idle memory yet a significant portion does. We chose `db2` to represent the applications whose dataset was much larger than the available idle memory.

For these applications, we used a "first in" policy for deciding which data-objects are stored in remote memory. That is, the application attempts to store data-objects in remote memory as long as there is available space. Once a data-object is allocated space in remote memory, it will not be replaced from that memory. (However, it will be dropped from remote memory if its host becomes busy). This results in a situation in which the objects that are accessed during the earlier stages of an application's execution having a greater chance of being stored in remote memory. The motivation for using a "first in" policy comes from the fact that the applications used in our simulations exhibit scan and triangle-scan I/O access patterns [26]. For such applications, a "first in" policy is a good caching policy.

In addition, we used two data-intensive synthetic benchmarks to evaluate the impact of access-patterns not exhibited by the real applications. For both these benchmarks, we used a 2 GB dataset. This dataset does not fit in the average memory available in `clusterB` but fits within the average memory available in `clusterA`. For both benchmarks, we assumed a constant compute time of 10ms between successive I/O requests. Both benchmarks make 262144 (256K) requests.

hotcold: this benchmark divides its dataset into a 20% "hot" region and a 80% "cold" region; 80% of the references are to the "hot" region. Within each region, the requests are random. It makes 8 KB read requests.

random: this benchmark makes random 8 KB read requests from the entire dataset.

3.3 System and Workload Model

In our simulations, we assumed that a single application was using the idle memory on the workstation cluster. We assumed that this application was being executed on a dedicated workstation with four 250 MHz processors and 256 Mbytes of main memory. Such configurations are becoming increasingly common in desktop machines. The simulator also models the communication between the Dodo runtime library and the processes used to harvest and manage idle memory (idle memory daemons on individual machines and the central manager process). For the core set of experiments, we assumed the workstations in the cluster were connected by a gigabit ethernet with an end-to-end bandwidth of 70 MB/s, and an end-to-end latency of 7.4μs. We based these parameters on a commercial evaluation report of a gigabit ethernet switch [25] and on a description of an efficient communication library for PC clusters [9]. To evaluate the impact of network bandwidth on the performance of remote caching, we performed additional experiments assuming a 100Mbps switched ethernet interconnect with 10 MB/s end-to-end bandwidth and 30 μs end-to-end latency (based on [18]).

We assume that the workstation has a 133 MB/s PCI bus with one Ultra-SCSI (40 MB/s) string and two Ultra-SCSI disks. The disk parameters used in our simulation were based on published parameters of the Seagate Cheetah 9 disk [24]. We assume an average seek time of 5.8ms, a maximum seek time of 15.7ms, a rotational speed of 10000 RPM, and an average media transfer rate of 18 MB/s. We model seek times using a truncated exponential distribution (min 0ms, mean 5.8ms, max 15.7ms).[10] We would like to point out that this I/O configuration has an aggregate disk bandwidth of 36 MB/s and is fairly aggressive. In comparison with the network interconnect, it can provide about half the bandwidth of the gigabit ethernet and over three times the bandwidth of 100 Mbps ethernet.

To account for the increase in processor speed (IBM SP-2 processors were 66MHz), we scaled down the computation time between I/O requests in the traces by a factor of four. To emulate the execution of an application on an eight processor configuration on smaller configurations, the operations on a processor are simulated until it reaches a synchronization point. At that point, the simulator switches to another processor.

Three sets of traces were used to drive our simulation: (i) workstation availability traces for each workstation in the cluster under consideration (ii) memory availability traces for each workstation in the cluster under consideration (iii) I/O traces for the applications. The workstation availability and memory usage traces for each workstation were obtained as described in Section 2.

For `lu`, `hotcold` and `random`, all remote memory regions created by an application are deleted at its completion. Thus any performance benefit they obtain from remote memory is due to accessing the same object multiple times during a single run. This corresponds to the operation of out-of-core applications that use remote memory to hold temporary data. For `db2` and `dmine`, remote memory regions are not deleted at the end of an application's execution. Instead, they are retained in remote memory unless dropped as a result of the node becoming busy. This corresponds to the operation of applications that access persistent data.

3.4 Metrics

Benefit: as the benefit metric, we used the speedup in execution time.

Cost: For a workstation owner, the primary cost of allowing her workstation's memory to be harvested is the delay experienced when she reclaims her workstation. This delay is a result of the workstation's memory contents being paged out to accommodate guest data while the workstation was idle. Measuring this delay directly is difficult – it poses some of the same challenges that arise in benchmarking interactive applications [10]. However, this delay will depend on the size of the memory context that will need to be re-established when the owner reclaims her workstations. We refer to this metric as the memory context size (MCS) and use it as a measure of the cost of exploiting idle memory. Note that the actual delay experienced by the user usually is much

[10] We found that our results were relatively insensitive to the seek time distribution.

smaller than time take to page in the memory context size, since the memory context size is based on the *entire* memory footprint – not just the working set size – of *all* active processes (including non-interactive processes).

As discussed in Section 2, the memory traces used to drive our simulation contain information about the memory footprint of all the active processes on a particular workstation at a given instant. Assume that the total memory footprint of the processes that become active in first 200 seconds of a busy interval is denoted by A, the memory used by the kernel is K, and that the amount of memory occupied by guest data at the time the workstation becomes busy is G.

In the common case, processes that were active just before the workstation went idle will also be the processes that become active when the workstation becomes busy (e.g. the `xterm` and/or the `netscape` processes the user was interacting with). Given the LRU-based page replacement policies used by modern operating systems, memory allocation by the idle memory daemon will result in a non-zero memory context size only if the difference between the physical memory on the workstation (say M) and the total memory demand $(G + A + K)$ falls below $lotsfree$ pages (the minimum amount of free memory maintained by the operating system). This give us the following estimate for the size of the user memory context (MCS) that will need to be reestablished:

$$MCS = \begin{cases} 0, & \text{when } A + G + K <= M - lotsfree \\ A + G + K - (M - lotsfree), & \text{when } A + G + K > M - lotsfree \end{cases} \tag{1}$$

Note, however, that without monitoring the memory management system, it is impossible to determine *exactly* which pages are replaced during an idle period. It is possible, however rather unlikely, that the entire memory footprint of the processes that become active when a workstation is reclaimed is paged out. To provide a (rather loose) upper bound on the memory context size, we compute the portion of A that is cached in memory during the idle period.

4 Results

We ran our experiments for one week of simulated time (Sept 14-20, 1998). This allowed us to understand the effect of time-of-day and day-of-the-week variations in workstation usage. All the results described below assume a memory refraction period of 1 minute.

Performance Benefits: Table 5 presents the speedups achieved by each of the applications on the two clusters. For comparison, it includes an `ideal` speedup for each application. We computed the `ideal` speedup using a configuration with infinite remote memory that is never reclaimed and a network with infinite bandwidth and zero latency.

The speedup obtained on a cluster depends on both the amount of memory available as well as the characteristics of the application. For compute-bound applications like `lu`, the speedup is limited (`ideal` speedup for `lu` is 1.27). Both `clusterA` and `clusterB` achieve a speedup of 1.19 for `lu`. For this application, the additional memory available in `clusterA` provides no advantage since `lu`'s footprint (536 MB) usually fits in the available memory on both clusters.

While `hotcold`'s memory footprint (2 GB) is larger than the available memory on `clusterB`, the speedup seen on the two clusters is comparable. This is due to the strong locality in its references: 80% of the references are to 20% of its memory footprint (400 MB). The speedups for `hotcold` are larger than those for `lu` as it has less computation per byte of data read.

The other three benchmarks are more I/O bound than `lu` and `hotcold`; as a result, the `ideal` speedups are larger. However, since all three have memory footprints larger than the average

Application	`ideal`	`clusterA`	`clusterB`
`dmine`	3.29	2.08	1.20
`db2`	2.45	1.28	1.08
`lu`	1.27	1.19	1.19
`hotcold`	1.57	1.39	1.32
`random`	2.01	1.72	1.24

Table 5: Speedup for different applications. This speedup is computed using the performance of the application running on the four-processor workstation with no remote memory as the baseline. The first column, `ideal` performance, is an upper bound on the performance improvement that can be achieved by using remote memory.

available memory on `clusterB`, they achieve relatively small speedups on this cluster (ranging from 1.08 for `db2` to 1.24 for `random`).

For `clusterA`, the entire data set of `random` and a large fraction of the data set of `dmine` often fit in the available memory; the data set of `db2`, on the other hand, is very large and never fits in the available memory. Accordingly, the first two achieve significant speedups (2.08 for `dmine` and 1.72 for `random`) while the speedup for `db2` is modest (1.28).

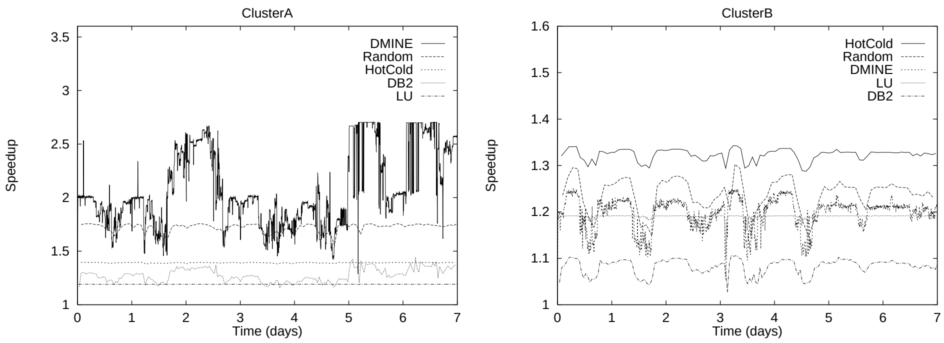


Figure 8: Variation of speedup for each benchmark on the two clusters over one week.

Figure 8 plots the temporal variation of the speedup obtained for each benchmark on the two clusters. The plots illustrate obvious time-of-day/day-of-week effects. The temporal variation in speedup for a benchmark depends on its sensitivity to the amount of available memory. As pointed out earlier, both clusters usually have enough available memory for `lu`. Thus, there is no variation over time in the speedup achieved by `lu` on both clusters. A similar trend can be observed, albeit to a smaller extent, for `hotcold` and `random` on `clusterA`. Since a significant portion of `dmine` can fit in `clusterA`'s available memory, its performance shows significant temporal variations. On the other hand, `db2` has much smaller variations as a much smaller fraction of its memory footprint fits in `clusterA`'s available memory.

Impact on Workstation Owners: Table 6 presents the average number of workstation reclamations per day per host and the average and median memory context size observed in our experiments. Our results show that in most cases a user will not experience any delays when she reclaims her workstation. The median of the memory context size is 0 for all five benchmarks on both clusters. Table 6 also shows that the average number of reclamations per day per host is 3.5 for `clusterA`

Application	clusterA		clusterB	
	reclaims/day	avg memory context	reclaims/day	avg memory context
	per host	(KB)	per host	(KB)
dmine	3.5	959 (0)	4.5	37.2 (0)
db2	3.5	929 (0)	4.5	38.4 (0)
lu	3.4	0.58 (0)	4.4	1.84 (0)
hotcold	3.3	0 (0)	4.3	17.2 (0)
random	3.5	104.1 (0)	4.5	30.6 (0)

Table 6: Impact of exploiting idle memory on workstation owners. The number in parentheses is the median. The average upper bound on the memory context size is 4 MB for clusterA and 3 MB for clusterB. As mentioned in section 3.4, this is a very loose bound.

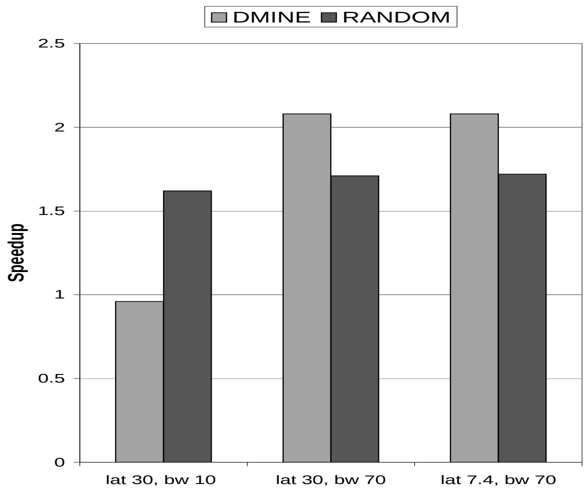


Figure 9: Impact of variation in network interconnect.

and 4.5 for clusterB. The difference in the number of reclamations reflects the fact that some of the machines in clusterA are used for compute-intensive jobs and sometimes idle for long periods. Note that the memory recruitment policy used in our simulations selects hosts randomly from among the idle hosts in the cluster; if necessary, a policy such as that proposed by Arpaci et al [4] can be used to limit the number of reclamations per day for a particular host.

Impact of variation in network interconnect: Our next set of results examines the impact of changing the network interconnect on the speedup obtained for individual applications. We considered the impact of increasing the network latency from 7.4 μs to 30 μs, and the impact of reducing the network bandwidth from 70 MB/sec to 10 MB/sec (e.g., switched Fast Ethernet) on the performance of the dmine and random benchmarks. Note that almost all requests from dmine are for 128 KB while all requests from random are for 8 KB.

We observe that increasing the network latency to 30 μs while keeping the network bandwidth fixed at 70 MB/s had a negligible impact on the speedup obtained for both dmine and random. On the other hand, reducing the network bandwidth to 10 MB/s converts the speedup for dmine into a *slowdown*. Recall that our experimental configuration assumes two disks each with a bandwidth of

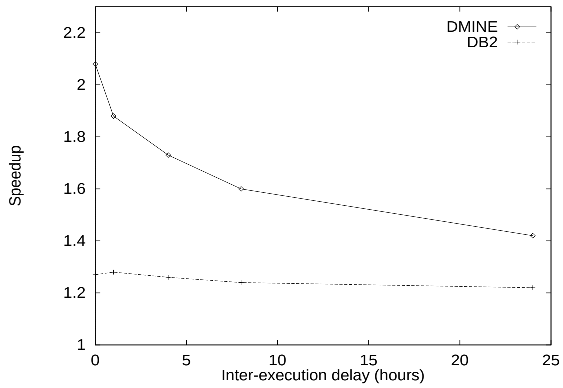


Figure 10: Impact of variation in inter-execution delay.

18 MB/sec. Since `dmine` makes large I/O requests, its performance depends largely on the network bandwidth, and a performance improvement is possible only if the network bandwidth is larger than the aggregate I/O bandwidth. Note, however, that `random` achieves significant speedup even for a 10 MB/s network. Due to the size and the random nature of its requests, the time for `random`'s disk requests is determined mostly by seek and rotational latencies.

Impact of variation in inter-execution delay: Our core set of experiments assumed repeated executions of the application with no delay between consecutive executions. To examine the impact of inter-execution delay on the performance of applications that access persistent data, we conducted additional experiments that varied the inter-execution delay. Figure 10 plots the speedup obtained for `dmine` and `db2` on `clusterA` as a function of the inter-execution delay. For `dmine`, increasing the inter-execution delay to *one day* reduces the speedup by about 40%, whereas for `db2`, it has a much smaller impact. The difference between the two benchmarks can be explained by the fact that the performance of `dmine` is much more sensitive to the amount of available memory than `db2` (see Figure 8).

Note that an application with a large memory footprint such as `dmine` can expect to get some performance benefit from using remote memory even if it is executed a day later. This indicates that there are some nodes that are idle for long periods. Memory regions stored by these nodes do not get dropped resulting in the performance benefits observed in our experiments.

5 Conclusions

There are four main conclusions of our study. First, on the average, a substantial fraction of the total memory of desktop machines is not in active use. This fraction grows as the amount of memory on a machine grows – from about 12-14 MB for a 32 MB machine to about 180-192 MB on a 256 MB machine. Second, for workstations with $\geq$ 64 MB of memory, memory regions that are equal to half the workstation's memory can be expected to be available for about 12 minute periods; the corresponding number for a quarter of the workstation's memory is 30 minutes. Third, there is significant benefit to harvesting idle memory on workstation clusters for data-intensive applications. Utilizing idle memory as an intermediate cache can result in significant speedups (up to 2.1) for applications with large memory footprints. Fourth, using a memory recruitment policy that targets only idle hosts and that does not harvest more memory than is idle on the host ensures that users experience virtually no delays when reclaiming their workstations.

Other conclusions of our study are:

- The total memory used by the kernel is a substantial fraction of the total memory in use. It ranges from 10 MB on 32 MB machines to 49 MB on 256 MB machines and is relatively insensitive to time-of-day/day-of-week effects.
- On average, a large fraction of the total memory installed on a cluster is available: 60-68% if we consider all hosts, and about 53% if we consider only idle hosts.
- Even though large fractions of a workstation's memory are available most of the time, there are many "dips" in memory availability which are likely to lead to a perception of memory being short.

Acknowledgments

We would like to thank Adrian Cockcroft for his *Performance Q&A* columns in *SunWorld Online*. They helped us better understand the different ways in which memory is used in Solaris. We would like to thank Adrian Cockcroft and Richard McDougal of Sun Microsystems for developing the RMCmem kernel module and making it publicly available. We would like to thank several research groups and faculty members at the University of California, Santa Barbara and the George Mason University for allowing us to trace the memory usage on their machines. We would also like to thank the systems staff in our departments for their help in installing RMCmem and lsof.

We would like

References

[1] A. Acharya, G. Edjlali, and J. Saltz. The utility of exploiting idle workstations for parallel computation. In *Proceedings of 1997 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, 1997.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of 20th Int'l Conf. on Very Large Databases (VLDB)*, Santiago, Chile, Sept. 1994.

[3] Anonymous. ___. Technical Report TRCS98-02, Dept of Computer Science, University of ___, 1998.

[4] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, and D. Patterson. The Interaction of Parallel and Sequential Workloads on a Network of Workstations. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 267–78, May 1995.

[5] A. Chowdhury, L. Nicklas, S. Setia, and E. White. Supporting dynamic space-sharing on clusters of non-dedicated workstations. In *Proceedings of the 17th International Conference on Distributed Computing*, 1997.

[6] A. Cockcroft. How Much RAM is Enough? Sun World Online[11], May 1996.

[7] D. Comer and J. Griffioen. A new design for distributed systems: The remote memory model. In *Proceedings of the 1990 USENIX Summer Conference*, 1990.

[11] http://www.sunworld.com/sunworldonline/swol-05-1996/swol-05-perf.html

[8] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative Caching: Using Remote Memory to Improve File System Performance. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 267–80, Nov 1994.

[9] S. Donaldson, J. Hill, and D. Skillicorn. BSP Clusters: High Performance, Reliable and Very Low Cost. Technical Report PRG-TR-5-98, Oxford University Computing Laboratory, 1998.

[10] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer. Using latency to evaluate interactive system performance. In *Proceedings of the 2nd USENIX Symposium on Operating System Design and Implementation*, pages 185–200, 1996.

[11] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, and C. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 201–12, Dec 1995.

[12] E. Felten and J. Zahorjan. Issues in implementation of a remote memory paging system. Technical Report 91-03-09, Department of Computer Science, University of Washington, 1991.

[13] B. Hendrickson and D. Womble. The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM J. Sci. Comput.*, 15(5), Sept. 1994.

[14] L. Iftode, K. Li, and K. Petersen. Memory Servers for Multicomputers. In *COMPCON Spring'93 Digest of Papers*, pages 538–47, Feb 1993.

[15] M. Litzkow and M. Livny. Experiences with the Condor Distributed Batch System. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, pages 97–101, Oct 1990.

[16] R. McDougal. The RMCmem kernel module. ftp://playground.sun.com/pub/memtool, June 1998.

[17] 512 MB PC100 SDRAM with ECC[12]. Micro X-press Inc. 5406 west 78th street Indianapolis, IN 46268, Oct 1998.

[18] Mier Communications. Evaluation of 10/100 BaseT Switches. *http://www.mier.com/reports/-cisco/cisco2916mxl.pdf*, April 1998.

[19] A. Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, College Park, August 1995.

[20] T. Narten and R. Yavatkar. Remote Memory as a Resource in Distributed Systems. In *Proceedings of the 3rd Workshop on Workstation Operating Systems*, pages 132–6, April 1992.

[21] 128 MB PC100 SDRAM[13]. Advanced PCBoost, PO Box 80811, Rancho Santa Margarita, CA 92688, Oct 1998.

[22] P. Sarkar and J. Hartman. Efficient cooperative caching using hints. In *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation*, 1996.

[23] B. Schilit and D. Duchamp. Adaptive remote paging for mobile computers. Technical Report CUCS-004-91, Department of Computer Science, Columbia University, 1991.

[12] *http://www.microx-press.com/online/* following link from *http://www.pricewatch.com*
[13] *http://www.pcboost.com* following link from *http://www.pricewatch.com*

[24] Seagate Technology Inc. *The Cheetah 9LP Family: ST39102 Product Manual*, July 1998. Publication number 83329240 Rev B.

[25] The Tolly Group. 3Com Corporation Superstack II 9300 Gigabit Ethernet Performance. Available off *http://www.tolly.com*, Jan 1998.

[26] M. Uysal, A. Acharya, and J. Saltz. Requirements of I/O systems for parallel machines: An application-driven study. Technical Report CS-TR-3802, Department of Computer Science, University of Maryland, 1997.

[27] G. Voelker, E. Anderson, T. Kimbrel, M. Feeley, J. Chase, A. Karlin, and H. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 33–43, 1998.

[28] G. Voelker, H. Jamrozik, M. Vernon, H. Levy, and E. Lazowska. Managing Server Load in Global Memory Systems. In *Proceedings of the 1997 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 127–138, June 1997.