Linger Longer: Fine-Grain Cycle Stealing for Networks of Workstations †

Kyung Dong Ryu Jeffrey K. Hollingsworth

Computer Science Department
University of Maryland
College Park, MD 20742
{kdryu, hollings}@cs.umd.edu

Abstract

Studies have shown that a significant fraction of the time, workstations are idle. In this paper we present a new scheduling policy called Linger-Longer that exploits the fine-grained availability of workstations to run sequential and parallel jobs. We present a two-level workload characterization study and use it to simulate a cluster of workstations running our new policy. We compare two variations of our policy to two previous policies: Immediate-Eviction and Pause-and-Migrate. Our study shows that the Linger-Longer policy can improve the throughput of foreign jobs on cluster by 60% with only a 0.5% slowdown of foreground jobs. For parallel computing, we showed that the Linger-Longer policy outperforms reconfiguration strategies when the processor utilization by the local process is 20% or less in both synthetic bulk synchronous and real data-parallel applications.

1. Introduction

Studies have shown that up to three-quarters of the time workstations are idle[11]. Systems such as Condor[9], LSF[19], and NOW[2] have been created to use these available resources. Such systems define a "social contract" that permits foreign jobs to run only when a workstation's owner is not using the machine. To enforce this contract, foreign jobs are stopped and migrated as soon as the owner resumes use of their computer. We propose a policy, called *Linger-Longer*, that refines the social contract to permit fine-grained cycle stealing. By permitting foreign jobs to linger on a machine at low priority even when foreground tasks are active, we can improve the throughput of background jobs in shared clusters by 60% while holding the slowdown of foreground jobs to only 0.5%.

The motivation for the Linger-Longer approach is simple: even when users are "actively" using workstations, the processor is idle for a substantial fraction of the time. In addition, a significant amount of memory is usually available. To minimize the effect on the owner's workload, current techniques do not use these fine-grain idle cycles.[1] Linger-Longer exploits these fine-grained idle periods to run foreign jobs with very low priority (so low that foreground jobs are allowed to starve the background task). Our approach enables the system to utilize most idle cycles while limiting the slowdown of the owner's workload to an acceptable level. To improve job response time, Linger-Longer will not let the foreign jobs linger forever on a busy machine. We employ a cost model to predict when the benefit of running on a free node outweighs the overhead of a migration.

The primary beneficiaries of the Linger-Longer scheduling policy are large compute-bound sequential jobs. Since most of these jobs are batch (no user interaction during execution), and consist of a family of related jobs that are submitted as a unit and must all be completed prior to the results being used (e.g., a collection of simulation runs with different parameters), job throughput rather than response time is the primary performance metric. We will concentrate on throughput as the metric we try to optimize.

A key question about Linger-Longer is whether a scheduling policy that can delay users' local jobs will be accepted. For several reasons, we think this problem can be overcome. First, as shown in Section 4.1, the delay that users will experience with our approach is very low. Second, existing systems that exploit free workstations also have an indirect impact on users due to the time required to re-load virtual memory pages and caches after a foreign job has been evicted.

In the rest of this paper, we present an overview of the Linger-Longer policy and evaluate its performance via simulation. Section 2 describes the Linger-Longer policy and explains its prediction model for migration. Section 3 characterizes the utilization of workstations, evaluates the potential for lingering, and presents a study of the available CPU time and physical memory on user worksta-

†This work was supported in part by NSF awards ASC-9703212 & ASC-9711364, and DOE Grant DE-FG02-93ER25176.

[1] Part of the motivation for this policy is to promote user acceptance of foreign jobs running on their system. However, after 10 years of experience with environments such as Condor, user acceptance seems to have been reached.

tions. Section 4 evaluates Linger-Longer scheduling impact and measures cluster-level performance by simulating a medium scale cluster of 64 nodes with sequential jobs. Running parallel jobs using Linger-Longer is also investigated in Section 5. Finally, Sections 6 and 7 present related work and conclusions respectively.

2. Fine-Grain Cycle Stealing

We use the term cycle stealing to mean running jobs that don't belong to the workstation's owner. The idle cycles of machines can be defined at different levels. Traditionally, studies have investigated using machines only when they are not in use by the owner. Thus, the machine state can be divided into two states: idle and non-idle. In addition to processor utilization, user interaction such as keyboard and mouse activity has been used to detect if the owner is actively using their machine. Acha[1] showed that for their definition of idleness, machines are in a non-idle state for 50% of the time. However, even while the machine is in use by the owner, substantial resources are available to run other jobs.

We introduce a new technique to make more idle time available. In terms of CPU utilization there are long idle intervals when the processor is waiting for user input, I/O, or the network. These intervals between run bursts by owners' jobs can be made available to others' jobs. We term running foreign jobs, while the user processes are active, lingering. Since the owner has priority over foreign jobs using their personal machine, use of these idle intervals should not affect the performance of the owner's jobs.

Delay of local jobs should be avoided. If not, users will not permit their workstations to participate in the pool. Priority scheduling is a simple way to enforce this policy. Current operating systems schedule processes based on their priority, and use a complex dynamic priority allocation algorithm for efficiency and fairness. To implement lingering, we need a somewhat stronger definition of priority for local and foreign job classes. Foreground processes have the highest priority and can starve background processes. In addition, when a background process is running, an interrupt that results in a foreground process becoming runnable, causes the foreground process to be scheduled onto the processor even if the background job's scheduling quanta has not expired.

Two strategies have been used in the past to migrate foreign jobs: Immediate-Eviction and Pause-and-Migrate. In *Immediate-Eviction*, the foreign job is migrated as soon as the machine becomes non-idle. Because this can cause unnecessary, expensive migrations for short non-idle intervals, an alternative policy, called *Pause-and-Migrate,* that suspends the foreign processes for a fixed time prior to migration is often used. The fixed suspend time should not be long because the foreign job makes no progress in the suspend state. With Linger-Longer scheduling, foreign jobs can run even while the machine is in use by the owner; therefore migration becomes an optional move to a machine with lower utilization rather than a necessity to avoid interference with the owner's jobs. Although migration can increase the foreign job's available resources, there is a cost to move the process's state. Also, the advantage of running on the idle machine depends on the difference in available processor time between the idle machines and current non-idle one. To maximize processor time available to a foreign job, we need a policy that determines the linger duration.

When to migrate in a Linger-Longer scheduler depends on the local CPU utilization on the source and destination nodes, the duration of non-idle state and the migration cost. The question is when will the foreign job benefit from migration. Given the local CPU utilization and migration cost, the minimum duration of non-idle interval (called an episode) before migration is advantageous can be found. Any idle period shorter than the minimum duration will not provoke a migration. We can compute the minimum duration by comparing the two timing diagrams in Figure 1. In the non-idle state, utilization by the workstation owner starts at t_1 and ends at t_4. The average utilization of the non-idle node is h, and the average utilization on an idle node is l. We assume the execution time of the foreign job exceeds the duration of the non-idle state, so the foreign job completion time t_{f1} comes after t_4. Migration happens at t_2, and the cost is T_{migr}. The following equations compute the total job CPU time $T_{C,M}$ and $T_{C,S}$ with and without migration respectively.

$$T_{C,S} = (1-l)\cdot(t_1 - t_0) + (1-h)\cdot(t_4 - t_1) + (1-l)\cdot(t_{f1} - t_4)$$
$$T_{C,M} = (1-l)\cdot(t_1 - t_0) + (1-h)\cdot(t_2 - t_1) + (1-l)\cdot(t_{f2} - t_3)$$

Since the same amount of work should be done for both cases, $T_{C,S} = T_{C,M}$. We can solve the relationship between parameters.

$$t_{f1} - t_{f2} = (t_4 - t_2)\cdot\frac{1-h}{1-l} - (t_4 - t_3)$$

And, to get benefit from the migration, $T_{f2} <= T_{f1}$. We can then express it with interval variables as:

$$T_{nidle} \geq T_{lingr} + \left(\frac{1-l}{h-l}\right)\cdot T_{migr}$$

where $T_{nidle} = t_4 - t_1$ is the non-idle state duration, $T_{lingr} = t_3 - t_1$ is the lingering duration and the migration cost is denoted as T_{migr}. If we knew the non-idle state would last long enough to make migration advantageous, an immedi-

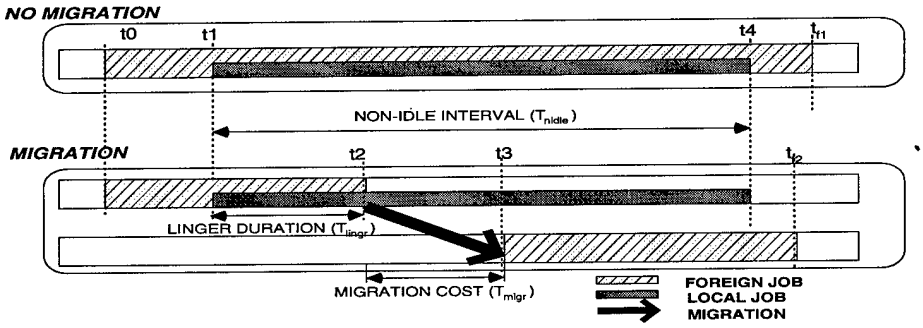


Figure 1: Migration Timing in Linger Longer

The timeline for migration using Linger-Longer scheduling. The top case shows a foreign job that remains on a node throughout an episode of processor activity due to local jobs. The lower case shows migration after an initial linger interval (t_1 to t_2) where the foreign job remained on the non-idle node.

ate migration would be the best choice. But because we don't know when the non-idle state will end, we have to predict it. We use the observations of Harchol-Balter and Downey[5], and Leland and Ott[8], which states that the median remaining life of a process is equal to its current age. So if a process has run for T units of time, we predict its total running time will be $2T$. Our use of this predictor is somewhat different since we use it to infer the duration of a non-idle episode rather than predict process lifetime. With this prediction, we can then compute the Linger duration by letting T_{nidle} be $2T_{lingr}$. If it is expected that the migration will benefit, it's better to migrate early. The lingering duration T_{lingr} will be:

$$T_{lingr} = (\frac{1-l}{h-l}) \cdot T_{migr}$$

So, the foreign job should linger T_{lingr} before migrating. For the non-idle interval shorter than T_{lingr} migration will be avoided. The migration cost consists of fixed part and variable part. The fixed part is for handling the process-related work at the source and destination nodes. The process transfer time varies on the network bandwidth and the process size. The simple equation is used for our experiments and can be easily extended for the different environment.

T_{migr} *=Processing_Time(source) + Process_size / network_bandwidth + Processing_Time(destination)*

We denote the policy of lingering on a node for T_{lingr}, LL. An alternative strategy of never leaving a node (called Linger-Forever) is denoted LF. This policy attempts to maximize the overall throughput of a cluster at the expense of the response time of those unfortunate foreign jobs that land (and are stuck) on nodes with high utilization.

3. Workload Analysis and Characterization

To evaluate our Linger-Longer approach, we need to characterize the workload of workstations to be used in such a system. The performance of the various scheduling disciplines for shared clusters depends on the characteristics of the workstation cluster such as time of day, day of week, and schedule of the primary users. We use the traces of the utilization patterns of existing workstations. Dusseau[4] and Acha[1] have also used this approach. Also, to evaluate our scheduling policy, we need data about individual requests for processors, at the granularity of scheduler dispatch records because of the fine-grained interaction between foreground and background processes.

It is not practical to record fine-grained requests for the long time periods required to capture the time of day and day of week changes in free workstations. As a result, we adopt a two level strategy to characterize the workload. First, we measure the fine-grained run-idle bursts at various level of processor utilization from 0% (idle) to 100% (full) utilization. We model a fine-grained workload as a random variable that is parameterized by the average utilization over a two-second window. This permits using a course-grained trace of workstation utilization to generate fine-grained requests for the processor.

3.1 Fine-Grain Workload Analysis

To analyze the fine-grained utilization of the CPU, we model processor activity as a sequence of run and idle periods that represent the intervals of time when the workstation owner's processes are either running or blocked. Since we give priority to *any* request by one of the local processes, a single run burst may represent the dispatching and execution of several local processes. Also, there is no upper bound on the length of a processor request since

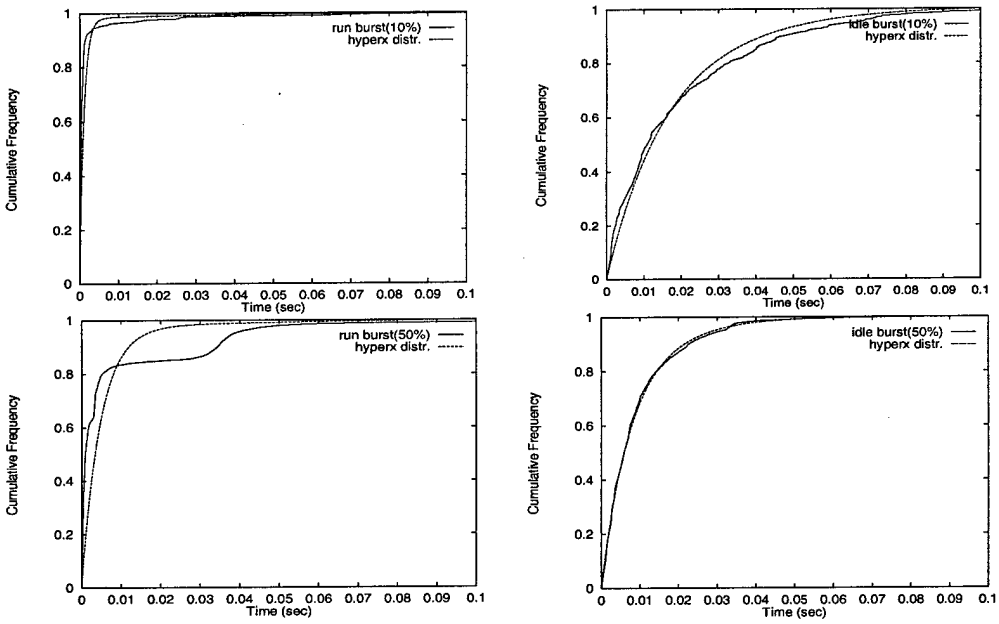


Figure 2: Run and Idle Burst Histograms

The CDF for the run and idle duration of local jobs. The first row is for 10% utilization, the second 50%.

we aggregate multiple consecutive dispatches due to time quanta into a single request.

To gather the fine-grained workload data, we used the tracing facility available on IBM's AIX operating system to record scheduler dispatch events. We gathered this data for several twenty-minute intervals on a collection of workstations in the University of Maryland, Computer Science Department. We then processed the data to extract different levels of utilization, and characterized the run-idle intervals for each level of utilization. We divided each trace file into two-second intervals and then computed the mean CPU utilization for each interval of time.

We divided utilization into 21 buckets ranging from 0% to 100% processor utilization. For each of the 21 utilization levels, we created a histogram of the duration of run and idle intervals for all two-second intervals whose average utilization was closest to that bucket. A selection of these histograms is shown in Figure 2. The solid line in the figure shows two sample distributions for low (10%) and medium (50%) CPU utilization. For the simulation, we generate a 2-stage hyper-exponential distribution from the mean and variance using a method-of-moment estimate[16 pg. 479]. The dashed line shows the CDF of the generated data. The curves almost exactly match in run and idle burst distributions.

To generate fine-grained workloads, we use linear interpolation between the two closest of the 21 levels of utilization. The values we derived from our analysis of the dispatch records are shown in Figure 3. The top-left curve shows the mean value of the run burst duration as a function of processor utilization. The upper-right graph shows the variance in the run burst. The bottom two graphs in Figure 3 show the idle duration mean and variance, respectively.

3.2 Coarse-Grain Workload Analysis

To generate the long-term variations in processor utilization, we use the traces collected by Arpaci et. al[4]. These traces cover data from 132 machines measured over 40 days, and contain samples every two seconds of: CPU usage, memory usage, keyboard activity, and a Boolean indicating idle/non-idle state. An idle interval is a period of time with the CPU less than 10% used and no keyboard action for 1 minute (called the recruitment threshold).

To assess the potential for Linger-Longer, we measured the overall CPU utilization and compared it to the CPU utilization during idle and non-idle intervals. On average, 46% of the time a machine was in a non-idle state. From the trace, we found that even non-idle intervals have very low usage, although it is somewhat higher than idle time. For example, 76% of the time in non-idle intervals, the processor utilization is less than 10%. The reason that these intervals of time are considered non-idle is due to keyboard activity and the requirement that a

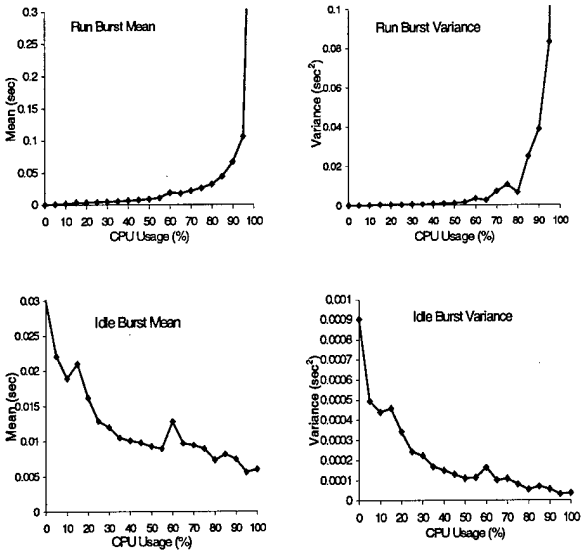


Figure 3: Workload Parameters

The mean and variance of the run and idle bursts seen in the fine-grained workload traces as a function of the processor utilization.

workstation have low utilization at least for 1 minute to be considered idle. This data hints at the potential leverage for a Linger-Longer approach to use short idle periods.

To meet our goal of allowing foreign jobs to linger on a workstation and at the same time not to interfere with local jobs, we need to ensure that enough real memory is available to accommodate the foreign job. Like processor time, we propose to use priority as a mechanism to ensure that foreign jobs do not consume memory needed by local jobs.[2] The idea is to divide memory into two pools: one for the local jobs and the other for foreign jobs. Whenever a page is placed on the free-list by a local job, the foreign job is able to use the page. Likewise, when the local job runs out of pages, it reclaims them from the foreign job prior to paging out any of its pages. A similar technique was employed in the Stealth scheduler[7].

To evaluate fully the availability of pages for foreign jobs, a complete simulation of the priority-based page replacement scheme is required. However, as an approximation of the available local memory, we analyzed the same workstation trace data used to evaluate processor availability to estimate available free memory. Each has 64Mbyte main memory. The CDF of available memory is shown in Figure 4. This graph shows that 90% of time,

[2] To implement this, we have added priority to the Linux paging mechanism.

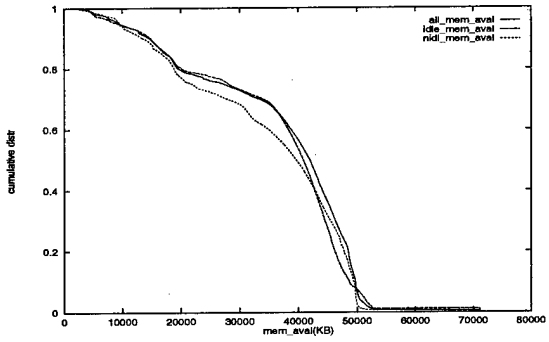


Figure 4: Distribution of Available Memory

The solid line shows the overall free memory and the two dashed lines show the free memory during idle and non-idle intervals. The y-axis shows the fraction of time that at least x KB of memory are available. Each workstation has 64 Mbyte main memory.

more than 14 Mbytes of memory available for foreign jobs, and that 95% of the time at least 10 MB of memory is available. Interestingly, there is no significant difference in the available memory between idle and non-idle states.[3] We feel that the amount of free memory generally available is sufficient to accommodate one compute-bound foreign job of moderate size.

4. Sequential Job Performance

In this section, we investigate how the Linger-Longer scheduling policy impacts the behavior of an owner's local processes, and then evaluate the performance of running sequential jobs in a 64 node cluster.

4.1 Linger-Longer Scheduling Impact

To understand the behavior of a Linger-Longer scheduling discipline we need to evaluate the impact of additional context switches of the priority-based linger mechanism on the node's foreground jobs. In this section we present a simulation study of the delay induced in a local process by a lingering foreign job.

A key question to evaluating the overhead of priority-based preemption is the time required to switch from the foreign job to the local one. There are two significant sources of delay in saving and restoring the context of a process: the time required to save the state of the registers and the time (via caches misses) to reload the process' cache state. On current microprocessors, the time to restore cache state dominates the register restore time. Paging by the VM system could increase the effective context

[3] One possible explanation for this that current versions of the UNIX operating system employ an aggressive policy to maintain a large free list.

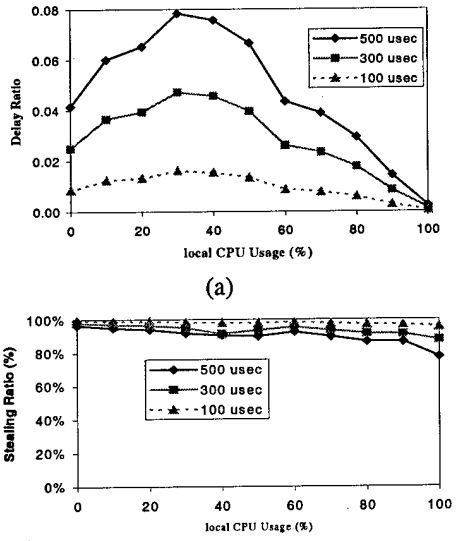


Figure 5: Local job Delay Ratio (LDR) and Fine-grain Cycle Stealing Ratio (FCSR)

Each curve shows the impact of three different effective context switch times (100, 300, and 500 microseconds). The graph (a) shows the delay experienced by foreground jobs at various level of utilization. The graph (b) shows the percent of the available *idle processor time made available to a compute bound foreign job at different levels of local job processor utilization.*

switch time, but our analysis of trace data shows significant memory available, and an implementation of Linger-Longer would include page priority (similar to processor priority). To estimate the effective context switch time, we use the results obtained by Mogul and Borg[10], and selected an *effective context-switch time* of 100 microseconds.

To evaluate the behavior of Linger-Longer we simulated a single node with a single compute bound (always runnable) process and various levels of processor utilization by foreground jobs. For each simulation, we computed two metrics: the local job delay ratio (LDR) and fine-grain cycle stealing ratio (FCSR). The LDR metric records the average slowdown experienced by local jobs due to the extra context switch delay introduced by background jobs. The FCSR metric records the fraction of the available idle processor cycles that are used by the foreign job.

Figure 5 shows the LDR and FCSR metrics for three different effective context switch times at various level of processor utilization by local jobs. For the chosen effective context switch time of 100 microseconds, the delay seen by the application process is about 1%. For context switch times up to 300 microseconds, the delay remains under 5%. However, when the effective context switch time is 500 microseconds, the overhead is 8%. In all of these cases, Lingering was able to make productive use of over 90% of the available processor idle cycles.

4.2 Sequential Jobs in a Cluster

We now turn our attention to the cluster level behavior of our scheduling policy. We first evaluate the behavior of a cluster running a collection of sequential jobs. We evaluated the Linger-Longer, Linger-Forever, Immediate-Eviction, and Pause-and-Migrate policies on a simulated cluster of workstations. We used a two-level workload generator to produce a foreground user workload for a 64-node cluster. Figure 6 shows the process that we use to generate fine-grained processor requests from long-term trace data. We randomly select a trace of a single node and map it to a logical node in our simulation. To draw a representative sample of jobs from different times of the day, each node in the simulation was started at a randomly selected offset into a different machine trace. The fine-grain resource usage is generated by looking up appropriate parameters, mean and variance, based on the current coarse-grain resource data from the trace files.

We then ran two different types of sequential foreign jobs on the cluster. Workload-1 contains 128 foreign jobs each requiring 600 processor seconds. This workload was designed to represent a cluster with a significant demand being placed on the foreign job scheduler, since on average each node had two foreign jobs to execute. Workload-2 contains 16 jobs each requiring 1,800 CPU seconds each. This workload was designed to simulate a somewhat lighter workload on the cluster since only ¼ of the nodes are required to run the foreign jobs. All foreign jobs are 8 Megabytes and migration takes places over a 10 Mbps Ethernet at an effective rate of 3Mbps (to limit the load placed on the network by process migration). We also assume that the foreign job is suspended for the entire duration of the migration. For each configuration, we computed four metrics:

Average completion time: The average time to completion of a foreign job. This includes waiting time before initially being executed, paused time, and migration time.

Variation: the standard deviation of job execution time (time from first starting execution to completion).

Family Time: The completion time of the last job in the family of processes submitted as a group.

Throughput: The average amount of processor time used by foreign jobs per second when the number of jobs in the system was held constant.

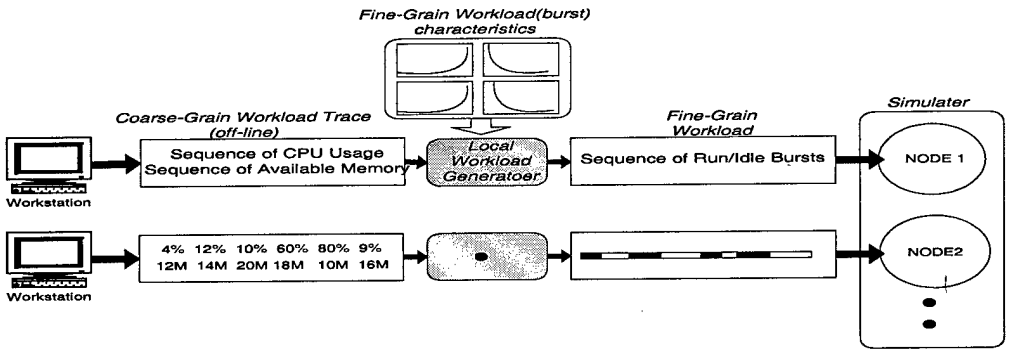


Figure 6: Local Workload Generation

The process of generating long-term processor utilization requests. By combining coarse-grained traces of workstation use with a short-term stochastic model of processor requests, long duration run-idle intervals can be generated.

The results of the simulation are summarized in the table in Figure 7. For the first workload, the average job completion time and throughput are much better for the Linger-Longer and Linger-Forever policies. Average job completion time is 47% faster with Linger-Longer than Immediate-Eviction or Pause-and-Migrate, and Linger-Forever's jobs completion time is 49% faster than either of the non-lingering policies. There is virtually no difference between the IE and PM in terms of average completion time. For the second workload, the average job completion time of all four policies is almost identical. Notice that the average job completion time ranges from 1,859 to 1,860 seconds; this implies that on average they were running 97% of the time. Since there is sufficient idle capacity in the cluster to run these jobs, all four policies perform about the same.

In terms of the variation in response time for workload-1, the LL policy is much better than either IE or PM. This improvement results from LL's ability to run jobs on any semi-available node, and thus expedite their departure from the system; so the benefit of lingering on a non-idle node exceeds the advantage of waiting for a fully free node. The LF policy has a somewhat higher variance due the fact that some jobs may end up on nodes that had temporarily low utilization when the job was placed there, but which subsequently had higher load. For workload-2, the availability of resources means that each policy has relatively little variation in its job completion time.

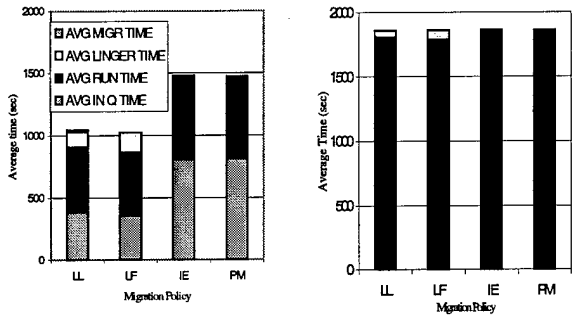
The third metric is "Family Time". This metric is designed to show the completion time of a family of sequential jobs that are submitted at once. This is a metric designed to characterize the responsiveness of a cluster to a collection of jobs that represent a family of jobs. For workload-1, the LL and LF metrics provide 36% improvement over the PM policy and 42% improvement over the IE policy. For workload-2, the LL and LF policies provide slight (1-3%) improvement over the IE and PM policies.

	Metric	LL	LF	IE	PM
Workload-1 (many jobs)	Avg. Job	1044	1026	1531	1531
	Variation	13.7%	20.5%	27.7%	22.5%
	Family Time	1847	1844	2616	2521
	Throughput	52.2	55.5	34.6	34.6
Workload-2 (few jobs)	Avg. Job	1859	1861	1860	1862
	Variation	0.9%	1.3%	1.3%	1.6%
	Family Time	1896	1925	1925	1956
	Throughput	15.0	14.7	14.5	14.5

Figure 7: Cluster Performance

For each of the four scheduling policies (LL, LF, IE, and PM), four performance metrics are shown for two different workloads.

The fourth metric we computed for the cluster-level simulations was throughput. The throughput metric is designed to measure the ability of each scheduling policy to make processing time available to foreign jobs. This metric is computed using a slightly different simulation configuration. In this case, we hold the number of jobs in the system (running or queued to run) constant for a simulated one-hour execution of the cluster. The number of jobs in the system is 128 for workload-1 and 16 for workload-2. The throughput metric is designed to show the steady-state behavior of each policy at delivering cycles to foreign jobs. Using the throughput metric, the LL policy provides at 50% improvement over the PM policy. Likewise the LF policy permits a 60% improvement over the PM policy. For workload-2, the throughput was very similar for all policies. Again, this is to be expected since the cluster is lightly loaded. For both workloads the delay, measured as the average increase in completion time of a CPU request, for local (foreground) processes was less than 0.5%. This average is somewhat less than the 1%



(a) (b)

Figure 8: Average Completion Time

The chart (a) shows the breakdown of the average time spent in each state (queued, running, lingering, or migrating) for workload-1 (many foreign jobs). The chart (b) shows the same information for workload-2 (few foreign jobs).

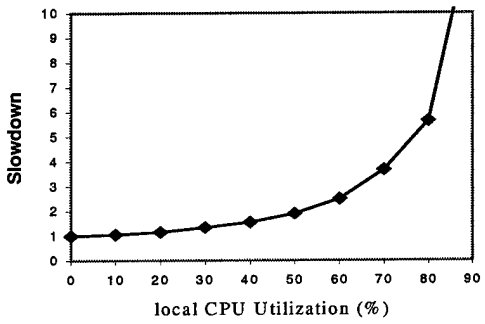


Figure 9: Parallel Job slowdown

The graph shows the slowdown of an eight-node parallel job vs. processor utilization by the foreground processes.

delay reported in the previous section since not all non-idle nodes have foreign processes lingering.

To better understand the ability of Linger-Longer to improve average job completion time, we profiled the amount of time jobs spent in each possible state: queued, running, lingering (running on a non-idle node), paused, migrating. The results are summarized in Figure 8. Figure 8(a) shows the behavior of workload-1. The major difference between the linger and non-linger policies is due to the reduced queue time. The time spent running (run time plus linger time) is somewhat larger for the linger policies, but the reduction in queuing delays more than offsets this increase. Figure 8(b) shows the breakdown for workload-2. With the exception that LL and LF spent small fraction of the time lingering, there is no noticeable difference between any of these cases.

The overall trends seen in the cluster level simulation show that Linger-Longer and Linger-Forever provide significant increased performance to a cluster when there are more jobs than available nodes, and that there is no difference in performance a low levels of cluster utilization.

5. Parallel Job Performance

The trade-offs in using Linger-Longer scheduling for parallel programs are more complex. When a single process of a job is slowed down due to a local job running on the node, this can result in all of the nodes being slowed down due to synchronization between processes. On the other hand, when a migration is taking place, any attempt to communicate with the migrating process will be delayed until the migration has been completed. However, we feel the strongest argument for using Linger-Longer is the potential gain in the throughput of a cluster due to the ability to run more parallel jobs at once. Improved throughput likely will come at the expense of response time, but we feel that throughput is the most important performance metric for shared clusters. To evaluate these different options, we simulated various configurations to determine the impact of lingering on parallel jobs.

5.1 Synthetic Parallel Jobs

To evaluate the impact of lingering on a single parallel job, we first simulated a bulk-synchronous style of communication where each process computes serially for some period of time, and then an opening barrier is performed to start a communication phase. During the communication phase, each process can exchange messages with other processes. The communication phase ends with an optional barrier. This synthetic parallel job model has been successfully used in [4] to explore various performance factors. We simulated an eight-process application with a 100 msec between each synchronization phase, and a NEWS[4] style of message passing within a communication phase.

The graph in Figure 9 shows the slowdown (compared to running on 8 idle nodes) experienced in the application's execution time when one node is non-idle and the CPU utilization by the foreground processes are varied from 0% to 90%. At utilization above 50%, the slowdown is so large that lingering slows down the jobs dramatically. A useful comparison of this slowdown is to consider alternatives to running on the non-idle node. The NOW[4] project has proposed migrating to an idle node when the user returns, however if there is a substantial load on the cluster we would have to keep idle nodes in reserve (i.e. not running other parallel jobs) to have one available. Alternatively, Acha et al.[1] proposed re-

[4] A process exchange messages only with it's neighbors in terms of data partitioning.

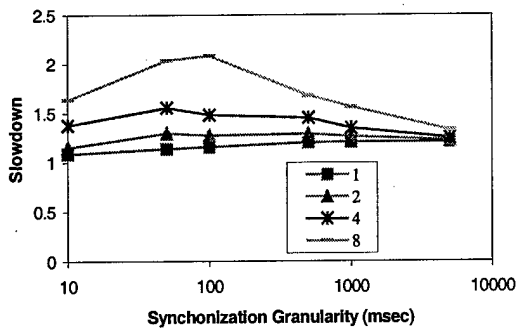


Figure 10: Synchronization Granularity vs. Slowdown

The graph shows the slowdown of running a parallel program with 1, 2, 4, or 8 non-idle nodes compared with running on all 8 idle nodes as a function of the synchronization granularity when the non-idle nodes have 20% utilization by local jobs.

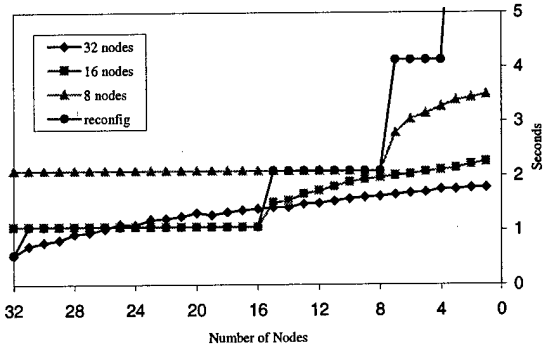


Figure 11: Linger Longer vs. Reconfiguration

The graph shows the completion time of a parallel job running on a cluster using several different scheduling policies. The x-axis shows the number of idle nodes. The first four curves show the Linger-Longer policy running using 8, 16, or 32 nodes. The fifth curve shows the reconfigure policy. For all curves, the local utilization of non-idle nodes is 20%.

configuring the application to use fewer nodes. However, many applications are restricted to running on a power of two number of nodes (or a square number of nodes). Thus the unavailability of a single node could preclude using many otherwise available nodes. Within this context, our slowdown of only 1.1 to 1.5 when the load is less than 40% is an attractive alternative.

One of the key parameters in understanding the performance of parallel jobs using Linger-Longer is the frequency of synchronization. Figure 10 shows the relationship between the granularity of communication and the slowdown experienced by an eight process bulk synchronous program. The x-axis is the computation time between communications in milli-seconds. Each of the four curves shows the slowdown when 1, 2, 4, and 8 nodes have 20% processor utilization by local jobs. The results show that larger synchronization granularity produces less slowdown. Also, for this level of loading, lingering provides an attractive alternative to reconfiguration since, even when four nodes are non-idle, the slowdown remains under a factor 1.5. (Note that reconfiguration with four nodes unavailable would have a slowdown of at least 2.)

We wanted to provide a head-to-head comparison of the Linger-Longer policy with the reconfiguration strategy. To do this, we simulated a 32 node parallel cluster. For each scheduling policy we considered the effect if the average processor utilization by the local jobs on a non-idle workstations was 20%. We defined the average synchronization frequency to be 500 msec. In this simulation, we didn't consider the time required to reconfigure the application to use fewer nodes, and assumed that the application was constrained to run on a power of two number of nodes. The results of running this simulation are shown in Figure 11. In each graph, the curves show the Linger-Longer policy using 8, 16, and 32 nodes, and the reconfiguration policy using the maximum nodes available. The Linger-Longer with k nodes means if k or more idle nodes are available in the cluster, the parallel job runs k processes on k idle nodes, otherwise it runs on all idle nodes available and some non-idle nodes by lingering.

The graph shows the results for 20%. The Linger-Longer policy outperforms the reconfiguration, when either 8 or 16 nodes are used. However, using 32 nodes and a Linger-Longer policy out performs reconfiguration when 5 or fewer non-idle nodes are used.

5.2 Real Parallel Jobs

To validate the results from the synthetic bulk synchronous application case, we ran several real parallel applications with Linger-Longer. To do so, we combined two different types of simulators: our Linger-Longer simulator generating local workloads and a CVM [6] simulator which can run shared memory parallel applications using ATOM binary rewriting tool [13]. We chose three common shared-memory parallel applications: `sor` (Jacobi relaxation), `water`(a molecular dynamics; from SPLASH-2 benchmark suite [17]) and `fft` (fast Fourier transformation) which have different computation and communication patterns.

First, we looked at how Linger-Longer on non-idle nodes slows down the parallel jobs. An 8 node cluster is used to run each application. The number of non-idle nodes and its local utilization were controlled. The results are summarized in Figure 12. For all 3 cases, when only one non-idle node is involved even with 40% local utili-

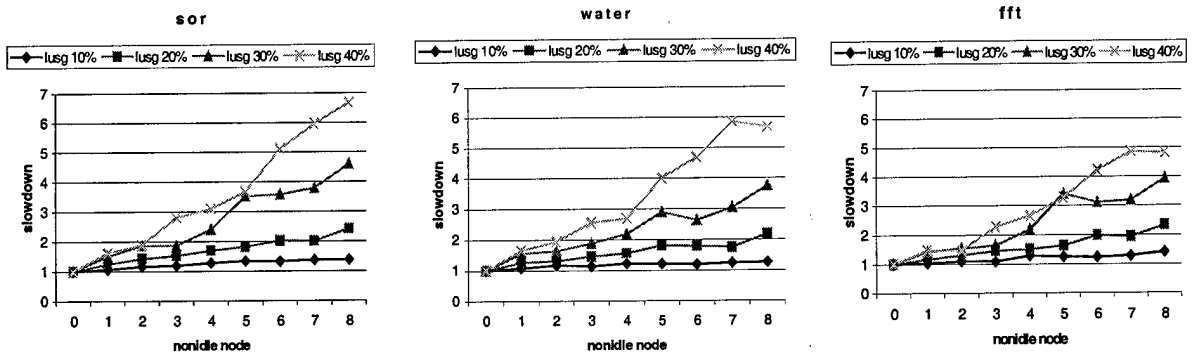


Figure 12: Slowdown by Non-idle nodes and their Local CPU Usage

The graphs show the slowdown of parallel jobs: `sor` (a), `water` (b) and, `fft`(c) as the number of non-idle nodes varies from 0 to all 8 with Linger-Longer. The cluster size is 8. The curves in each graph represent the different local utilization of non-idle nodes.

zation the slowdown (compared to running on eight idle nodes) reaches only 1.7. When more than half the nodes are non-idle, 0 to 20% local utilization looks endurable. Linger-Longer with 4 non-idle nodes and 20% local utilization causes only 1.5 to 1.6 slowdown. Even when all 8 nodes are non-idle, the job is slowed down by just above a factor of 2 for all three applications. Also, we can find that the slowdown is different on applications. `Sor` is the most sensitive to local utilization and the number of non-idle nodes. `Water` is less sensitive to local activity and `fft` is the least. A possible explanation is that `water` and `fft` have much more communication that `sor` and the time spent waiting on communication won't be affected as much by local CPU activity.

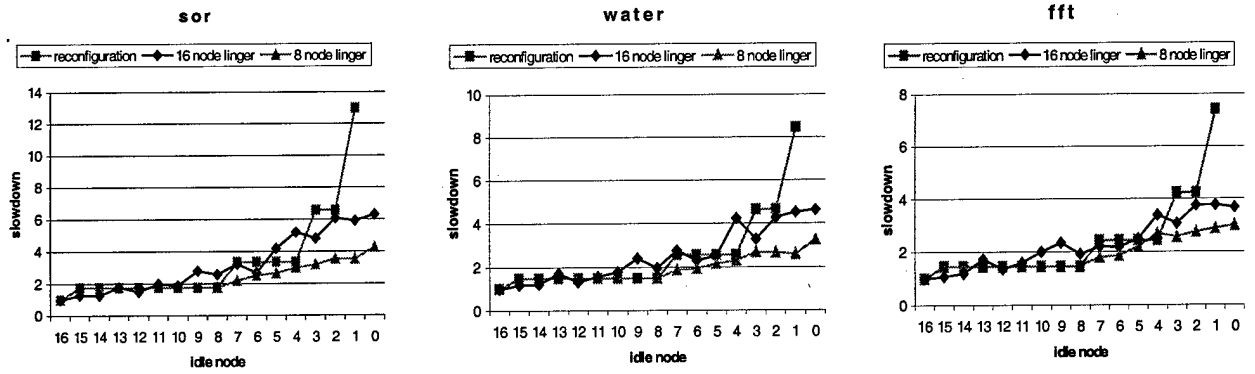
Again, we compared the Linger-Longer and reconfiguration policies for three real parallel application performance. The same assumptions as in the synthetic application case were maintained except for the fact that only a 16 node cluster was simulated. The results of running this simulation are shown in Figure 13. In each graph, the curves show the Linger-Longer policy using 16 and 8 nodes, and the reconfiguration policy using the maximum power of 2 number of idle nodes available. The graphs show the results for `sor`, `water` and `fft` when the local utilization for non-idle nodes is 20%. For all cases, the Linger-Longer policy using 16 nodes outperforms the reconfiguration when the number of idle nodes is at least 12. Considering the cost to reconfigure the parallel job to run time, the gain would be even bigger. However, when less than 8 idle nodes are left, lingering with 8 nodes looks much better than both lingering using 16 nodes and the reconfiguration policy. This indicates that a hybrid strategy of lingering and reconfiguration may be the best approach.

6. Related Work

Previous work on exploiting available idle time on workstation clusters used a conservative model that would only run jobs when the local user was away from their workstation and no local processes were runnable. Condor[9], LSF[19], and NOW[2] use variations on a "social contract" to strictly limit interference with local users. However, even with these policies, there is some disruption of the local user when they return since the foreign job must be evicted and the local state restored. The Linger-Longer approach permits slightly more disruption of the user, but tries to limit the delay to an acceptable level. One system that used non-idle workstations was the Stealth distributed scheduler[7]. It implemented a priority-based approach to running background jobs. However none of the tradeoffs in how long to run foreign jobs, or the potential of running parallel jobs was investigated.

Prior studies that investigated running parallel jobs on shared workstation clusters also employed fairly conservative eviction policies. Dusseau, et al.[4] used a policy based on immediate eviction. They were able to use a cluster of 60 machines to achieve the performance of a dedicated parallel computer with 32 processors. Acha et al.[1] used a different approach that reconfigured the parallel job to use fewer nodes when one became unavailable. This approach permitted running more jobs on a given cluster, although the performance of any single job would be somewhat reduced.

Process migration and load balancing have been studied extensively. MOSIX[3] provides load-balancing and preemptive migration for traditional UNIX processes. DEMOS/MP[12], Accent[18], Locus[15], and V[14] all provided manual or semi-automated migration of processes.



(a) (b) (c)

Figure 13: Linger-Longer vs. Reconfiguration for Shared-Memory Parallel Applications

The graphs show the slowdown of 3 parallel jobs running on a cluster of 8 nodes using several different scheduling policies. The x-axis shows the number of idle nodes. The first curve shows the reconfigure policy. The other 2 curves show Linger-Longer policy running using 16 or 8 nodes. For all curves, the local utilization of non-idle nodes is 20%.

7. Conclusions

In this paper we provided a workload characterization study that described the fine-grained requests for processor time at various levels of utilization and evaluated traces of workstation load at the 2-second level for long time durations. We then presented a technique to compose the workload and to generate fine-grained workloads for long intervals of time.

We have devised a new approach, called Linger-Longer to using available workstations to perform sequential and parallel computation. We presented a cost model that determines how long a process should linger on a non-idle node. The results for our proposed approach are encouraging, we showed that for typical clusters lingering can increase throughput by 60%, and on average cause only a 0.5% slowdown of foreground user processes.

For parallel computing, we showed that the Linger-Longer policy outperforms reconfiguration strategies when the processor utilization by the local process is 20% or less in both bulk synchronous and real data-parallel applications. However, reconfiguration outperforms Lingering for higher levels of local process utilization. The throughput improvement that would be possible by making more nodes available to run parallel jobs would likely offset some of this slowdown. An end-to-end evaluation of cluster throughput for parallel jobs is currently being investigated.

Currently, we are implementing the prototype based on the Linux operating system running on Pentium PCs. The strict priority-based scheduler and page allocation module have been developed and being evaluated.

Acknowledgements

The authors thank Pete Keleher and Dejan Perkovic for helping us combine their CVM simulator with the Linger-Longer simulator. Anurag Acharya and Remzi H. Arpaci have been very kind to provide the workload trace data. We also appreciate the useful comments of Bryan Buck and the anonymous reviewers.

References

1. A. Acharya, G. Edjlali, and J. Saltz, "The Utility of Exploiting Idle Workstations for Parallel Computation," *SIGMETRICS'97*. May 1997, Seattle, WA, pp. 225-236.
2. R. H. Arpaci, A. C. Dusseau, A. M. Vahdat, L. T. Liu, T. E. Anderson, and D. A. Patterson, "The Interaction of Parallel and Sequentail Workloads on a Network of Workstations," *SIGMETRICS*. May 1995, Ottawa, pp. 267-278.
3. A. Barak, O. Laden, and Y. Yarom, "The NOW Mosix and its Preemptive Process Migration Scheme," *Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments*, **7**(2), 1995, pp. 5-11.
4. A. C. Dusseau, R. H. Arpaci, and D. E. Culler, "Effective distributed scheduling of parallel workloads," *SIGMETIRCS*. May 1996, Philadelphia, PA, pp. 25-36.
5. M. Harchol-Balter and A. B. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *SIGMETRICS*. May 23-26, 1996, Philadelphia, PA, pp. 13-24.
6. P. Keleher, "The Relative Importance of Concurrent Writers and Weak Consistency Models," *ICDCS*. May 1996, Hong Kong, pp. 91-98.
7. P. Krueger and R. Chawla, "The Stealth Distributed Scheduler," *International Conference on Distributed*

Computing Systems (ICDCS). May 1991, Arlington, TX, pp. 336-343.

8. W. E. Leland and T. J. Ott, "Loadbalancing heuristics and process behavior," *SIGMETRICS*. May 1986, North Carolina, pp. 54-69.
9. M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," *International Conference on Distributed Computing Systems*. June 1988, pp. 104-111.
10. J. C. Mogul and A. Borg, "The effect of context switches on cache performance," *ASPLOS*. Apr. 1991, Santa Clara, CA, pp. 75-84.
11. M. W. Mutka and M. Livny, "The available capacity of a privately owned workstation environment," *Performance Evaluation*, **12**, 1991, pp. 269-284.
12. M. L. Powell and B. P. Miller, "Process migration in DEMOS/MP," *SOSP*. 1983, pp. 110-119.
13. A. Srivastava and A. Eustace, "ATOM: A system for Building Customized Program Analysis Tools," *SIGPLAN Conference on Programming Language Design and Implementation*. May 1994, Orlando, FL, pp. 196-205.
14. M. M. Theimer, K. A. Lantz, and D. R. Cheriton, "Premptable Remote Execution Facilities for the V-System," *SOSP*. Dec. 1985, pp. 2-12.
15. G. Thiel, "Locus Operating System, A Transparent System," *Computer Communications*, **14**(6), 1991, pp. 336-346.
16. K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. 1982: Prentice-Hall.
17. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 24-37.
18. E. R. Zayas, "Attacking the Process Migration Bottleneck," *SOSP*. 1987, pp. 13-24.
19. S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *SPE*, **23**(12), 1993, pp. 1305-1336.