A Comparison of Preemptive and Non-Preemptive Load Distributing

Phillip Krueger and Miron Livny

Computer Sciences Department
University of Wisconsin - Madison
Madison, WI 53706

ABSTRACT

Numerous load distributing algorithms have been proposed over the past several years, with widely varying characteristics. While some of these algorithms rely solely on non-preemptive process *placement*, others make use of preemptive process *migration*. Because the state of a process becomes considerably more complex after it begins execution, the mechanism necessary for migration is correspondingly more complex than that for placement, and may incur significantly greater resource overhead. In light of this complexity, as well as the consequent implementation expense, we consider whether the addition of a migration facility to a distributed scheduler already capable of placement can significantly improve performance. We examine performance over a broad range of workload characteristics and file system structures. We find that, while placement alone is capable of large improvement in performance, the addition of migration can achieve considerable additional improvement.

1. Introduction

In recent years, several studies, including [Livny82, Eager86], have shown load distributing to be an essential component of scheduling for distributed systems. While the *local scheduling* component of a distributed scheduler determines how the local resources at a single node are allocated among the resident processes, load distributing distributes the system workload among the nodes through *process transfer*. Process transfers can be performed either non-preemptively, through process *placement*, or preemptively, through process *migration*. Placement entails selecting a suitable node as the execution site for a process and initiating the process at that node. Later, if another node should become a better execution site, migration entails transferring the process to that node, where it continues executing.

Migration is more costly than placement, since the process state, which must accompany it to its new node, becomes much more complex after execution begins. First, implementing and maintaining the mechanism necessary to encapsulate, transfer and resume execution from this complex state is expensive. In addition, having implemented this mechanism, it is not obvious what performance improvement might result, since its resource overhead is likely to be much greater than that for placement [Powel83, Theim85]. In light of this two-pronged expense, we address the question: Is migration worthwhile, or can most or all of the performance improvement potentially available through load distributing be achieved using placement only?

For many computer systems, the best candidate for local scheduling is a preemptive discipline, such as Round-Robin [Klein76], that quickly provides an initial burst of service to newly-arrived processes. Such disciplines have long been used to provide service acceptable to the users of general-purpose single-processor systems, and have been shown to be necessary to provide equivalent performance for distributed systems [Krueg87]. We assume the use of such a local scheduling discipline in this study. Under such local scheduling, process transfers that are negotiated by a receiving node (referred to as *receiver-initiated* transfers [Eager86a]) are always migrations, since it is unlikely that a receiver-node would open negotiation with a potential sender at the moment that a new process arrived at the sender. Transfers that are *sender-initiated* however, may be either placements or migrations, depending on which process the sender chooses to transfer. While receiver-initiated migration has more obvious potential for performance improvement, sender-initiated migration may also improve performance. A load distributing algorithm that is free to choose any process to transfer, rather than being restricted to a newly-arrived process, may choose some other process that can be identified as likely to result in a larger improvement in performance. Though sender-initiated migration holds promise for future study, this paper focuses on receiver-initiated migration.

Ignoring, for a moment, the overhead incurred by load distributing, we can predict the conditions under which the addition of receiver-initiated migration will be most useful. Sender-initiated placement can improve performance whenever a process arrives at a busy node, which can potentially occur even when only a single process resides in the system. In contrast, the addition of receiver-initiated migration can improve performance only when a process completes at a time when the system contains more processes than nodes, so that the initial placement of processes resulted in some nodes servicing more than one process. For an M/M/m system composed of 10 nodes that is 65% utilized, the probability of this latter occurrence is only 0.1 [Laven83]. This probability rises to 0.45 if the utilization increases to 85%, but falls to 0.03 if the number of nodes increases to 20. From this simple analysis, the addition of receiver-initiated migration can be predicted to be useful only at high system loads or for systems containing few nodes. However, this result does not address the complications that arise in practical distributed systems, in which load distributing overhead is significant. This paper investigates the effect of this overhead on the ability of migration to improve performance.

Numerous load distributing algorithms have been proposed over the past several years, with widely varying characteristics. However, Eager, Lazowska and Zahorjan [Eager86] have noted that most load distributing algorithms can be categorized as following one of two archetypical strategies. While *load sharing* (LS) algorithms simply attempt to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for service, *load balancing* (LB) algorithms* go a step further by striving to equalize the workload among nodes. Though the LB strategy has

* LB can be considered a generalization of the Join-the-Shortest-Queue (JSQ) strategy, which has been studied by [Chow77, Ni81, Wang85]. While JSQ is limited to placement, LB may have a migration component.

CH2541-1/88/0000/0123$01.00 © 1988 IEEE

been shown to have the potential to provide better performance than LS when the overhead of load distributing is ignored [Krueg87], there are at least two reasons why an LS algorithm may prove advantageous for practical systems. First, when overhead is not ignored, an LS algorithm may provide better performance by avoiding process transfers that are not *cost-effective*, since they do not provide enough improvement in performance to justify their overhead [Krueg87a]. Second, particularly for distributed systems composed of workstations, ownership rights of individual nodes or groups of nodes may preclude LB [Nicho87, Litzk88]. To expose dependencies between improvement in performance and the particular strategy chosen, we consider both the LB and LS strategies.

Since the central issue of scheduling is performance, we must identify suitable performance indices with which to evaluate preemptive and non-preemptive load distributing. The goal of a scheduling policy is to allocate resources in such a way that the performance expectations of the users are most nearly met. User performance expectations generally center on the quality of service provided to the processes they initiate, for which both response time and response ratio are accepted measures. **Response time** is the total amount of time a process resides in the system, while **response ratio** is the response time per unit of service, which is the reciprocal of the *rate* at which the process receives service. The use of response ratio, similar to such human institutions as grocery store checkout lines for '10 items or less,' carries the assumption that a process requesting more resource use should expect to wait longer. Particularly for systems in which processes are associated with 'owners', such as multiuser timeshared systems, response ratio may more accurately reflect the user's perception of performance than response time. Alternatively, the use of response time may be more appropriate for systems in which all process have the same owner, such as real-time systems.

While the average quality of service provided is clearly an important performance index, how fairly service is allocated is also a common concern. For example, two users simultaneously initiating equivalent processes expect to receive about the same quality of service. While fairness is certainly a complex issue, a good first-order measure is the level of variation in quality of service provided to processes having the same priority. The most commonly used measures of variation, due to their pleasant mathematical properties, are the standard deviation and the coefficient of variation. The performance indices examined in this paper are the means and standard deviations of response time ($\overline{RT}$ and σ_{RT}, respectively) and response ratio ($\overline{RR}$ and σ_{RR}).

In the following pages, we show that while placement can indeed achieve much of the performance improvement available through load distributing, considerable additional improvement can be gained through migration. Under many conditions, this additional improvement would be obvious to the users of the system. We conclude that, while not as essential as placement, a migration facility is often a worthwhile investment.

We begin by describing the distributed system model assumed for this study. Using this model, we present data from simulation experiments representing a broad range of workload conditions and file system structures, identifying those systems and workloads in which a migration facility is most advantageous, together with the performance indices that are most improved. Finally, we summarize our results and draw detailed conclusions.

2. Distributed System Model

The model used in this study is an extension of the m*(M/M/1) family of distributed system models proposed by Livny [Livny82, Livny83], augmented to allow Processor Sharing (the theoretical limit of Round-Robin as the time quantum goes to zero [Klein67]) to be specified as the local scheduling discipline and to allow hyperexponentially, as well as exponentially, distributed task service demands. Hyperexponential distributions are important to consider, as those task service demand distributions that have been observed [Rosin65, Trive82, Zhou86] are poorly approximated by exponential distributions. The resulting m*(M/H/1) system consists of m functionally identical nodes, fully connected by a communication device. Tasks arrive independently at each node and join the local queue. After arriving, a load distributing algorithm allows process transfers. The distribution of interarrival times is exponential, so the external task arrival process of the system consists of m independent Poisson processes.

For this study, we assume a communication device with a single queue using the First-Come-First-Served queuing discipline. Since the communication device is not a bottleneck in our experiments, this simple model is sufficient. Processes are assumed to execute independently, with no intercommunication. Furthermore, we assume that nodes have equal processing bandwidths and that the service demands of processes arriving at different nodes are identically distributed. However, the rates at which processes initially arrive (as opposed to arriving as the result of transfer), however, may be different at different nodes. We refer to such a workload as having *heterogeneous initiation rates*. Such workloads may be common for some types of distributed systems, particularly those composed of workstations [Mutka87]. Since we are interested in scheduling for general-purpose computer systems, we assume the scheduler has no deterministic a priori information about process service demands. In addition, we assume that processes do not leave the system before completing service.

The task service demand distribution assumed is derived from data collected by Leland and Ott [Lelan86] from over 9.5 million processes executed by a VAX-11/780 under UNIX*. Using weighted least squares regression analysis, we have found that the 3-phase hyperexponential distribution having the following probability density function:

$$f(x) = .79(e^{-x/.31} / .31) + .192(e^{-x/2.8} / 2.8) + .018(e^{-x/27} / 27)$$

is a very good fit for the published data. The mean service demand, $\overline{X}$, deriving from this density is 1.27 seconds, while the coefficient of variation, C_X, is 5.3. We feel that this distribution is more useful for our purposes than that found by Leland and Ott ($1 - F(x) \approx rx^{-c}$) because, due to our choice of performance indices, we are interested in an accurate model for the largest *number* of processes (hence our use of least squares regression), rather than for the processes accumulating the greatest portion of CPU time. Thus, in addition to providing a usable density function, our model is a significantly better fit for all but the 0.2% of processes having the longest service demands. We generalize the above density function to form a family of functions sharing C_X, but varying in $\overline{X}$:

$$f(x) = (.79 / .243\,\overline{X})e^{-x/.243\overline{X}} + (.192 / 2.2\,\overline{X})e^{-x/2.2\overline{X}} + (.018 / 21.2\,\overline{X})e^{-x/21.2\overline{X}} \quad (2.1)$$

3. Simulation Study

In this section, we compare the improvement in performance resulting from sender-initiated placement with the *additional* improvement that results from augmenting placement with receiver-initiated migration. We study performance through simulation, since no analytic models are available that allow hyperexponentially distributed process service demands, heterogeneous process initiation rates, or the number of nodes participating in the system to be considered. These simulations are based on the assumption that negotiation and transfer require use of CPU resources at the sending and

* VAX is a trademark of Digital Equipment Corporation and UNIX is a trademark of AT&T Bell Laboratories.

receiving nodes, as well as use of the communication device. All simulation results presented have less than 10% error at the 90% confidence level.

3.1. Description of the Load Distributing Algorithm

In designing a load distributing algorithm, consideration must be given to its resource overhead. For example, since the resources used for load distributing, CPU and communication device bandwidth, are shared with user processes as well as other instances of the distributed scheduler, load distributing may increase queue delays experienced by user processes and decrease the responsiveness of load distributing. To best improve performance, it may be necessary to back off from the load balancing or load sharing strategy, performing only those transfers that most effectively improve performance. As a second example, the performance penalty caused by the overhead of negotiating with every node in the system to find the best *transfer partner* may be prohibitive, particularly when the likelihood of finding a partner is small. To maximize performance, a load distributing algorithm may be forced to limit negotiation to a subset of nodes. Additionally, because process transfers require time to complete, the length of the CPU queue at a node is not a sufficient measure of its load. Since transfers are not instantaneous, the queue length does not change as soon as a transfer is negotiated. However, a node should not overcommit. A load metric that avoids this problem augments the node's queue length with its *reservations* [Livny83], the change in queue length expected to be induced by processes that are in transit. Another consequence of the time required for negotiation and transfer is that a node that becomes idle is unable to immediately acquire new processes to execute even though processes wait for service at other nodes, resulting in a loss of available processing power in the system. To avoid this loss, *anticipatory* transfers to nodes that are not idle, but are expected to soon become idle, are necessary. As a final example, when resource overhead is not negligible, sophisticated criteria for selecting a process to migrate become advantageous. In our experiments, the process selected to migrate belongs to the set of processes that have been transferred least often among those residing at the sending node, and has executed for at least *F * sender CPU time required for transfer*, where *F* is a parameter of the algorithm and the CPU time required for a transfer is calculated from the size of the process and the CPU overhead per transfer message at the sender. The first of these criteria assures that an unusually small process is not repeatedly migrated, disproportionally degrading its service, and thus degrading the standard deviations of response time and response ratio (σ_{RR} and σ_{RT}). The second criterion avoids transferring very short processes, the overhead of which would severely degrade their response ratios. In addition, since the expected residual service demand of a process increases as it accumulates processing time, given that service demands are hyperexponentially distributed, this criterion avoids transferring those processes having the shortest expected residual service demands, which have the most transient effects on the load of the sender. Among the processes meeting these two criteria, the process selected to transfer has the smallest *migration size*, which may not exceed the parameter *MaxMigSize*, including the size of its executable image as well as any data that must accompany it.

To evaluate the performance impacts of placement and migration, we use variations of the *PollGen* algorithm [Krueg87a], which range from pursuing the LS strategy to the LB strategy, and from sender-initiated to symmetrically-initiated. In its strictest form, PollGen is symmetrically-initiated: A node initiates negotiation with probability *SendProb* on becoming *overloaded*, or with probability *RecvProb* on becoming *underloaded*. A node determines that it is overloaded or underloaded according to the relationship of its load to two static thresholds: A node becomes overloaded when the initiation of a process causes the load to be greater than one, and becomes underloaded whenever the completion of a process causes the node to become idle. Similarly, identification of a suitable transfer partner depends on two static thresholds: First, for all transfers, a suitable partner differs in load from the node initiating negotiation by at least two. Second, applying only to those transfers that are sender-initiated, the load of a suitable receiver is $\leq T$. This parameter controls the strategy pursued by PollGen. An algorithm following an LS strategy without anticipatory transfers has $T = 0$, while an algorithm implementing the LB strategy has $T = \infty$. In this section, we consider the following five PollGen variants, ranging in strategy from no load distributing (NoLD), through sender-initiated, to symmetrically-initiated load balancing (SymLB):

	SendProb	*RecvProb*	*T*
NoLD	0	0	0
SenderLS	1	0	0
SenderLB	1	0	∞
SymLS	1	1	0
SymLB	1	1	∞

How negotiation proceeds depends on whether it is sender or receiver-initiated. A potential receiver, being idle, chooses the *first* suitable transfer partner, so that it is not idle any longer than necessary. A potential sender, however, searches for the *best* partner: an idle node. To negotiate, a potential receiver polls a randomly chosen set of size *PollLimit* nodes until a suitable partner is found. If no suitable partner is found, the node remains idle. A potential sender polls a random set of size *PollLimit* nodes until an idle node is found. If no idle node is found, each suitable partner that has been found in this set is polled again, beginning with the node having the lowest load when last polled, until a partner that remains suitable is found. If no suitable partner is found, the node remains overloaded.

3.2. Simulation Assumptions

Based on observations of user processes executed on research computers at our department, we assume that the physical migration sizes of processes are independent of all other process characteristics and are exponentially distributed. Transfer messages are assumed to be divided into packets, with the CPU service demands required to process these packets, as well as those containing negotiation messages, preempting all other processing. This assumption implies that processes may not begin to receive service immediately on initiating. In [Krueg87b], it is shown that for such scheduling, the mean and standard deviation of wait ratio, and thus response ratio, are infinite when measured over the entire population of processes. To avoid this problem, processes having the shortest 1% of service demands are trimmed from the sample when measuring these performance indices. Default parameters of the simulation are:

Parameter	Value
Number of nodes (m)	20
System load (ρ)	0.85
Mean process CPU service demand ($\bar{X}$)	1.27 seconds
Mean process migration size	100K bytes
CPU service demand for transfer packets	.004 seconds
Maximum packet size	4K bytes
CPU service demand for negotiation messages	.002 seconds
Negotiation message size	32 bytes
Process initiation rates	Homogeneous
Communication device bandwidth	10 Mbits/sec.
PollLimit	5
F	0.1
MaxMigSize	100K bytes

For these default parameters, the mean CPU overhead of migrating a process of average size is 0.2 seconds, divided evenly between the sending and receiving nodes. To stress that this overhead is not as important in absolute terms as it is relative to the service demand of

a process, a key parameter in our presentation is *relative migration overhead*, which is the mean CPU overhead normalized by mean service demand, $\bar{X}$. This parameter, which is 0.16 for the above defaults, can be varied by varying either $\bar{X}$, the mean migration size, or the CPU overhead for transfer packets.

Assuming, for simplicity, that the input and output of a process reside on disk, the way migration and placement are modeled depends on the file system structure of the distributed system. If nodes have no local secondary storage, but rely on a shared disk server, placement can be accomplished simply by sending a message to another node specifying the program, input and output files. For such a system, migration carries much greater overhead than placement, since the process state considerably increases the size and complexity of the data that must be transferred. At the opposite end of the spectrum, if each node has local disk storage and no files are replicated, placement entails transferring the program and its input to the new node, and transferring the output back to the originating node when the program completes. Migration, for such a system, is not greatly more expensive than placement, since the process state does not greatly increase the size or complexity of the transfer. Between these endpoints lie systems having local secondary storage but some replication of files, and systems having 'minimal' local disks used only for swapping. We model process transfers for this spectrum of distributed systems as occurring in two logical parts. A message having constant size for all transfers (128 bytes) is followed by a message having a size corresponding to the migration size of the process being transferred. This size includes its executable image, its state description, and any input or output. For placement, the size of the second message is smaller than the migration size by a factor *PlaceFactor*. A small *PlaceFactor* models a system having no local disk storage, while *PlaceFactor* approaches 1 when nodes have local disks and no files are replicated.

3.3. Results

To compare the improvement in performance resulting from placement with that of migration, we compare the percent *reductions* in each of the performance indices that result from placement:

$$\textit{placement improvement} = 100\left[1-(\textit{Sender} \,/\, \textit{NoLD})\right]$$

with the *additional* reductions that result from augmenting placement with receiver-initiated migration:

$$\textit{additional migration improvement} = 100\left[1-(\textit{Sym} \,/\, \textit{Sender})\right]$$

To begin, we examine the interplay between the distributed file system structure and the ability of PollGen to improve performance. Figures 3.1 and 3.2, which plot percent improvement against *PlaceFactor*, show that in spite of the large improvement resulting from placement alone, the addition of migration can provide performance that is significantly better. This improvement is most obvious for the standard deviation of response time (σ_{RT}), which is generally *degraded*, rather than improved, when placement is used alone. The additional improvement from migration couteracts this degradation, allowing SymLS to achieve lower σ_{RT} than NoLD across the range of file system structures, while SymLB improves σ_{RT} for *PlaceFactor* ≤ 0.7. Even when migration is *greatly* more expensive than placement, at low values of *PlaceFactor*, it can significantly improve performance.

Not surprisingly, while an increase in *PlaceFactor* reduces the ability of placement to improve performance, it generally *increases* the ability of migration to additionally improve performance. Placement becomes less able to improve performance as it becomes increasingly expensive. In contrast, migration becomes better able to additionally improve performance as its overhead relative to placement decreases. We can conclude that load distributing algorithms relying solely on placement are best applied to distributed systems modeled by a small *PlaceFactor*, while migration is most valuable for systems modeled by a larger *PlaceFactor*.

While placement improvement from load balancing is often larger than that from load sharing, we observe the opposite for migration, where improvement is greater for LS than for LB. A related trend, decreasing *PlaceFactor* is more advantageous for SenderLB than SenderLS, and less harmful to SymLB than SymLS. Both these phenomena have the same cause: the LB strategy is more reliant on sender-initiated transfer, while LS is more reliant on receiver-initiated migration. Intuitively, since the sender-initiated component of an LB algorithm is less restricted in its search for a suitable destination node, placement carries a heavier load distributing burden for LB than for LS. Conversely, receiver-initiated migrations are more necessary to LS. This reliance of LB algorithms on placement and LS on migration can be quantified by calculating and comparing the rates of sender-initiated and receiver-initiated transfers for these two strategies. Derivations of these rates under assumptions of negligible load distributing overhead and exponentially distributed service demands [Krueg87a] show that, holding other parameters constant, the lower bound for each rate occurs when process initiation rates are homogeneous, while the upper bound is reached in the *single-source case*, when all processes initiate at a single node. Figure 3.3, which plots the transfer rates against system load, shows that the rate of sender-initiated transfers is higher for LB, while the receiver-initiated transfer rate is *lower* for LB, particularly when process initiation rates are heterogeneous. If migration is a great deal more costly than placement, LB may carry less overhead than LS, even though it has a higher overall transfer rate.

An important feature of figures 3.1 and 3.2 is that, for both placement and migration, improvement in mean response time ($\overline{RT}$) is greater than that in mean response ratio ($\overline{RR}$). Intuitively, since a given reduction in response time reduces the response *ratio* of a long-running process less than that of a short process, this difference in improvement implies that response time improvements have been more heavily 'allocated' to long processes than to short processes. Alternatively, since a given response ratio improvement reduces the response *time* of a long process more than that of a short process, this difference implies that the response ratio improvements have also been more heavily allocated to long processes. The reason for this bias is that the processes that are potentially most helped by a transfer (the transferred process and the processes left behind at the sender) must execute for some period before they 'recover' from the delay imposed by the overhead of the transfer. The amount that the response time or response ratio of a process is reduced as the result of a transfer increases with increasing residual service demand. Since the expected residual service demand of a long process is longer than that of a short process (regardless of the service demand distribution), a transfer that incurs a given level of overhead can improve the performance of a long process more than that of a short process. This mechanism is more clearly illustrated by figures 3.4 and 3.5, which present results of experiments in which mean process service demand ($\bar{X}$) is varied. The resulting data are plotted in terms of relative migration overhead (section 3.2), with longer $\bar{X}$ corresponding to smaller relative overhead. Similar results are shown if, instead of varying $\bar{X}$, mean migration size or CPU overhead per packet are varied. Figure 3.4 shows that, for placement, this bias in favor of long-running processes decreases with relative overhead, disappearing when overhead becomes negligible. Correspondingly, figure 3.1 shows that this bias decreases with *PlaceFactor*, nearly disappearing when load distributing overhead is solely from negotiation. However, figure 3.5 shows that the migration-induced bias follows a different trend, being significant even when overhead is negligible. The reason for this difference is that a bias in favor of long-running processes is *inherent* to migration, rather than arising solely as the result of overhead. Even when

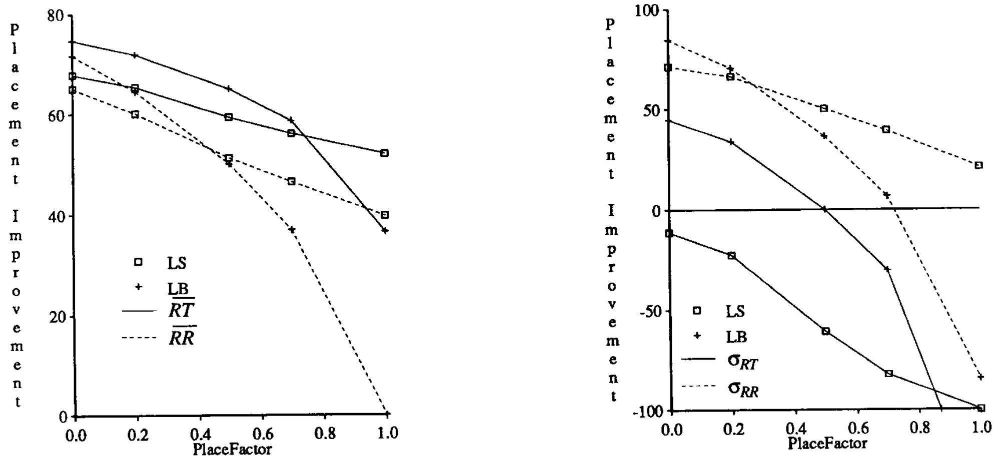


Figure 3.1 Percent placement improvement vs. *PlaceFactor*

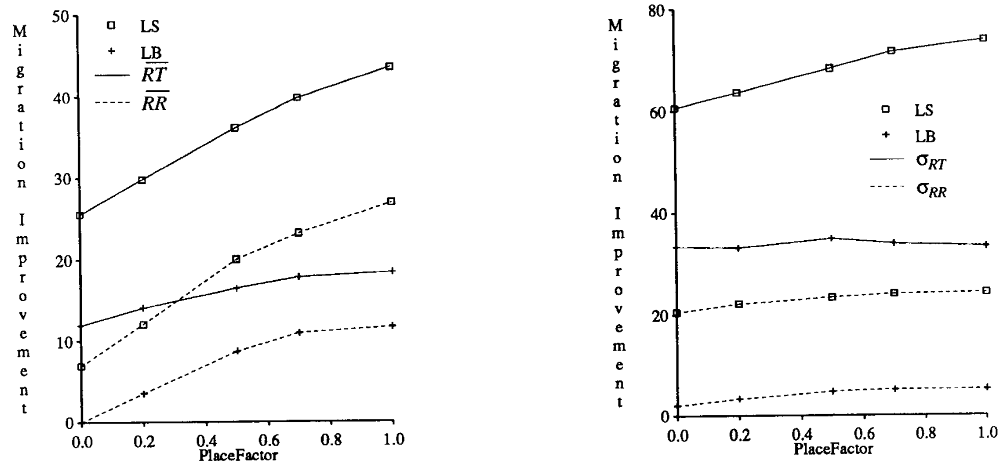


Figure 3.2 Percent migration improvement vs. *PlaceFactor*

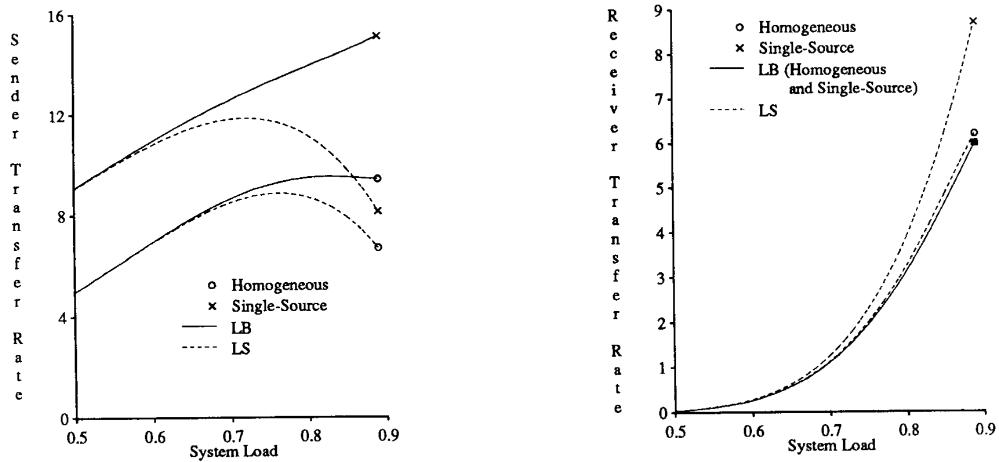
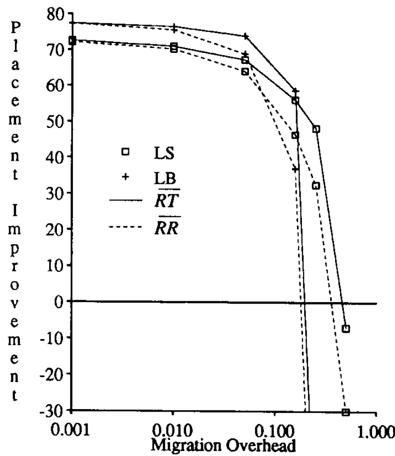


Figure 3.3 Sender-initiated (left) and receiver-initiated (right) transfer rates vs. system load

overhead is negligible, all processes that are immediately helped by a transfer have executed for some period of time before the transfer, so the mean service demand of these processes is greater than the overall mean. Thus, migration is more heavily biased toward long-running processes than placement.

Figure 3.5 shows that migration can provide significant improvement in performance over a broad range of relative overhead. The largest migration improvement is exhibited by SymLS at high levels of overhead. This improvement is more 'important' than the negligible improvement from SymLB, because LS is the strategy of choice under such conditions. Similar to the results of [Krueg87a], both load sharing algorithms provide better performance than either load balancing algorithm at high overhead, when the additional transfers performed by LB beyond those of LS are too costly to be worthwhile. Thus, at high overhead, an LS algorithm would likely be chosen regardless of whether a migration facility was included. For sufficiently high relative overhead, performance is *degraded*, rather than improved, when placement is used by itself. However, the large migration improvement achieved by SymLS counteracts this degradation, significantly broadening the range of overheads for which performance is better than without load distributing. For example, SenderLS provides lower $\overline{RT}$ than NoLD only when relative overhead < 0.5, but SymLS extends this range to 0.8. Similarly, SenderLS improves σ_{RT} only when relative overhead < 0.04, though SymLS extends this range to 0.4. This extension of the useful range of load distributing makes a migration facility particularly attractive for systems expected to perform well under a broad range of workloads, resulting in widely varying relative overhead.

The large increase in migration improvement resulting from SymLS at high levels of relative overhead, as well as the rapid decrease resulting from SymLB, can be better understood by examining figure 3.6, which plots migration improvement against system load. The trends at high system load are similar to those at high levels of overhead, because high overhead has the effect of increasing the *actual* system load. The increasing migration improvement from SymLS mirrors the results of Eager, Lazowska and Zahorjan [Eager86a], who showed that, while the effectiveness of sender-initiated LS drops off at high loads, receiver-initiated LS becomes increasingly effective with increasing system load. On the



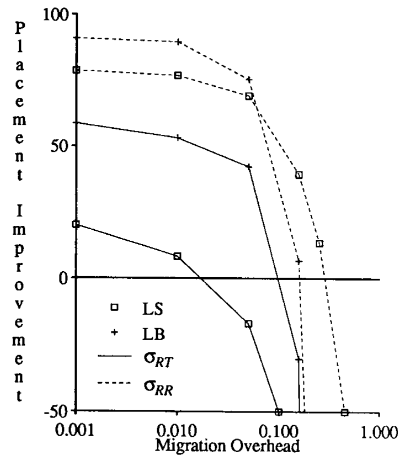
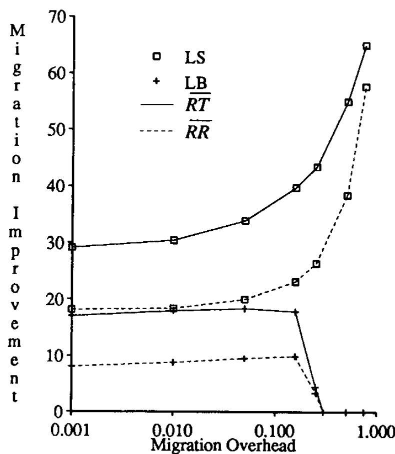


Figure 3.4 Percent placement improvement vs. relative migration overhead (*PlaceFactor* = 0.7)



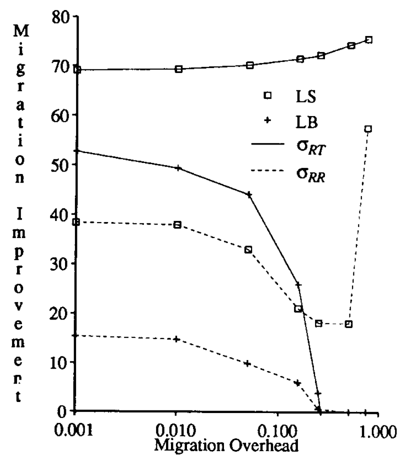
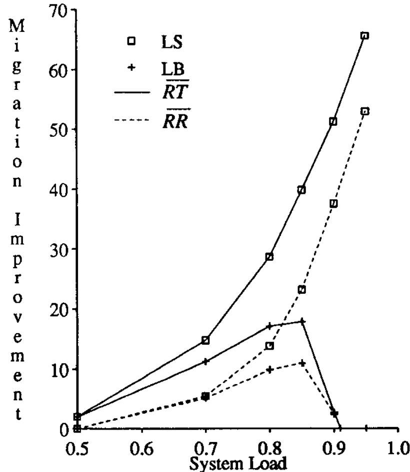


Figure 3.5 Percent migration improvement vs. relative migration overhead (*PlaceFactor* = 0.7)



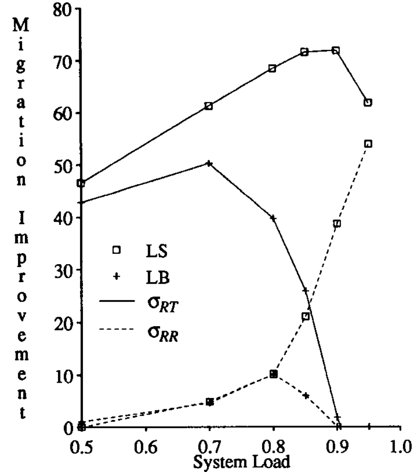


Figure 3.6 Percent migration improvement vs. system load (*PlaceFactor* = 0.7)

other hand, the dropoff in migration improvement from SymLB at high system loads occurs because its placement component allows less variation in load among nodes than that of SymLS. As a result, these migrations are not sufficiently advantageous to be worth the significantly increased CPU contention resulting from their overhead at high system loads. Similar to the case at high levels of relative overhead, the large improvement from SymLS at high system load is more important than the negligible improvement for SymLB. LS algorithms typically outperform LB at sufficiently high system load, where the additional transfers performed by LB increase CPU contention too much to be worthwhile. Following this trend, our simulations show LS to be the strategy of choice under such workloads. For sufficiently high system load, placement degrades performance when used alone. However, the large migration improvement resulting from SymLS counteracts this degradation. While SenderLS provides lower σ_{RT} than NoLD only when $\rho < 0.7$, SymLS extends this range beyond 0.95. Thus, receiver-initiated migration is particularly useful for systems that, at times, operate at high levels of utilization.

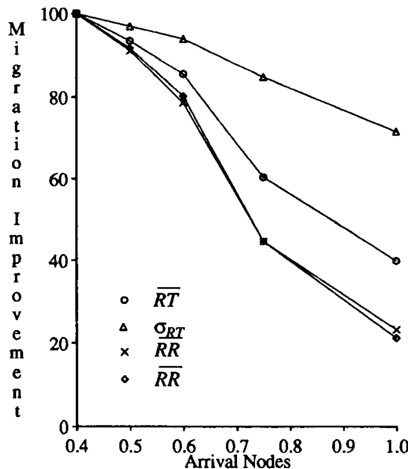


Figure 3.7 Percent migration improvement vs. portion of nodes that are arrival nodes (*PlaceFactor* = 0.7)

Still another environment in which receiver-initiated migration provides significant improvement in performance is one in which process initiation rates are heterogeneous. As predicted by [Krueg87], the level of heterogeneity has little effect on the performance of the LB algorithms, though the LS algorithms are strongly affected. For all workloads having sufficiently high levels of heterogeneity, our simulations show that the LB algorithms outperform the LS algorithms, and are thus preferable. However, particularly for distributed systems composed of workstations, ownership rights of individual nodes or groups of nodes may preclude LB, and LS may be necessary. Figure 3.7 plots migration improvement resulting from SymLS against the level of heterogeneity in process initiation rates. The level of heterogeneity is manipulated by varying the portion of the system nodes that are *arrival nodes*. As might be found in a workstation environment, the entire system workload is assumed to initiate at this subset of nodes, with an equal rate at each node, while no processes initiate at the remaining nodes. These results show that when process initiation rates are heterogeneous, the users of a distributed system would perceive a large improvement in performance as a result of a receiver-initiated migration facility.

4. Summary and Conclusions

In this paper, we have considered whether the addition of a process migration facility to a distributed scheduler already capable of process placement is a worthwhile investment. This question is particularly important to distributed operating system developers, because implementation of a migration facility is likely to be much more expensive than a placement mechanism. One must question whether this investment can provide significant gains in performance. Since, perhaps, the most obvious use for migration is to allow receiver-initiated process transfers, this paper has focused on the performance gains available through such use.

We have found that, while placement alone is capable of large improvement in performance, the addition of receiver-initiated migration, in many cases, achieves considerable additional improvement. This improvement is particularly important, because it broadens the range of workload conditions under which load distributing improves, rather than degrades, performance. The magnitude of this improvement depends on the workload characteristics of the system, as well as on the overhead of migration relative to placement, which is influenced by the file system structure. The key characteristic of a distributed system that can profit from the addition

of a migration facility is a high level of utilization. Although most systems are not specifically designed to operate at high utilization over the long term, many are exposed to such workloads for short periods of time. While these periods are short with respect to the lifetime of the system, they may be long enough to significantly affect the performance perceived by the users and should, therefore, not be ignored.

In addition to periods of high utilization, several other system characteristics increase the ability of migration to achieve performance gains:

- Process initiation rates are heterogeneous.
- The file system stores much locally at a node, with little replication at other nodes.
- The overhead of migrating a process tends to be high relative to its service demand.

Again, even for those systems in which these characteristics are intermittent, they may persist long enough to have a strong impact on user performance. It is important, then, to understand the effects of such conditions and to develop distributed scheduling algorithms that perform well, even under 'abnormal' circumstances which may otherwise threaten the stability of the system.

We have shown that receiver-initiated migration has an inherent bias toward reducing the wait times of long-running processes. As a consequence, receiver-initiated migration is more capable of improving $\overline{RT}$ than $\overline{RR}$, and improves σ_{RT} more than σ_{RR}, though large improvements in these latter indices may also be achieved. Also, we have shown that migration improves the performance of load sharing algorithms more than that of load balancing algorithms, though again, significant performance may also be achieved for load balancing.

In summary, throughout the range of file system structures considered, a wide range of workload conditions exist under which the additional improvement provided by receiver-initiated migration would be obvious to the users of the system. While not as essential to load distributing as placement, a migration facility is potentially a worthwhile investment.

References

[Chow77] Y. C. Chow and W. H. Kohler, "Dynamic load balancing in homogeneous two processor distributed systems," *Computer Performance*, North-Holland, (1977).

[Eager86] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering SE-12*, 5, pp. 662-675 (May 1986).

[Eager86a] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Dynamic Load Sharing," *Performance Evaluation 6*, 1, pp. 53-68 (March 1986).

[Klein67] L. Kleinrock, "Time-Shared Systems: A Theoretical Treatment," *Journal of the ACM 14*, pp. 242-261 (1967).

[Klein76] L. Kleinrock, *Queuing Systems: Volume 2, Computer Applications*, John Wiley & Sons (1976).

[Krueg87] P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies," *Proc. 7th International Conference on Distributed Computing Systems*, pp. 242-249 (September 1987).

[Krueg87a] P. Krueger and M. Livny, "When is the Best Load Sharing Algorithm a Load Balancing Algorithm?," Technical Report 694, University of Wisconsin—Madison, Dept. of Computer Sciences (April 1987).

[Krueg87b] P. Krueger and M. Livny, "Load Balancing, Load Sharing and Performance in Distributed Systems," Technical Report 700, University of Wisconsin—Madison, Dept. of Computer Sciences (July 1987).

[Laven83] S. S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press (1983).

[Lelan86] W. E. Leland and T. J. Ott, "Load-balancing Heuristics and Process Behavior," *Proceedings of PERFORMANCE '86 and ACM SIGMETRICS 1986*, p. 54 (May, 1986).

[Litzk88] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations," *Proc. 8th International Conference on Distributed Computing Systems*, (June 1988).

[Livny82] M. Livny and M. Melman, "Load balancing in homogeneous broadcast distributed systems," *Computer Network Performance Symposium*, pp. 47-55 (April 1982).

[Livny83] M. Livny, The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems, PhD Thesis, Weizmann Institute of Science, Rehovot, Israel (available as Technical Report 570, University of Wisconsin—Madison Computer Sciences) (August 1983).

[Mutka87] M. Mutka and M. Livny, "Profiling Workstation's Available Capacity for Remote Execution," Proceedings of PERFORMANCE '87 (December 1987).

[Ni81] L. M. Ni and K. Abani, "Nonpreemptive load balancing in a class of local area networks," *Proceedings of Computer Networking Symposium*, (December 1981).

[Nicho87] D. Nichols, "Using Idle Workstations in a Shared Computing Environment," *Proc. Eleventh ACM Symposium on Operating Systems Principles*, pp. 5-12 (8-11 November 1987) In *ACM Operating Systems Review 21*:5.

[Powel83] M. L. Powell and B. P. Miller, "Process migration in DEMOS/MP," *Proc. Ninth ACM Symposium on Operating Systems Principles*, pp. 110-118 (10-13 October 1983) In *ACM Operating Systems Review* 17:5.

[Rosin65] R. F. Rosin, "Determining a Computing Center Environment," *CACM 8*, 7, (July 1965).

[Theim85] M. Theimer, K. Lantz, and D. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *Proc. Tenth ACM Symposium on Operating Systems Principles*, pp. 2-12 (December 1985).

[Trive82] K. S. Trivedi, *Probability & Statistics with Reliability, Queuing and Computer Science Applications*, Prentice-Hall (1982).

[Wang85] Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers c-34*, 3, pp. 204-217 (March 1985).

[Zhou86] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," Technical Report, UCB/CSD 87/305, Computer Science Division, University of California, Berkeley (September 1986).