Attacking the Process Migration Bottleneck

Edward R. Zayas
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

(Currently at the Information Technology Center, Carnegie Mellon University)

Abstract

Moving the contents of a large virtual address space stands out as the bottleneck in process migration, dominating all other costs and growing with the size of the program. Copy-on-reference shipment is shown to successfully attack this problem in the Accent distributed computing environment. *Logical* memory transfers at migration time with individual on-demand page fetches during remote execution allows relocations to occur up to one thousand times faster than with standard techniques. While the amount of allocated memory varies by four orders of magnitude across the processes studied, their transfer times are practically constant. The number of bytes exchanged between machines as a result of migration and remote execution drops by an average of 58% in the representative processes studied, and message-handling costs are cut by over 47% on average. The assumption that processes touch a relatively small part of their memory while executing is shown to be correct, helping to account for these figures. Accent's copy-on-reference facility can be used by *any* application wishing to take advantage of lazy shipment of data.

1. Introduction

Process migration is a valuable resource management tool in a distributed computing environment. However, very few migration facilities exist for such systems. Part of the problem lies in providing an efficient method for naming resources that is completely independent of their location. The major difficulty, though, is the cost of transferring a computation's context from one system node to another. This context, which consists primarily of the process virtual address space, is typically large in proportion to the usable bandwidth of the interconnection medium. Moving the contents of a large virtual address space thus stands out as the bottleneck in process migration, dominating all other costs. As programs continue to grow, the cost of migrating them by direct copy will also grow in a linear fashion.

This research was supported by the AT&T Cooperative Research Fellowship Program. It was also supported by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520.

Any attempt to make process migration a more usable and attractive facility in the presence of large address spaces must focus on this basic bottleneck. One approach is to perform a *logical* transfer, which in reality requires only portions of the address space to be *physically* transmitted. Instead of shipping the entire contents at migration time, an *IOU* for all or part of of the data can be sent. As the relocated process executes on the new host, attempts to reference "owed" memory pages will result in the generation of requests to copy in the desired blocks from their remote locations. Context transmission times during migration are greatly reduced with this demand-driven *copy-on-reference* approach, and are virtually independent of the size of the address space. Processes are assumed to touch relatively small portions of their address spaces, justifying the higher cost of accessing each page during remote execution.

This paper describes the process migration facility built for the SPICE [12] environment at Carnegie Mellon University, which demonstrates the validity of using copy-on-reference transfer to attack the migration bottleneck. Section 2 describes the design of the Accent copy-on-reference mechanism, available to *any* application wishing to lazy-evaluate its data transfers. Accent's organization and abstractions not only provide the transparency needed to support migration, but lend themselves to the natural construction of such a mechanism. Section 3 show how the migration system capitalizes on copy-on-reference data delivery. Section 4 presents performance measurements taken on a set of representative processes that were migrated using different transmission strategies. Process relocations occur up to one thousand times faster using copy-on-reference transfers. While the amount of allocated data varies by four orders of magnitude across the processes studied, their transfer times are practically constant. The number of bytes exchanged between machines as a result of migration and remote execution drops by 58.2% on average, and message-handling costs are cut by 47.8%. The assumption that processes touch a relatively small part of their memory while executing is shown to be correct, helping to account for these figures. The detailed measurements are used to assess the effect of such copy-on-reference variations as prefetching in response to remote page requests and migration-time transfer of the address space portions resident in main memory. Section 5 compares the Accent migration work to other activity in the

field. Finally, Section 6 summarizes the lessons learned from the Accent migration system and considers future research directions suggested by this work.

2. The Accent Copy-On-Reference Mechanism

Accent's design and organization allows such intelligent virtual memory techniques as copy-on-write to be applied to data passed through the IPC system. It is this feature which aids in the construction of another intelligent strategy, copy-on-reference. This section begins by providing a quick overview of the Accent features that contribute to the natural construction of a transparent, generic copy-on-reference facility. Accent's *imaginary segment* abstraction serves as the basis for lazy data delivery, and is described next. The consequences of permitting imaginary objects to exist are explored, along with the method of shipping imaginary areas between machine boundaries.

2.1. Accent Features

The Accent IPC and virtual memory facilities are closely integrated, operating symbiotically. Unlike most message-based systems, a single Accent IPC message can hold *all* of the memory addressible by a process. Message contents are conceptually copied by value directly from the sender's address space into the receiver's. In reality, a message is first copied into the kernel's memory, buffered there until the recipient decides to accept it, and then copied out again. Accent provides the advantages of double-copy semantics for transferring message data between address spaces while still achieving the performance expected of a system that passes data by reference. This is possible through the use of a *copy-on-write* virtual memory mechanism by the IPC facility. If the amount of message data falls below a certain threshold, it is physically copied to the receiver. However, the kernel uses much faster memory-mapping techniques for messages exceeding this threshold. The receiver's virtual memory map is modified to provide access to the message data, and the region is marked copy-on-write for both parties. The two processes share this single copy of the data until either one tries to modify it. The deferred copy operation is then carried out, but *only* for the 512-byte page(s) affected. Files are accessed through an IPC interface and mapped in their entirety into process memory, allowing these techniques to be applied to their data as well. Since large amounts of data are often transferred through IPC messages and only rarely modified to any degree, this lazy strategy realizes performance that approaches by-reference transfer. Fitzgerald's study [3] reveals that up to 99.98% of data passed between processes in a system-building application did not have to be physically copied.

2.2. Imaginary Segments

Accent's copy-on-reference mechanism is based on a new segment class, the imaginary segment. Imaginary segment data is accessed not by direct reference to physical memory or a hard disk, but rather through the IPC system. Each imaginary segment is associated with a *backing IPC port* which provides memory management services for the object. When a process touches a page mapped to an imaginary segment, the high-level *Pager/Scheduler* process sends an *Imaginary Read Request* message to the region's backing port. The process with Receive rights for this port interprets the request and returns the required page in an *Imaginary Read Reply* message. The *Pager/Scheduler* completes the handling of the imaginary "fault" by mapping in the page and resuming the process attempting the access. Currently, page-outs for imaginary data are performed to the local disk at the site that touched the page. Any process may create an imaginary segment based on one of its ports, map all or part of it into its address space and pass this memory to another process via an IPC message. In effect, it transmits an "IOU" for the region's data, promising to deliver it as needed. The backing process continues to field page request messages aimed at the imaginary object until all references to it die out. At this point, Accent informs the backer of the object's demise by sending it an *Imaginary Segment Death* message.

2.3. Accessibility Maps

The existence of imaginary objects forces the operating system to provide a facility for determining the accessibility of any given virtual address range. Carelessly touching imaginary regions can result in deadlock. For example, an Accent process executing in the kernel context deadlocks if it touches a page with port-based backing. The faulter is caught holding the system critical section, preventing the backing process from executing the protected *Receive* operation needed to respond to the fault.

Accessibility Maps (or *AMaps*) were created to supply the necessary addressing information in Accent. Four different memory "distances" have been defined for AMaps:

1. **RealZeroMem**: This is a region that has been validated (allocated) by a process but has never been accessed. When memory is validated, it is conceptually filled with zeros. Accent postpones these filling operations until the pages are first touched. A special fault condition, the *FillZero* fault, is realized for this case. The only action the *Pager/Scheduler* process takes is to reserve a page of physical memory, fill it with zeros and create the appropriate virtual memory mappings. The disk is never consulted while handling this type of fault. In practice, Accent processes validate large amounts of virtual memory and only touch a small percentage. Lazy initialization of address space regions and the use of a special inexpensive fault-handling operation combine to make creation and maintenance of large virtual memory regions affordable. These *RealZeroMem* pages are considered immediately accessible to the process.

2. **RealMem**: The data in this type of region is either already present in physical memory or accessible by fetching the corresponding local disk page. The distinction between disk address mappings owned by the kernel and process mappings for the same data allows a disk page image to be resident without being visible to a user process. In this eventuality, the

Pager/Scheduler again simply fills in the missing user mapping and promotes the faulted process to a runnable state. If neither the disk nor the process mapping are available for the page, the matching disk block is determined. The page is brought in, and disk and process mappings for it are entered. *RealMem* pages are rated "moderately" accessible, since the system may have to go out to disk to get them.

3. **ImagMem:** The contents of memory regions mapped to imaginary segments have *ImagMem* accessibility. Touching a page in this accessibility class results in the the generation and processing of an imaginary fault, as described in Section 2.2. *ImagMem* pages are considered distantly accessible, since it may take an arbitrarily long time to complete a page fetch. The network state, the load on the machines involved and the amount of work being performed by the backing process all contribute variables to the service time.

4. **BadMem:** Attempting to touch a page in a region that hasn't been validated causes a true addressing error. Referencing a *BadMem* page invokes a debugger so the human user can analyze and properly terminate the delinquent process. Since referencing a *BadMem* page is illegal, its accessibility is considered infinitely distant.

2.4. Extending Imaginary Segments

As with the port abstraction, copy-on-reference access via imaginary segments depends on a user-level server for transparent extension across the network. The *NetMsgServer* process, running on each host, provides this service by changing its message fragmentation and reassembly algorithms to account for imaginary subranges. Using an AMap as a guide on both sides, the *RealMem* portions are physically transmitted to the remote location and placed in the corresponding locations in the reassembly buffer. The receiving *NetMsgServer* creates its own local ports and imaginary object(s) to stand in for the originals. Messages generated in response to faults on the remote imaginary objects are automatically channeled to the correct backing site.

On its own initiative, a *NetMsgServer* may cache the *RealMem* portions of a message destined to a remote site and instead pass IOUs for them, becoming the manager for that data. Senders can inhibit this behavior by setting the *NoIOUs* bit in the message header, which is inspected by the *NetMsgServer*. This action guarantees that non-imaginary message data is physically copied to the remote site.

3. Migration Using Copy-On-Reference

The SPICE migration facility is designed to take advantage of the copy-on-reference mechanism described in the previous section. This is done by special migration primitives which automatically separate out the context portions eligible for copy-on-reference shipment. Using these operations, the *MigrationManager* process on each machine has several options for context delivery to the new execution site.

3.1. *ExciseProcess* and *InsertProcess*

The *ExciseProcess* kernel trap allows the complete context of an active process to be removed from its current host. Accent contexts are divided into five components: the state of the Perq[1] microengine, the kernel stack if the process is executing in supervisor mode, the PCB, the set of port rights owned by the process and the virtual address space contents. While the first four parts combined only account for roughly 1 Kbyte, the address space contributes up to 4 gigabytes. Once a context is excised, the process ceases to exist. Since all port rights are passed transparently to the caller, there is no disruption to the set of processes capable of naming these ports.

ExciseProcess delivers a process context in two separate IPC messages, ready for shipment to the new execution site. The *Core* message contains the first four context pieces, which must be physically copied to the remote site. It also carries an AMap describing the entire process address space. The *RIMAS*[2] message contains all of the *RealMem* and *ImagMem* portions of the address space, collapsed into a contiguous area. This allows the caller to fit one or more excised address spaces into its own memory at one time. It also allows the bearer to cache the *RealMem* portions and substitute its own imaginary objects in the *RIMAS* message. If the migration agent doesn't wish to actively manage the excised address space, it simply turns off the *NoIOUs* bit in the *RIMAS* message header as described in Section 2.4, prompting the local *NetMsgServer* to assume backing services for the memory.

The counterpart for *ExciseProcess* is *InsertProcess*, which uses the two context messages to recreate the process. Since the messages are self-contained, they do not have to be preprocessed in any way. The embedded port rights are passed to the new incarnation. Using the AMap for guidance and the *RIMAS* data for ammunition, the process address space mappings are restored. The reconstituted process is finally placed into the kernel queue representing the original execution status.

3.2. The *MigrationManager* Process

Each SPICE machine wishing to participate in process migration runs a simple *MigrationManager* process. This server accepts and executes commands to perform migrations. Given a process name, it uses the *ExciseProcess* primitive to acquire the process context. The two context messages are then simply sent to the *MigrationManager* at the new execution site, which uses *InsertProcess* to reconstruct the target process.

The current *MigrationManager* doesn't attempt sophisticated address space management for the processes it extracts. If asked to use copy-on-reference transfer for process memory, the *MigrationManager* allows the intermediary *NetMsgServers* to cache the data and become its backer.

---

[1]Designed by Perq Systems, Inc., the Perq workstation has a microcoded CPU, 16-bit words and a 150 nanosecond cycle time. It's rated at between 1/5 and 1/2 the speed of a Vax-11/780, depending on the instruction set used. More detailed specs are in [3].

[2]*RIMAS* stands for *R*eal and *I*maginary *M*emory *A*ddress *S*pace.

4. Evaluation

This section summarizes the results of experiments carried out on the augmented Accent testbed system to determine the effectiveness of the copy-on-reference technique in reducing the dominant cost of migration: transfer of large process address spaces. Representative processes were chosen and monitored as they were migrated with the different strategies of interest. These programs are implemented in a variety of languages, perform widely different tasks and differ greatly in memory requirements and access patterns. Figures on their address space composition and utilization are presented, along with the basic costs of the migration primitives used to extract and insert process contexts on a host. Based on such metrics as the quantity and distribution of byte traffic, message processing costs and end-to-end elapsed times, copy-on-reference is shown to be superior to the brute-force method. Two variations on the basic lazy-transfer theme were simulated using the detailed performance figures and also evaluated. While prefetch of between 1 and 15 nearby pages in response to imaginary faults proved to be a valuable optimization, the shipment of process resident sets (as an approximation to their working sets [2]) was found to be generally detrimental. Overall, the experiments show that this lazy transfer technique significantly reduces the dominant context transmission costs by exploiting the fact that processes tend to use only small portions of their address spaces during execution.

4.1. Representative Processes

Several processes were chosen to undergo relocation, each representing a class of programs sharing similar attributes. The results obtained for these representatives should be characteristic of other programs in their class.

1. **Minprog**: This program is used to judge the effects of the various transmission strategies on a "minimal" program. Written in Perq Pascal, Minprog prints a message on the standard output, waits for user input and terminates. Measurement of this program is the equivalent of timing the "null trap" when exploring operating system performance.

2. **Lisp-T**: Accent supports the SPICE Lisp dialect, complete with a customizable screen editor and compiler. The Lisp-T trial resembles Minprog in that the minimum computation is performed. After migration, the Lisp interpreter is simply asked to evaluate T. This process represents simple Lisp programs, or larger Lisp jobs migrated late in life. The primary difference between Lisp-T and Minprog is the amount of address space used. Lisp processes validate their entire 4 gigabyte address spaces at birth, compared to Minprog's use of only 330 Kbytes.

3. **Lisp-Del**: This Lisp process performs a significant amount of computation and I/O. Immediately after migration, a Delaunay triangulation package written at Carnegie Mellon by Rex Dwyer is loaded. Utilizing a divide-and-conquer algorithm on a random set of points, this package displays its actions graphically on the screen as the triangulation is built.

4. **PM-Start**: The Pasmac macro processor for Perq Pascal represents the class of programs whose primary duty is to read files from the disk, process them in some way and write the results back out. In this instance, a 164 Kbyte file containing the program with macro references imports five definition files totaling 114 Kbytes. Migration takes place at the point the first definition file is being accessed.

5. **PM-Mid**: This trial postpones migration of the above macro processor until all of the definition files have been read in. Thus, the file images have become part of the process context and are carried along by the migration. The relocated program doesn't perform any more file accesses until it writes out the expanded program text.

6. **PM-End**: The final trial involving the Perq Pascal macro processor further postpones migration until the original file has almost been completely expanded. With little computation left to perform, this trial reveals the performance of the various migration strategies on processes near the end of their lifetimes.

7. **Chess**: A chess program written by Charly Drechsler at Siemens rounds out the group. It performs a large amount of computation to evaluate board positions and generate moves, but doesn't use a lot of its address space. A graphical representation of the chess board is displayed on the screen along with a game clock. The game clock ticks every second, so screen updates occur at least that often. Migration takes place as soon as the program initializes itself and draws the first screen image.

4.2. Address Space Analysis

4.2.1. Composition

Table 4-1 expresses the address space sizes and breakdowns of the representative processes at migration time.

	Real	RealZ	Total	% RealZ
Minprog	142,336	187,904	330,240	56.9
Lisp-T	2,203,136	4,225,926,144	4,228,129,280	99.9
Lisp-Del	2,200,064	4,225,929,216	4,228,129,280	99.9
PM-Start	449,024	501,760	950,784	52.8
PM-Mid	446,464	466,432	912,896	51.1
PM-End	492,032	398,848	890,880	44.8
Chess	195,584	305,152	500,736	60.9

Table 4-1: Representative Address Space Sizes in Bytes

Listed for each representative process is the amount of non-zero data it addresses (*Real*), the allocated but untouched zero-filled memory (*RealZ*), the total memory addressed (*Total*) and the percentage of the overall process memory taken up by allocated, untouched zero-filled regions (*% RealZ*). Memory quantities are in bytes.

There is wide variance in the amount of validated memory in

the representative Accent processes. The space utilized by the biggest process is a factor of 12,803 larger than that of the smallest. This is the consequence of the way Lisp processes manage their address spaces. The amount of *RealMem* mapped into processes doesn't vary nearly as much, only by a factor of 15 for these samples. Notice that *RealZeroMem* forms a significant part of all process address spaces, more than half even in most non-Lisp examples.

4.2.2. Resident Set Analysis

The process resident set sizes at migration time and their relationships to their host address spaces are shown in Table 4-2.

	RS Size	% of Real	% of Total
Minprog	71,680	50.4	21.7
Lisp-T	190,464	8.6	0.005
Lisp-Del	190,464	8.7	0.005
PM-Start	132,096	29.4	13.9
PM-Mid	190,976	42.8	20.9
PM-End	302,080	61.4	33.9
Chess	110,080	56.3	22.0

Table 4-2: Representative Resident Sets

Listed is the resident set size in bytes at migration time (column *RS Size*) for each representative, as well as the relative size compared to the process non-zero data (*% of Real*) and total allocated space (*% of Total*).

The range of resident set sizes is even narrower than that of the *RealMem* figures in Section 4.2.1, a factor of only 4. With the unrealistic Minprog process excluded, the factor drops to 2.7. This implies that the transfer of a process resident set will contribute a relatively consistent delay to the migration operation. Because of the amount of memory involved, resident set transfers are a significant expense. Viewing resident set transfer as a middle ground between a pure-copy transfer and a pure-IOU strategy appears reasonable, since the resident sets are roughly half as large as the *RealMem* in most cases. However, Section 4.3.4 demonstrates that this added expense at migration time doesn't translate into better overall performance.

4.2.3. Address Space Utilization

As postulated, Accent processes reference a small portion of their address spaces on average in their lifetimes. Table 4-3 reveals the amount of data transferred between machines during the trials in relation to address space size. Percentages are listed for the pure-IOU and resident set strategies without prefetching (pure-copy transmits 100% of *RealMem* by definition). Pure-IOU figures (the first column) indicate the portions actually touched by the process at the remote site.

	IOU	RS
Minprog	8.6 [3.7]	50.4 [21.7]
Lisp-T	3.0 [0.002]	9.0 [0.005]
Lisp-Del	16.5 [0.009]	17.4 [0.009]
PM-Start	58.0 [27.4]	76.0 [35.9]
PM-Mid	51.5 [25.2]	77.5 [37.9]
PM-End	26.9 [14.8]	72.5 [40.1]
Chess	35.6 [13.9]	66.0 [25.8]

Table 4-3: Percent of Address Space Accessed

For each representative process, the portion of the address space transferred to the new site is given for the pure copy-on-reference (*IOU*) and resident set (*RS*) strategies. The first number in each column represents the percent of the allocated, non-zero (*RealMem*) memory shipped, while the number in square brackets reports the percent of the total allocated address space. By definition, the pure-copy technique transfers 100% of non-zero data.

The Lisp representatives, while they have the largest address spaces, touch the smallest percentage in the course of execution. This applies even when performing a considerable amount of computation and I/O, as in the case of Lisp-Del. The Pasmac macro processor trials showed the highest address space utilization, as their mapped disk files are touched sequentially and in their entirety. In all cases, the resident set transfer method accessed larger portions of the address space, bringing over pages that are never used. This is especially acute for Pasmac. Since physical memory under Accent tends to act as a disk cache, old file pages that have *already* been processed are still sent to the new execution site. This explains why the pure-IOU method references significantly less of the Pasmac process address space the later in life it is migrated while the resident set approach results in nearly constant utilization.

4.3. Migration Phase Timings

Migration under Accent may be broken down into three phases:

1. Packaging and unpackaging the process context at the source and destination hosts.
2. Transferring the context between the sites.
3. Running the program at its new location.

This section examines how the migration strategies and their variations perform in each of these phases, and also presents an end-to-end analysis. Copy-on-reference transfers are shown to greatly reduce the time spent in the transfer phase while only moderately increasing remote execution times, resulting in significant overall performance improvements. While the first phase is insensitive to the migration strategy chosen, the experiments reveal some interesting facts about Accent's virtual memory system.

4.3.1. Process Excision and Insertion

Two operations dominate the removal and packaging of a process context, as revealed by Table 4-4: AMap construction for the target address space and the collapse of process memory into a contiguous chunk.

	AMap	RIMAS	Overall
Minprog	.37	.36	.82
Lisp-T	2.12	.59	2.79
Lisp-Del	2.46	.73	3.38
PM-Start	.98	.63	1.67
PM-Mid	1.01	.68	1.74
PM-End	1.4	.94	2.45
Chess	.37	.43	1.00

Table 4-4: Process Excision Times in Seconds

The rightmost column of this table lists the amount of elapsed time used by the *ExciseProcess* kernel trap on each of the representatives (*Overall*). Also listed are the individual timings for the two dominant activities carried out during extraction: AMap construction (*AMap*) and creation of the IPC message containing the condensed process address space (*RIMAS*).

There are two reasons why AMap construction is an expensive operation under Accent. The complex process map organization chosen to support sparse address spaces and copy-on-write makes it difficult to determine accessibility for *ranges* of addresses. Also, the lazy update algorithm employed for process maps often forces a costly search of

system virtual memory tables. The Lisp processes take the longest to service, as might be expected. The Minprog and Chess programs have small, uncomplicated address spaces and hence require the shortest amount of time.

While process memory is rearranged into a compact form and delivered to the migration agent via memory-mapping techniques instead of physical copies, it is still an important part of the excision activity. Address space collapses contribute a much smaller variation to excision times than does AMap construction. Overall, excision times vary only by a factor of 4, compared to the 4 *orders of magnitude* difference in the address space contents.

Process reincarnation given the two context messages involves reestablishing the microcode and port state of the process, along with setting up its address space to correspond to the original structure. The times required to insert the transferred contexts into the new site ranged from 263 milliseconds for Minprog to 853 milliseconds for Lisp-Del. Address space reconstruction is the major factor in the insertion operation, and times are very similar to the *RIMAS* creation times during context extraction. As with other portions of the migration mechanism, this insertion costs grow much more slowly than the address spaces involved, only a factor of 3.3.

4.3.2. Context Transfer Times

Approximately one second is required to transmit the *Core* context message (microstate, PCB, port rights) in all cases. These messages differ by a small number of bytes, since some AMaps are slightly larger than others. The real variation involves the delivery of the *RIMAS* message (valid, non-zero address space) under the different transfer strategies. Table 4-5 provides these timings.

	Pure-IOU	RS	Copy
Minprog	.16	5.0	8.5
Lisp-T	.16	25.8	157.0
Lisp-Del	.17	25.8	168.5
PM-Start	.15	9.0	30.8
PM-Mid	.16	13.0	28.1
PM-End	.19	20.5	31.0
Chess	.21	7.7	11.7

Table 4-5: Address Space Transfer Times in Seconds

Address space transfer times are closely clustered for the copy-on-reference approach (*IOU*), but vary considerably for the resident set (*RS*) and pure-copy (*Copy*) techniques.

Times required to ship process address spaces pure-IOU are nearly independent of the amount of memory involved. Use of pure-copy doesn't fare nearly as well, where *RIMAS* trans-

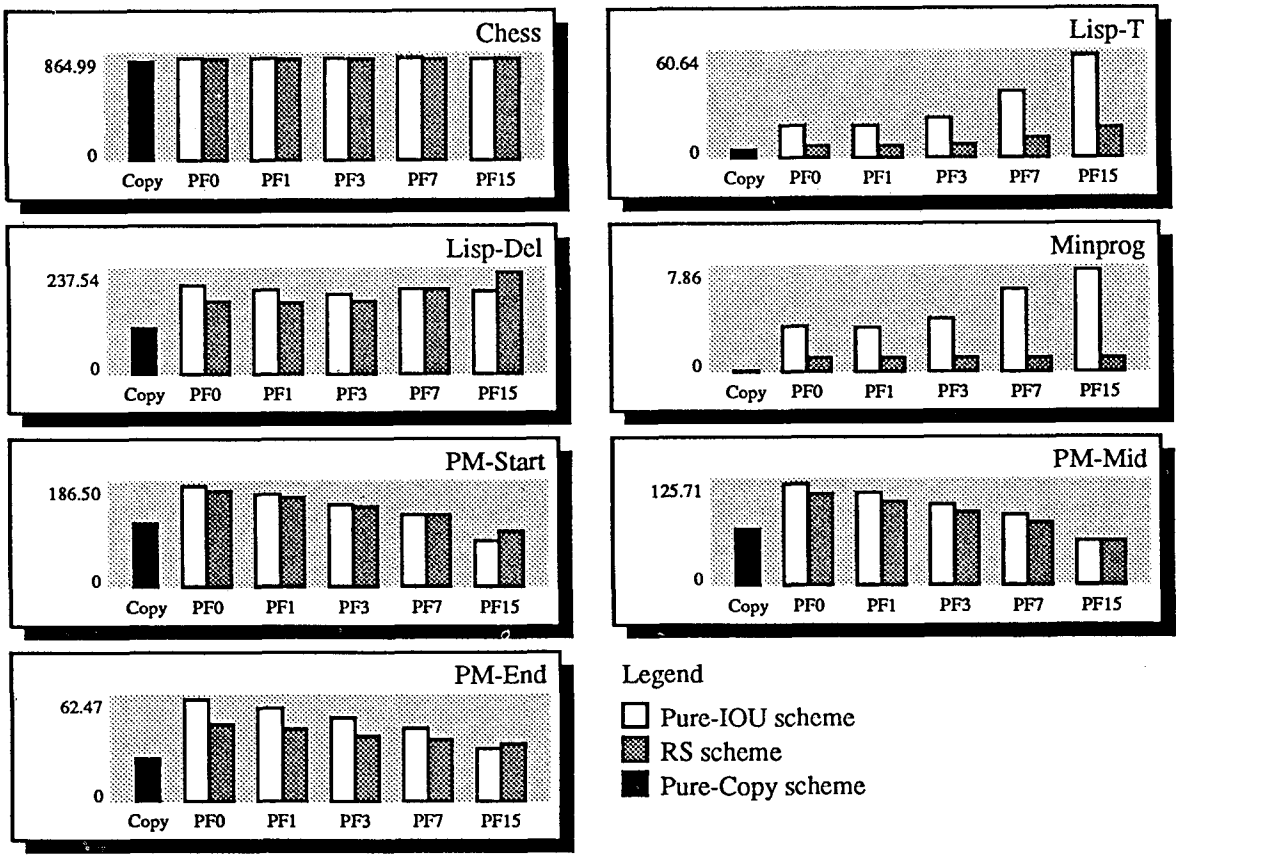

Figure 4-1: Remote Execution Times in Seconds

The measurement interval starts when the relocated program is restarted at its new location and ends when remote execution completes. Column PF*n* describes a trial where *n* pages were prefetched in response to an imaginary fault. Note: each chart is scaled individually.

mission times vary by a factor of 20. Pure-IOU allows the address space transmission to complete in significantly less time. Lisp-Del is the most extreme example, where a physical copy is almost 1,000 times more expensive. Resident set transfers once again display intermediate performance.

4.3.3. Remote Execution Times

Figure 4-1 shows the remote execution times of the representative processes, namely the elapsed time in seconds from the first instruction executed at the new host up to the program's termination. These figures show the effects of the different migration strategies, combined with differing prefetch values for the pure-IOU and resident set approaches.

Part of the effort saved in the lazy transfer of an address space must be expended as the process accesses its memory remotely. Referencing imaginary memory through the intermediary *Scheduler* and *NetMsgServer* processes on both testbed machines is roughly 2.8 times more expensive than accessing data backed by a local disk (115 milliseconds vs. 40.8 milliseconds). The most glaring effect of this cost differential on remote execution time is seen in the Minprog case, which executes 44 times slower under the pure-IOU strategy. The majority of this time is spent collecting its working set as it attempts to execute the few instructions before it terminates. The long-lived, compute-bound Chess program suffers a much smaller execution penalty, running only about 3% longer.

Dependent on the memory access patterns exhibited, the effect of prefetch varies considerably among the representatives. The Lisp family, which doesn't display memory locality, suffered from increased prefetch. The additional pages were rarely used and did not justify the larger fault-handling time. Hit ratios on these extra Lisp pages dropped from around 40% to 20% as prefetch increased. On the other hand, programs such as Pasmac, which access large tracts of memory in a sequential fashion, benefitted greatly from large prefetch. Pasmac tallied a steady 78% hit ratio across all prefetch values used, and improved its IOU remote execution times by up to a factor of 2 across this range.

Transferring process resident sets to the new execution site only had a significant impact on the extremely short-lived processes (Lisp-T, Minprog). This implies that the underlying working sets change quickly for Accent processes, in turn suggesting that resident set transfers are not a useful optimization in this setting.

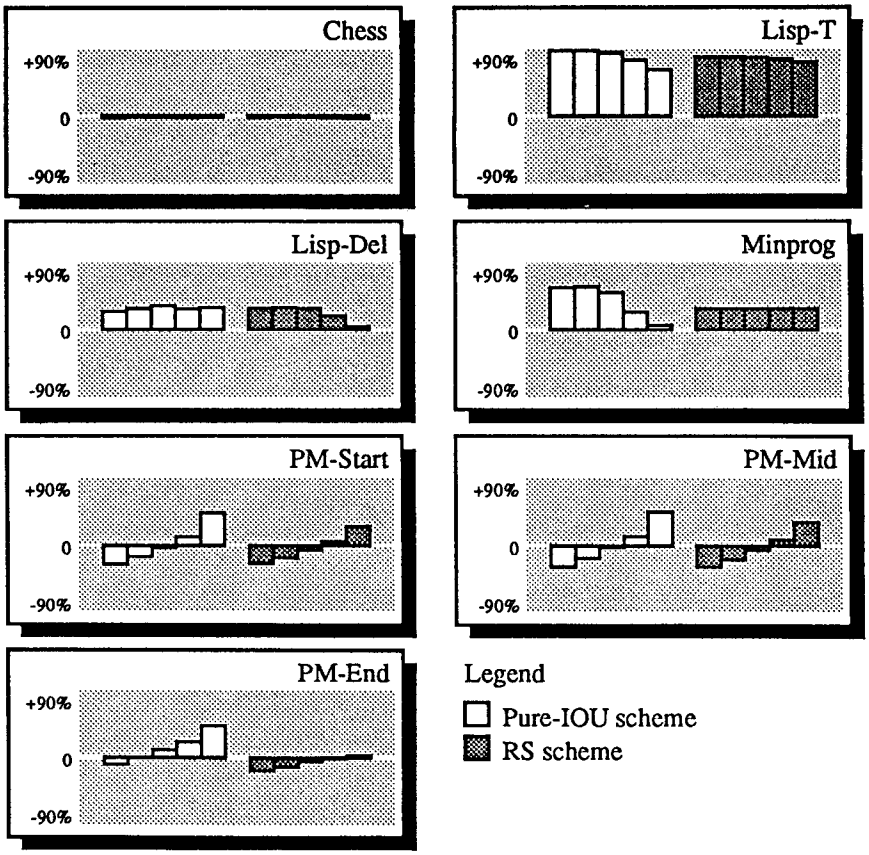

Figure 4-2: Percent Overall Speedup over Pure-Copy

Shown are the end-to-end speedups resulting from use of pure-IOU and RS transmissions. Elapsed times for address space transfer and remote execution are summed for each representative process and prefetch value and compared to the pure-copy results. From left to right in each group, bars indicate percent speedup over pure-copy for prefetch values of 0, 1, 3, 7 and 15 pages. Negative values (bars drawn in the bottom half of each gray area) represent slowdowns in relation to pure-copy.

4.3.4. Overall Migration Speedup

As demonstrated above, the pure-IOU and RS schemes hold a clear advantage in the address space transfer phase of migration yet generally cause processes to execute longer at the remote site. In order to get overall or end-to-end performance figures, elapsed times for context transfer and remote execution are summed for these strategies and compared to the pure-copy results. The percent speedups over the straightforward pure-copy technique are displayed in Figure 4-2 for the pure-IOU and RS approaches as different amounts of prefetch are performed. The pure-IOU results (white bars) are grouped together on the left-hand side of each chart; similarly, the resident set results (dark gray bars) are placed on the right-hand side. From left to right in each group, the bars show the percent speedup for prefetch values of 0, 1, 3, 7 and 15. Negative values indicate slowdowns in relation to pure-copy.

As expected, processes that access the smallest portion of their address spaces at the new site are best suited to use the copy-on-reference technique when overall elapsed time for migration and remote execution is the metric. In the current implementation, the breakeven point is around one-quarter of the process *RealMem*. Once past this percentage, as in the Pasmac family of processes, the higher cost of fetching individual pages during remote execution in the pure-IOU system outweighs the savings achieved during migration itself. The exception to this observation is the Chess program, which is insensitive to the transfer method used. In that case, the differences imposed the various strategies were drowned out by the program's longevity.

With its strong influence on remote execution times, the amount of prefetch performed is a critical factor in end-to-end performance. Pasmac, as a representative for processes past the breakeven point and demonstrating strong sequential access patterns, went from an overall 21% slowdown on average to a 44% *speedup* as prefetch increased. In all cases, the results demonstrate that returning one additional contiguous page per remote fault improves performance. With intelligent use of prefetch, copy-on-reference migration is significantly faster than pure-copy transfer for the representatives (except the long-lived Chess process) when overall timings are considered. On the other hand, process resident sets didn't "pay their way" by cutting remote faulting activity enough to offset their shipment costs.

4.4. Cost Analysis

Section 4.3 reports that copy-on-reference treatment of ad-

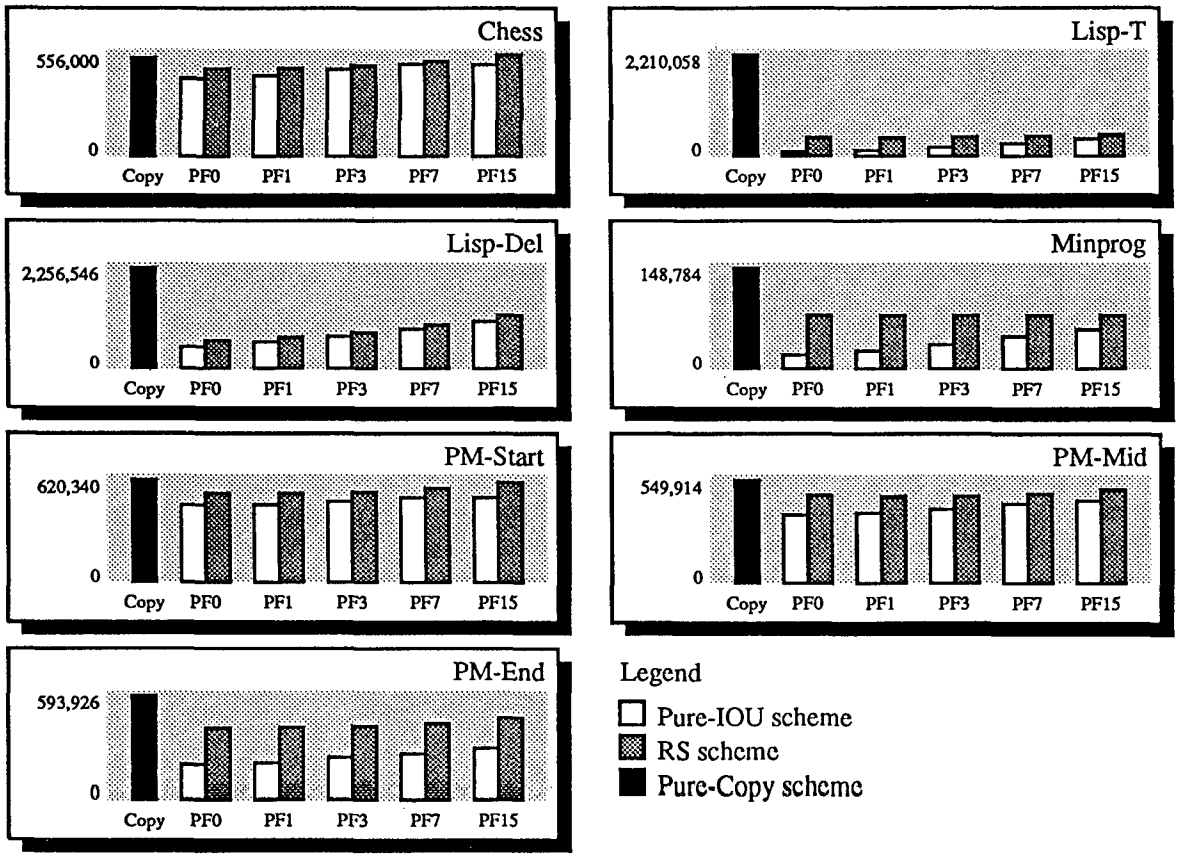

Figure 4-3: Bytes Transferred During Trials

Number of bytes transferred for each program, transmission strategy and page prefetch value during the migration trials. The measurement interval starts when the migration request is received by the *MigrationManager* and ends when the program completes its remote execution. Column PF*n* describes a trial where *n* pages were prefetched in response to an imaginary fault. Note: each chart is scaled individually.

dress space transfers significantly improves the time required to migrate a process to a new site and complete its execution there. This section supports these results by examining the specific costs incurred by the different migration strategies, and how these costs are distributed across the migration phases. Experiments reveal that copy-on-reference reduces the number of bytes transferred between the hosts as well as the cost of handling messages related to migration activities. Not only are the overall costs lowered by this approach, but they are also more evenly distributed across the context transfer and remote execution phases.

4.4.1. Bytes Transferred

Figure 4-3 reports the number of bytes exchanged between machines due to migration and remote execution of the representatives under the different strategies. Note that a single value is reported for each pure-copy trial, since prefetch doesn't apply in these cases.

The pure copy-on-reference strategy was superior to pure-copy across all prefetch settings. This technique reduced byte traffic by an average of 58.2% over pure-copy when no prefetch was used. As a rule, more data was exchanged as the number of contiguous pages prefetched grew. This is reasonable, since not all the extra pages were referenced. Shipping resident sets cut into the savings realized by the IOU strategy, again implying that very little of this data was actually used at the remote site.

4.4.2. Message Costs

Pure-copy is the clear winner when evaluated by the *number* of messages processed by the test systems. However, it does not fare nearly as well in a more important metric, the amount of time required to process and deliver these messages. Each second of execution time spent by the *NetMsgServer* to handle message traffic is not only a second stolen from the migrated process but from *all* processes in *both* systems. Figure 4-4 displays the amount of time spent by each node in message manipulation.

These figures further confirm the utility of a lazy approach to address space access. By putting off the apparent work that needs to be performed until the last moment, a significant portion does not need to be done at all. Although the bulk transfer of the process context when the pure-copy strategy is employed allows a higher throughput than the page-by-page access imposed by the pure-IOU and resident set approaches, the majority of pages sent by the pure-copy approach are never used. The pure-IOU strategy only performs work that is productive and necessary.

In every case, the IOU and resident set strategies outperform pure-copy. The average savings in message processing is

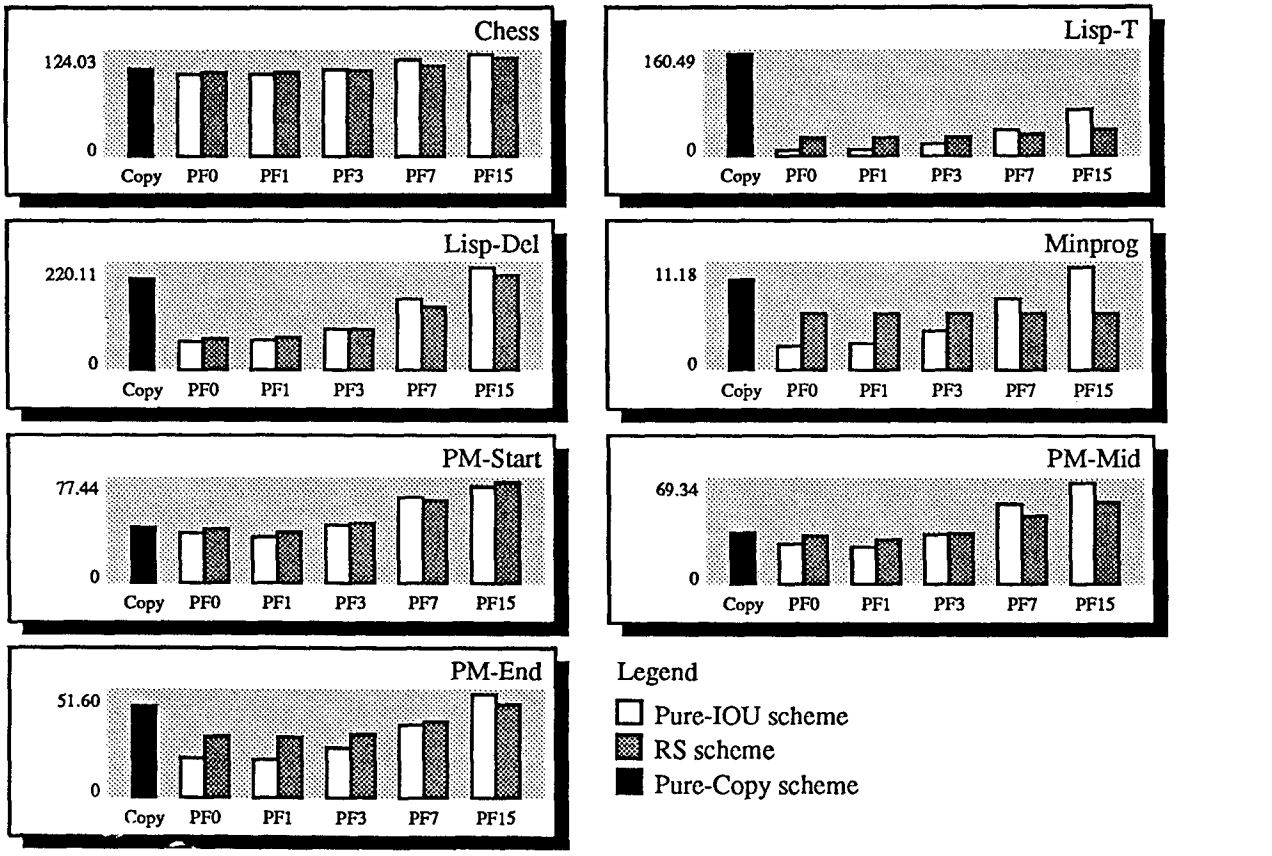

Figure 4-4: Total Message Times in Seconds

Displayed are the elapsed time in seconds required to process the IPC messages generated for each migration trial. Column PF*n* describes a trial where *n* pages were prefetched in response to an imaginary fault. Note: each chart is scaled individually.

47.8% for IOU trials without prefetch. The effect of prefetch is an interesting one. When only a single additional page is prefetched in response to an imaginary fault, the time spent processing messages drops slightly. As we increase the number of pages prefetched, the system spends more and more time in message handling. Although the prefetching eliminates many of the imaginary faults, it also transfers some "dead weight" pages that are never used. Also, since each message carries more data, the time to process each imaginary reply message grows.

Combined with the results on end-to-end costs, these figures suggest that one page should be prefetched regardless of the transfer strategy chosen.

4.4.3. Distribution of Costs

The vast majority of migration costs charged to the pure-copy strategy are incurred during the transfer phase of process migration. On the other hand, the copy-on-reference approaches radically reduce the cost of context shipment and instead incurs its expenses across the remote lifetime of the process involved. Thus, not only are costs reduced overall, but they are also more evenly distributed. Pure IOU transfers don't experience the same magnitudes and bursts of activity required by the pure-copy strategy. Instead, a lower, more constant rate of work is exhibited. The trials demonstrate that sustained network transmission speeds are reduced up to 66%.

Figure 4-5 presents the data transfer rates caused by the migration and remote execution of the Lisp-Del case under the different strategies, starting at the time of migration and ending with the execution of the final remote instruction.

These panels depict the results of a full-IOU transfer of Lisp-Del, a resident set approach and finally the full-copy method. The areas in white represent bytes exchanged in support of imaginary fault activity. Full-copy transfers have a characteristic signature, with a large bulk data transfer early on. The resident set panel illustrates that a sizable amount of data is still physically shipped during the migration phase, but does not improve the overall time significantly from the pure-

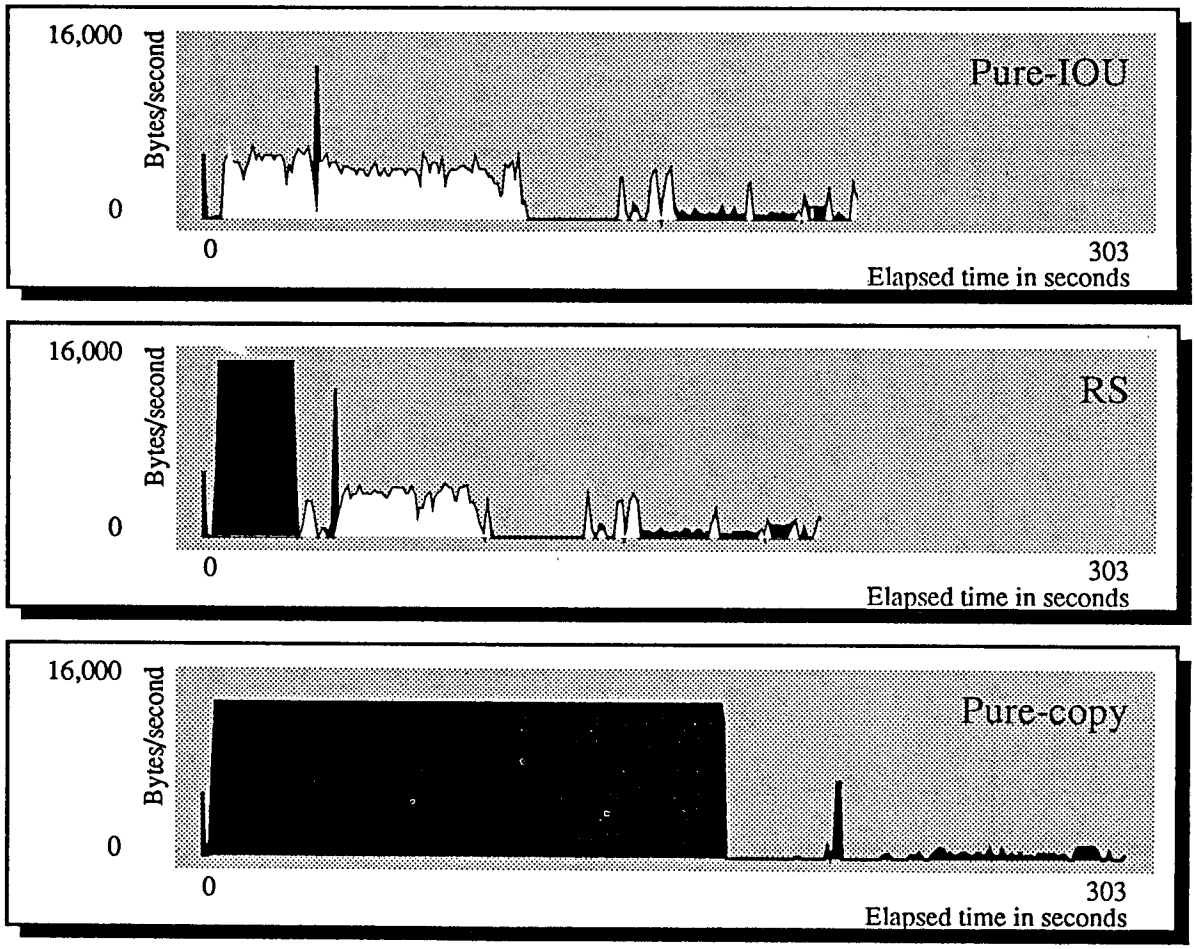

Figure 4-5: Byte Transfer Rates for Lisp-Del

Network data transfer rates during the migration and remote execution of the Lisp-Del (Delaunay triangulation) program. No prefetch is being performed. White areas show data transferred in support of imaginary faults, black areas show all other transfers.

IOU approach. Copy-on-reference allows the process to resume execution very quickly. In this case, Lisp-Del finishes its work shortly after the full-copy trial *begins* its remote execution.

4.5. Summary

The trial data collected for the Accent migration facility reveals several interesting facts about process composition and behavior. While address space size varies by as much as a factor of 12,803 in the representative processes, the amount of *RealMem* only differs by a factor of 15. *RealZeroMem* forms a significant portion of every process address space, more than half in most cases and 99.9% in the Lisp examples. These representatives touched between 0.002% and 27.4% of their validated address spaces, and between 3% and 58% of the *RealMem* portions. This verifies the assumption that processes access relatively small parts of their addressible data.

Process excision and insertion times are also much less variable in this study, factors of 4 and 3.3 respectively across the samples. IOU context transfers take roughly one second in all cases, and thus provide a lower bound for this activity. Pure-copy transfers vary by a factor of 20, and in the most extreme case are 1,000 times more expensive than the corresponding pure-IOU transfer.

Much less data needs to be communicated between machines when copy-on-reference tactics are used. On average, 58% fewer bytes are transferred and message processing times drop by 47%. Touching remote pages via the copy-on-reference mechanism is roughly 2.8 times more expensive than local disk accesses, and this figure can likely be improved through tuning.

The copy-on-reference variations studied in this system produced mixed results. Resident sets were found to be poor predictors of the data required by the process at its remote site. Since Accent uses its physical memory as a disk cache, many resident pages are sometimes guaranteed *not* to be referenced again, especially by the Pasmac class of processes. On the other hand, small amounts of page prefetch were found to always be useful. Prefetching more pages each time degrades performance in some cases, but greatly aids programs performing mostly sequential accesses.

5. Related Work

Investigation into process migration began in the early 1970's. Such efforts as the "Creeper" program [11] by Bob Thomas at BBN and the "Relocatable McRoss" [14] air traffic controller demonstrated migration's feasibility. However, they did little to address the transparency issues. DCN [6] added name transparency by associating resources with processes, but failed to provide location transparency. DCN's resource names specified the supplying host, and were invalidated if the resource was moved. The RIG system [4] is Accent's direct ancestor and shared many of the same concepts. RIG's ports were visibly tied with the process owning them, so it suffered from DCN's problem. The DEMOS/MP operating system [9] was among the first to offer full transparency. Link names contained *hints* to the location of the service, and were not invalidated by resource relocation. The University of Washington's object-oriented Eden [5] system provided full transparency and migration services, but could not take advantage of a copy-on-reference mechanism. Eden's objects were forced to reside entirely on a single host. Dannenberg's Butler [1] made use of an older version of Accent which did not provide copy-on-reference data shipment, but demonstrated Accent's suitability for transparent migration support.

Various systems have attempted different attacks on the cost of context transfer. The LOCUS [8] remote invocation facility exploits shared code present at the target site, cutting down the amount of data that must flow to the new site. This approach doesn't address the data portions of a process context, including memory-mapped files. Marvin Theimer's migration facility for the V system [13] tried to hide transmission costs from processes by *pre-copying* the context in an iterative fashion before moving the process. Process downtime was thus reduced, but both hosts still paid the transfer costs. Theimer's measurements reveal that this technique suffers from network buffering problems and overruns.

6. Conclusions

The Accent testbed's use of copy-on-reference address space transfer has demonstrated its effectiveness in tackling process migration's dominant cost. Unlike the conventional transmission technique, copy-on-reference avoids the linear growth in costs as processes address more and more data. Any distributed system in the same class can expect similar results in the construction and use of a copy-on-reference facility.

Studying the Accent example also teaches important lessons in operating system design. The simple yet powerful port abstraction and the close integration of IPC and virtual memory facilities give Accent the transparency needed to cleanly support migration without sacrificing performance. These features, along with extensibility through user-level processes, allows a generic copy-on-reference mechanism to be built in a natural way. This mode of data transfer has proven useful in the migration domain, but may be just as easily applied to *any* task requiring sparse access to large tracts of memory.

Copy-on-reference data transmission is inherently more flexible than the conventional method. Only two variations of actual data delivery have been explored here. Tasks with special knowledge of the data requirements they will encounter may apply that knowledge to optimize the physical shipment of data.

This investigation opens many avenues for future research. The creation and evaluation of automatic migration strategies appropriate for such systems have not been addressed here. Good strategies are necessary to capitalize on the inherent advantages of lazy transfers. Part of this activity will involve the development of good load metrics which specifically take into account the fact that a process virtual address space may be physically dispersed among several computational hosts. Copy-on-reference may be proven useful in remote file and database accesses, remote invocation facilities and intelligent RPCs. It would be interesting to attempt to extend this work

to systems allowing shared memory, and to evaluate the application of copy-on-reference techniques to a shared centralized file system such as Andrew [7].

Although Accent is no longer actively in use at Carnegie Mellon University, the lessons learned from this work are being applied to the Mach environment [10] currently being developed there. A successor to Accent aimed at supporting a wide range of hardware configurations, Mach allows *external pager* processes which provide copy-on-reference administration of data. Study of copy-on-reference behavior in this new facility will provide further insights on the basic mechanism in a more modern computing system.

References

1. Roger B. Dannenberg. *Resource Sharing in a Network of Personal Computers*. Ph.D. Th., Carnegie Mellon University, December 1982.

2. Peter J. Denning. "The Working Set Model for Program Behavior". *Communications of the ACM 11*, 5 (May 1968), 323-333.

3. Robert P. Fitzgerald. *A Performance Evaluation of the Integration of Virtual Memory Management and Inter-Process Communication in Accent*. Ph.D. Th., Carnegie Mellon University, October 1986. Available as CMU technical report CMU-CS-86-158.

4. Keith A. Lantz, Klaus D. Gradischnig, Jerome A. Feldman and Richard F. Rashid. "Rochester's Intelligent Gateway". *Computer* (October 1982), 54-68.

5. E.D. Lazowska, H.M. Levy, G.T. Almes, M.J. Fischer, R.J. Fowler, S.C. Vestal. The Architecture of the Eden System. Tech. Rept. 81-04-01, Department of Computer Science, University of Washington, April, 1981.

6. David. L. Mills. An Overview of the Distributed Computer Network. National Computer Conference, University of Maryland, 1976, pp. 523-531.

7. James H. Morris, Mahadev Satyanarayanan, Michael E. Conner, John H. Howard, David S. H. Rosenthal and Donelson Smith. "Andrew: A Distributed Personal Computing Environment". *Communications of the ACM 19*, 3 (March 1986), 184-201.

8. G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, G. Thiel. LOCUS: A Network Transparent, High Reliability Distributed System. Joint Conference on Computer Performance Modelling, Measurement and Evaluation, ACM, 1986.

9. Michael L. Powell and Barton P. Miller. Process Migration in DEMOS/MP. Proceedings of the Sixth Symposium of Operating System Principles, ACM, November, 1983, pp. 110-119.

10. Richard F. Rashid. "Threads of a New System". *Unix Review 4*, 8 (August 1986), 37-49.

11. John F. Shoch and Jon A. Hupp. "The 'Worm' Programs - Early Experience with a Distributed Computation". *Communications of the ACM 25*, 3 (March 1982), 172-180.

12. CMU Computer Science Department. Proposal for a Joint Effort in Personal Scientific Computing. Carnegie Mellon University, August, 1979.

13. Marvin M. Theimer, Keith A. Lantz and David R. Cheriton. Preemptable Remote Execution Facilities for the V-System. Proceedings of the Tenth Symposium on Operating System Principles, ACM SIGOPS, 1985, pp. 2-12.

14. Robert H. Thomas and D. Austin Henderson. McRoss - A Multi-Computer Programming System. Proceedings, Spring Joint Conference, 1972.