Operating System Support for Concurrent Remote Task Creation

Dejan S. Milojičić, David L. Black and Steve Sears

Open Software Foundation Research Institute[1]

Abstract

This paper describes improvements to the Mach microkernel's support for efficient application startup across multiple nodes in a cluster or massively parallel processor. Significant improvements in application startup times have been achieved by optimizing the existing remote task creation operation, implementing a facility to concurrently create multiple remote tasks in a single operation, and restructuring the underlying distributed virtual memory system to improve its scalability. One component of the restructuring involves the use of a hierarchical tree of objects to implement the paging path instead of a flat single level tree; this eliminates bottlenecks at the node that initiates the application. The other component consists of limiting the copy on write virtual memory optimization to single node operations; this achieves a separation of network sharing (read/write) from network read access (implemented by copy on reference). Although our implementation is specific to Mach, the architecture and design are applicable to other modern operating systems.

1 Introduction

The lack of adequate operating system support for application startup can be a significant barrier to the use of modern parallel hardware architectures such as Massively Parallel Processors (MPP) and clusters of powerful workstations. Sequential startup of the components of a parallel application and serial bottlenecks in the underlying operating system structures can seriously impact overall application execution speed. This paper describes our work to address this problem by implementing concurrent remote task creation and related virtual memory changes for the Mach 3.0 microkernel [1]. This work was performed on a version of Mach that supports MPPs and clusters [13], but the techniques are applicable to other operating systems for these architectures.

1. This work was supported in part by the Advanced Research Projects Agency and the Rome Laboratory of the Air Force Materiel Command.

To be presented at the 9th International Parallel Processing Symposium, Santa Barbara, California, April 24-29, 1995.

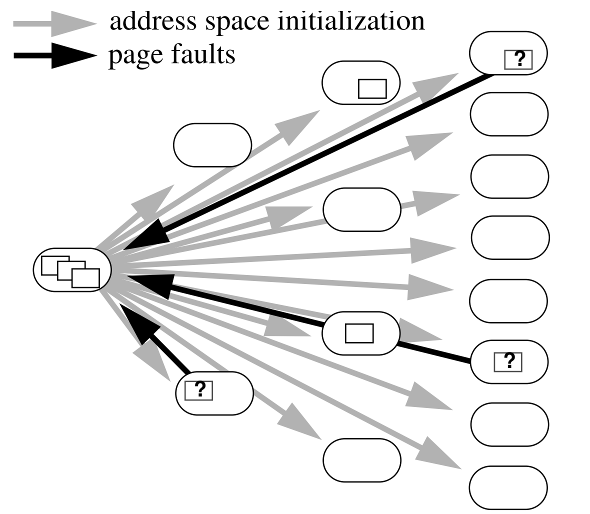

Figure 1: Centralized Distributed Memory Management: the source node is a bottleneck for page-in/out.

The current single remote task creation mechanism in the distributed version of Mach lacks performance and scalability. Information describing the task is transferred sequentially in a series of network IPC messages. The combination of this with the high latency of Mach's current network IPC implementation makes this state transfer a costly operation. Another problem is that the Distributed Shared Memory (DSM) implementation employs a centralized paging architecture that introduces a bottleneck on the originating node for both task creation and paging of the resulting tasks (See Figure 1). Finally, the distributed Virtual Memory (VM) design distributes a local VM optimization, asymmetric Copy On Write (COW), across node boundaries. The resulting design and implementation are unnecessarily complex and cause performance penalties. This paper describes our work to overcome these problems via design changes and optimizations to all of the areas involved.

The rest of the paper is organized as follows. Section 2 provides background on the Mach VM system. Section 3 describes optimizations to the creation of a single remote task. Section 4 describes the new hierarchical paging architecture. Section 5 presents our design for separating between distributed sharing and copy on reference. Section 6 describes the

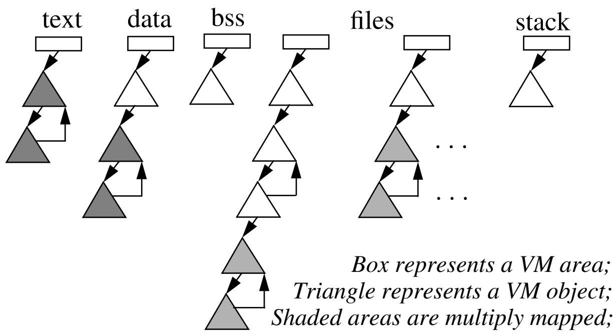

Figure 2: Mach Task Address Space, a UNIX Perspective: a VM area consists of one or more VM objects linked in a shadow chain that is traversed on a page fault to find the required page.

current state of the implementation and provides performance measurements. Section 7 compares our work to related projects. Conclusions are presented in Section 8.

2 Mach VM background

The address space of a Mach task consists of a collection of VM regions with individual attributes, such as protection and inheritance [9]. Each region is mapped to an associated VM object that represents the source of the memory (e.g., file, pager) and a kernel cache for memory pages obtained from that source. Copy on write operations can cause this VM object to be chained to other VM objects to implement the copy on write functionality. See Figure 2 for an example of a Mach address space and its associated VM objects.

Mach VM optimizes copying of large memory areas by employing a virtual copy technique instead of physically copying memory. The virtual copy operation write protects the page and allows access via both the source and copy mappings. A write fault causes the page to be copied into a new VM (shadow) object (chained to the original object) so that it is not visible to the other mapping; this technique is known as copy on write. The actual implementation of this copy depends on the copy strategy type, *symmetric* or *asymmetric* [11].

Copy on write initially results in a VM object being mapped into both the source and copy addresses. As long as the pages backed by the object have not been modified, multiple tasks can reference object. If a page is modified, behavior depends on the object copy strategy type. In the symmetric case, the original object is preserved and new objects are created for both mappings in response to writes, as shown in Figure 3. Symmetric copy on write cannot be used for external objects such as mapped files because changing the VM object would disconnect the mapping from the file. The asymmetric copy strategy for such objects retains the original object in the source and creates a

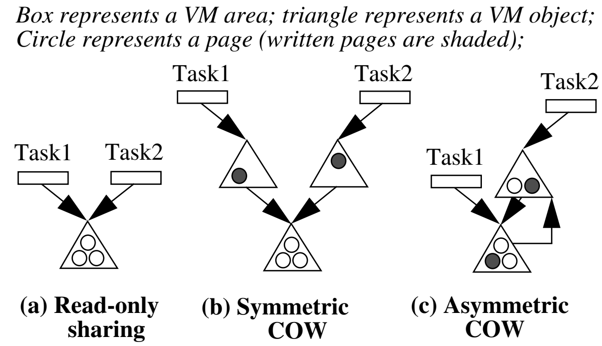

Figure 3: COW Optimizations in Mach VM: when written, in symmetric case parent and child get new copies, in asymmetric, parent retains original page and pushes the copy into child object.

new (copy) object for the copy mapping. A write fault on the source mapping causes an unmodified copy of the page to be pushed into the copy object before allowing write access to the page in the original object. In both the symmetric and asymmetric cases, a not-present fault causes the object chain to be searched from the top to find the page that satisfies the fault.

Task creation entails making virtual copies of much of the source task's memory and hence causes growth of these object chains. When a parent task sequentially creates multiple child tasks, the parent will typically write to its memory (e.g., stack) between operations, requiring shadow chain growth for each task creation. An important difference between the symmetric and asymmetric copy strategies is that in the symmetric case, the object chain always grows from the top (mappings) while in the asymmetric case, additional copy objects will be created at the bottom of the chain between the bottom object in the chain and the first object above it.

Object chain growth causes performance problems in a distributed system that are not present on a single node. Mach VM limits the potential performance problem on single node systems by collapsing shadow chains when intermediate objects are no longer needed. In the distributed case, where the chain may have been copied to Mach kernels on multiple nodes, this code is not effective; hence object chain growth can cause performance problems both in the length of the chain traversal required to handle a page fault and in the amount of state that must be transferred to create a new task. The creation of an additional object at the bottom of a shadow chain required by asymmetric copy on write (noted in the previous paragraph) is a particularly expensive operation in the distributed case.

The distributed version of Mach also includes an implementation of distributed shared memory that takes advantage of Mach's VM functionality [4]. Consistency is maintained on a page granularity using a conventional single-writer/

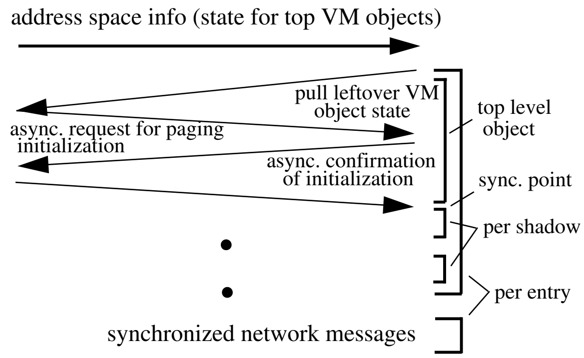

Figure 4: State Transfer for original remote task creation: sending a part of the task address space state, followed by pulling leftover state and synchronized paging path initialization**.**

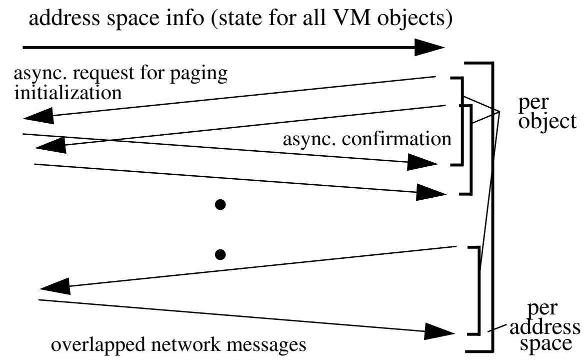

Figure 5: State Transfer for Optimized Remote Task Creation: all available state is initially transferred (the whole shadow chain), followed by overlapping object initialization.

multiple-readers paradigm. This functionality is layered within Mach's external memory management interface so that pagers need not implement distributed shared memory.

3 Improving single remote task creation

Our work focused on optimizing the creation of remote address space because this operation consumes most of the time required to initialize a remote task. There are two steps involved in remote address space creation. The first step is to send information to the remote node that describes the address space areas to be created there. This information includes size, permission, inheritance, a pager capability for the VM object, and the like. At the remote node the second step consists of creating a new VM map and then initializing each VM object. This includes discovering and initializing every object in the shadow chain for each area in the new VM space as shown in Figure 4.

This design of the existing remote address space creation has three major performance problems. The first is that the state transfer in the initial message to set up the address space does not contain the internal VM object state necessary to initialize it, requiring a message exchange with the originating node in order to retrieve this information. The second is that the lazy evaluation of the shadow chain objects requires two message exchanges for each object in the chain (one to discover the object and one to initialize it). The third problem is that all of these message exchanges are serialized in the existing design. Our work corrects all three of these problems.

Eager copying of all object state is used instead of relying on subsequent retrieval by the remote kernel. There are some rare cases that require retrieving this information from a third node; the algorithm falls back to the previous design in these cases. This reduces the number of message exchanges involved in remote address space creation almost in half and was the single largest performance improvement for single remote task creation.

Eager copying of shadow chains is implemented instead of expanding one object at a time. If a part of the shadow chain occurs more than once in an address space, it is transferred only once. Typical cases are *text* and *data*, or multiply mapped files, shown as shaded areas in Figure 2**.** Although our experiments show that eager copying of shadow chains improves performance, lazy copying in the case of very long shadow chains is a possible optimization in the case that portions of the chain are not used (or used only a small number of times). Our preferred approach to this area is to investigate the cause of the long shadow chains and shorten them, but lazy copying is a possible future optimization.

Elimination of synchronization points in object initialization allows overlapping message exchanges. Paging path setup requires additional communication between the client and the pager; the structure of the paging system makes it infeasible to optimize this communication via batching (absent a total rewrite). Instead, the serialization of these message exchanges has been eliminated by transferring additional VM object state that was causing the synchronization and employing a copy on reference strategy for objects in a shadow chain (See Section 5 for details). The resulting overlapped message exchanges for remote address space creation are shown in Figure 5.

4 Hierarchical paging and task creation

A bottleneck in the paging path structure imposed by the current DSM implementation [4] causes performance problems for creation of multiple remote tasks from a single parent (source). The created tasks share (via copy on write) VM objects that are in the parent task's object chains. The centralized DSM implementation imposes a single management node for each object through which all paging operations (pagein in particular) must flow. This node becomes a bottleneck when large numbers of tasks are created from a single parent. An obvious method to attack this bottleneck is

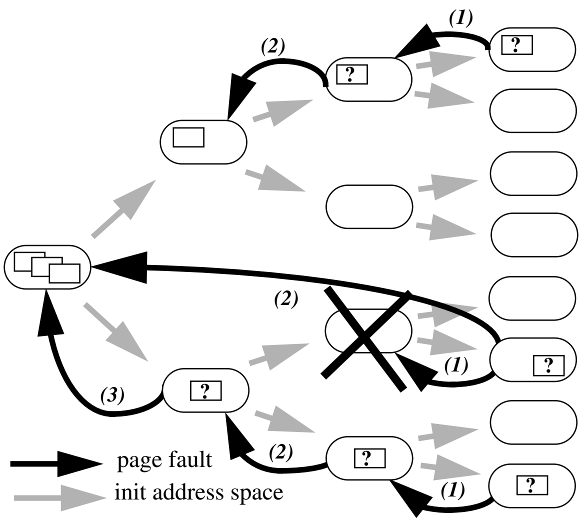

Figure 6: Hierarchical Distributed Memory Management: the paging path is organized in the form of a spanning tree. On the page fault, the tree is traversed up to the root and possibly even to the pager. The pages are left on the intermediate nodes.

to use a tree structure for multiple task creation; the parent creates two children who each create two children, etc. This fails to achieve the desired results because an optimization in the centralized DSM system bypasses the intermediate nodes and directs pagein operations to the original parent; decentralization of distributed memory management is the only way to eliminate this problem.

This bottleneck is alleviated by using a tree structure for the paging path, as shown in Figure 6. The tree is created by recursive expansion from the root node where the original VM object is located as corresponding objects on other nodes are initialized. This hierarchical structure directs pageins to intermediate nodes in the tree where they may be satisfied or combined with identical pagein requests from other nodes. In both cases, the paging request load visible to nodes further up the tree is reduced. Each node in the tree also retains a pointer to the root of the tree; a node uses this pointer to reconnect to the tree should one of the intermediate nodes drop out of the tree due to failure. Our current implementation uses binary trees, but can easily be extended to use broader trees.

The hierarchical remote address space creation algorithm is optimized to provide maximum parallelism. Once a vector representing the original address space is received by an intermediate node, its contents are used to instantiate the new address space. This vector is copied to subsequent nodes with the state relative to the local node substituted appropriately. For example, pagers in the shadow chain, size and protection of VM regions are simply copied, while the capabilities for DSM objects are substituted with newly created ones. It is important to carefully combine local initialization and forwarding of the task state. For example, it is not necessary to complete address space creation on an intermediate node before the state is sent onward. Only the minimum necessary initialization is performed, allowing the balance to be performed in parallel with other nodes. Once an address space is created, it is also possible to eagerly push the pages, e.g. parts of the stack and text, although our current implementation does not do so.

This hierarchical approach to creating multiple tasks in a single operation is superior to an application based tree approach (e.g., parent creates two children who each create two children, etc.). The concurrency and scalability of the hierarchical approach are superior because it is not necessary to fully create and start a task at an intermediate node in the tree before beginning to create its child tasks. In contrast, the application based approach must fully create the task and start two threads before it can create its child task.

5 Copy on reference versus sharing

Significant reductions in complexity and corresponding performance improvements can be achieved by substantially changing Mach's approach to virtual copy operations in a distributed system. Experience has shown that the current approach of distributing copy on write operations causes problems with respect to correctness, performance and scalability. Distributed **shared** memory in Mach is overly used. Even the objects that are inherently distributed **read-only** (via virtual copy) are considered potentially read/write shared, and managed as such. Consequently, the distributed read-only case is encumbered with excessive complexity and unnecessary state information to support read/write sharing.

In the distributed version of Mach, asymmetric COW objects can appear in the middle of an object shadow chain when a symmetrically copied object becomes shared (e.g. via changing its inheritance to shared before creating a new task). If this object is then used on multiple nodes, the page push operation that implements copy on write potentially has to be executed on all of the nodes involved. This is both inefficient and complex.

We addressed these two deficiencies by making two major modifications:

- Distributed read/write shared objects are separated from distributed read-only objects, by introducing a new class of Copy On Reference (COR) objects.

- The asymmetric COW technique is not used for an object in the middle of an object shadow chain. In situations that would cause this, the contents of the object involved are eagerly copied to a new object and the new object (which has no shadow chain) is shared instead. The performance

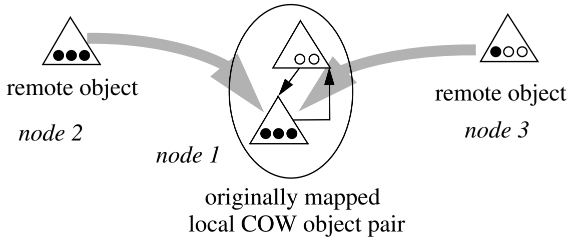

Figure 7: Mapping Asymmetric COW Objects over Network: the COW object pair on remote node observe the original COW pair through one mapping.

impact of this eager copying is minor because sharing of memory that has been virtually copied is a relatively rare case (e.g., Unix memory mapping interfaces do not cause such sharing).

These modifications combine the benefits of the copy on reference technique with better reuse of existing data on the remote nodes, namely portions of the object shadow chains. As part of this modification we avoid distributing a VM object with an asymmetric copy strategy when performing remote task creation. Instead we create a new COR object on the originating node as a copy of the original object and distribute the COR object so that copy on reference is used between nodes rather than copy on write as shown in Figure 7. If the underlying asymmetric object has already been distributed, we retain the object chain links to avoid remote operations to obtain pages present on the node in that object.

Separating COR from the sharing path takes advantage of the common read-only case, separating support for the less common sharing case. This leads to a simplification of code and data structures for the common case as well as improvements in performance because it is no longer necessary to maintain and update read/write sharing state. A surprising discovery was that the introduction of COR objects also simplified the code involved in read/write sharing by eliminating the need for distributed copy on write operations. This also improves the performance of distributed data access in the COR case because these distributed copy on write operations can be quite expensive (especially the operation of inserting a new object at the bottom of a shadow chain for asymmetric copy on write).

6 Current status and evaluation

Implementation of the work described in this paper was performed on the OSF nmk17 version of Mach using an OSF/1 AD 1.0 server on a 64-node Intel Paragon. An ethernet-linked cluster of PCs was used as the development platform, but all results are reported from the Paragon. Results are reported as a mean of 3 measurements. All of our

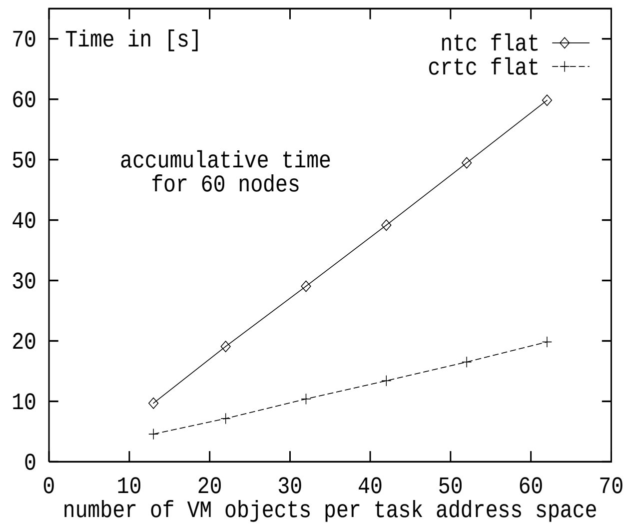

Figure 8: Remote task creation as a function of the number of VM objects: consecutive remote task creation on 60 nodes for original and improved version.

measurements are reproducible; the standard deviations are essentially not visible on the graphs in this paper and are hence not shown.

Eagerly copying all VM object state improves performance of remote task creation by approximately 30%. Eager copying of an additional object in a shadow chain (the unmodified kernel eagerly copies one shadow) yields another 14% improvement; the exact figure depends on the length of the shadow chain. Additional improvement is due to the elimination of synchronization points. The total improvement can be seen in Figure 8, where remote task creation performance is shown as a function of the number of VM objects that must be transferred. In this and subsequent figures, *ntc* refers to the original (norma) task creation, and *crtc* refers to our concurrent remote task creation implementation.

The number of VM objects increases with both the number of areas in the address space and the depth of the shadow chains. Large numbers of VM areas can arise for applications with sparsely populated address space. Deep shadow chains are a consequence of many successive copy on write optimizations, e.g. in a child with many ancestors. A typical Mach task requires transfer of 10-20 objects, but we have observed more complex tasks (e.g., an operating system server), with hundreds of VM objects.

Figure 9 compares performance of both the original and improved remote task creation (flat and hierarchical cases for the latter) as a function of the number of remotely created tasks. Although we have achieved significant performance improvements over the original case, we did not meet our expectations of logarithmic scaling by the number

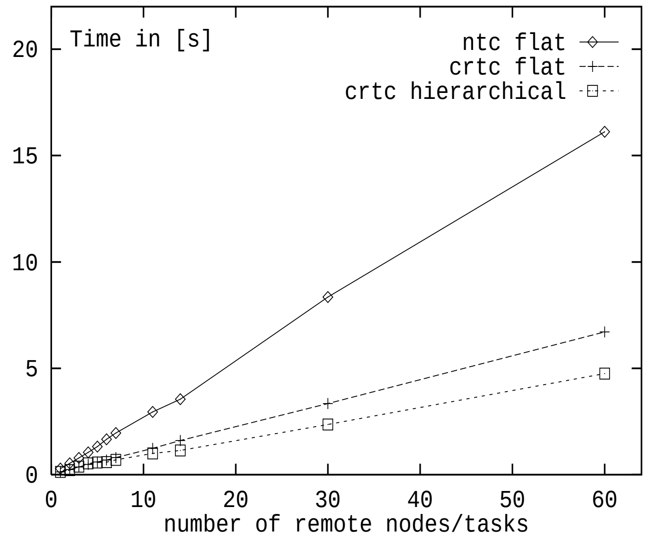

Figure 9: Hierarchical v. centralized remote task creation: as a function of the number of task remotely created.

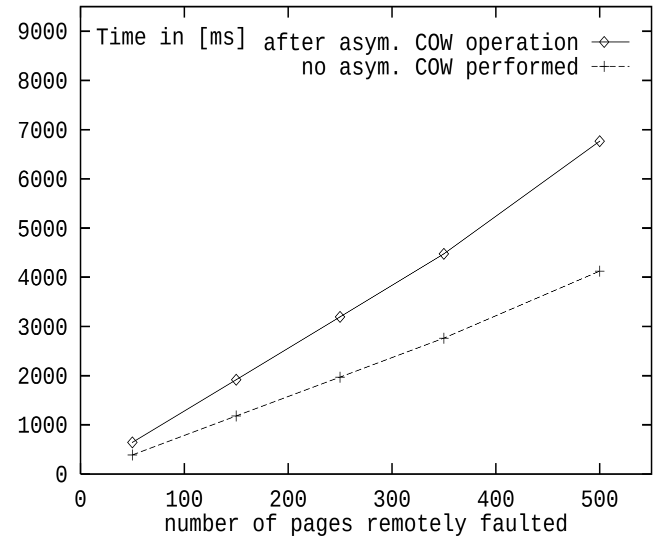

Figure 10: Remote faults after an asymmetric distributed COW operation is performed: the costs are almost doubled.

of nodes. This is most likely caused by bottlenecks in the old network IPC implementation used for these measurements; we intend to verify this in the near future by rerunning these test on an improved (rewritten) implementation.

Eliminating the distribution of asymmetric copy on write operations results in major performance improvements. Not only can the processing required to perform one of these operations in a distributed fashion be costly, (one second if the operation must be performed to all of the nodes in a 64 node Paragon), but the resulting structures impose severe performance penalties on page faults that must traverse them. This is because a page fault must check with the potential source of a copy operation that pushes a page higher in the shadow chain to ensure that page being faulted on was not copied in this fashion on another node while the fault was in progress. These checks require node to node communication that imposes performance costs. Figure 10 shows the significant performance impact of the presence of one distributed asymmetric copy on write operation in a shadow chain. Our work completely eliminates this impact by confining copy on write operations to single nodes.

Besides being a performance bottleneck, the centralized architecture also penalizes consumption of operating system resources, such as memory. For a small number of remote tasks created, this is not observable, however for a large number, resources on the source node can become critical. The hierarchical approach relaxes resource usage by distributing the consumption across the nodes involved in a hierarchical remote task creation. Without this work, creating a typical Mach task on a thousand nodes would consume over 10Mb of memory on the source node. Hierarchical creation eliminates this lopsided impact on the source node.

7 Related work

A number of systems have implemented process migration based on eager copying of the address space. These include MOS(IX) [2], Chorus [7], and Locus TCF [12]. Optimizations to overcome these costs include pre-copying in the V kernel [10], copy on reference in Accent [14] and flushing dirty pages to disk in Sprite [5]. While our basic technology for remote task creation is similar to these systems, none of them addressed the problem of creating multiple tasks or processes at the same time.

The NX/2 [8] operating system supported concurrent remote process creation. The address spaces are created by loader in a spanning tree fashion. Performance is function of $\log n$, and it is even further improved by using advanced communication mechanisms. Resulting performance ranged from as high as a fraction of a minute for huge tasks to as low as a couple of seconds. However, beside loading code and data, there are no additional requirements, such as paging path construction. This significantly simplifies the algorithm and allows for performance optimization, but it does not support programs with large address spaces.

Functionality and complexity similar to our concurrent remote task creation, albeit at the user level, is provided by the Locus TNC operating system [13]. Although it is possible to optimize the initial loading of code and data by performing it in a spanning tree fashion, this cannot be achieved with the paging path. The underlying Mach distributed memory management still enforces a centralized paging structure.

Barrera did the original work on extending Mach to distributed environment [3], and his implementation is the base on which we have implemented and to which we compare our improvements. His main contributions are in unifying support for read-only and distributed shared memory for remote task creation and migration, as well as for improved caching. As discussed earlier in the paper, this implementation suffers from severe performance problems that motivated our work.

Milojicic et al., supported task migration in user space by providing copy on reference through a user level pager [6]. Although we are of the opinion that moving functionality into the user space is better architecturally, the existing Mach VM prevented us from doing so efficiently. The kernel is currently the best (and only) source of knowledge about pages; hence it is the appropriate place to start the address space transfer.

8 Conclusion

The performance of parallel and distributed application startup is an important, yet often overlooked component of operating system support for parallel and distributed processing. This paper has described our work to address this shortcoming in the Mach microkernel by implementing concurrent remote task creation and redesigning the associated distributed memory management support. Our implementation makes extensive use of concurrency among kernels and hierarchy in the memory management architecture to achieve better performance and scalability. Separating the common virtual copy (copy on reference) case from the distributed shared memory system allowed us to focus optimization efforts on it, achieving performance improvements and code simplification for both it and the distributed shared memory case. These results improve the ability of operating systems to utilize the computational resources of new MPP hardware and workstation clusters. Parallel applications benefit from faster startup and distributed paging, improving both performance and scalability of the system as a whole.

Acknowledgments

We would like to thank Fred Douglis, Alan Langerman, Simon Patience, and the anonymous IPPS reviewers for suggestions that have greatly improved this paper.

References

[1] Accetta, M., et al., “Mach: A New Kernel Foundation for UNIX Development”, *Proceedings of the Summer USENIX Conference,* Atlanta, GA, pp 93-112, 1986.

[2] Barak, A., Shiloh, A., “A Distributed Load-Balancing Policy for a Multicomputer”, *Software-Practice and Experience*, vol. 5, no 9, pp 901-913, September 1985.

[3] Barrera, J., “Unpublished Work, Norma 14 Code Base”, Carnegie-Mellon University, 1994.

[4] Bryant, B., Sears, S., Black, D. and Langerman, A., “An Introduction to Mach 3.0’s XMM Subsystem”, *OSF RI Operating Systems, Collected Papers,* vol. 2, October 1993.

[5] Douglis, F., Ousterhout, J, “Transparent Process Migration: Design Alternatives and the Sprite Implementation”, *Software-Practice and Experience*, vol. 2, no 8, August 1991, pp 757-785.

[6] Milojicic, D., Zint, W., Dangel, A., Giese, P., “Task Migration on the top of the Mach Microkernel”, *Proceedings of the third USENIX Mach Symposium*, pp 273-290, Santa Fe, New Mexico, April 1993.

[7] Philippe, L., “Contribution à l‘étude et la réalisation d‘un système d‘exploitation à image unique pour multicalculateur”, *Technical Report 308, Ph.D. Thesis*, Université de Franche-comté, 1993.

[8] Pierce, P., “The NX/2 Operating System”, *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, vol 3, pp 384-390, January 1988.

[9] Rashid, R., et al., “Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures”, *IEEE Transactions on Computers*, vol. 37, no. 8, pp 896-908, August 1988.

[10] Theimer, M., Lantz, K., Cheriton, D., “Preemptable Remote Execution Facilities for the V System”, *Proceedings of the 10th ACM Symposium on OS Principles*, pp 2-12, December 1985.

[11] Young, M. W., et al., “The Duality of Memory and Communication in the Implementation of Multiprocessor Operating System”, *Proceedings of the Symposium on Operating System Principles*, pp 63-76, November 1987.

[12] Walker, B. J., Mathew, R. M., “Process Migration in AIX’s Transparent Computing Facility”, *IEEE TCOS Newsletter,* Winter 1989, vol. 3(1), pp 5-7.

[13] Zajcew, R., et al., “An OSF/1 UNIX for Massively Parallel Multicomputers”, *Proceedings of the Winter USENIX Conference,* January 1993, pp 449-468.

[14] Zayas, E., “Attacking the Process Migration Bottleneck”, *Proceedings of the 11th Symposium on Operating Systems Principles,* pp 13-24, November 1987.