Transparent Process Migration for Distributed Applications in a Beowulf Cluster

Mark Claypool and David Finkel
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609 USA
{claypool | dfinkel}@cs.wpi.edu

Abstract

Powerful, low-cost clusters of personal computers, such as Beowulf clusters, have fueled the potential for widespread distributed computation. While these Beowulf clusters typically have software that facilitates development of distributed applications, there is still a need for effective distributed computation that is transparent to the application programmer. This paper describes the design and development of the PANTS Application Node Transparency System (PANTS), which enables transparent distributed computation. PANTS employs a fault-tolerant communication architecture based on multicast communication that minimizes load on busy cluster nodes. PANTS runs on any modern distribution of Linux without requiring any kernel modifications.

1. Introduction

Cluster computing offers several benefits to application programmers and to users. A large class of computations can be broken into smaller pieces and executed by the various nodes in a cluster. However, sometimes on a cluster it can be beneficial to run an application that was not designed to be cluster-aware. One of the main goals of our research is to support such applications.

We have developed a system for distributed applications running in a cluster environment. Our system automatically detects which nodes of the cluster are overloaded and which are underloaded, and transfers work from overloaded nodes to underloaded nodes so that the overall application can complete more quickly. Two fundamental design goals of our system are that its operation should be transparent to the programmer and to the user of the distributed application, and that the system should impose only minimal overhead on the system.

Our system is called PANTS, an acronym for the PANTS Application Node Transparency System. It runs on a Beowulf cluster, which is a cluster of PC-class machines running a free/open-source operating system, in our case Linux, and connected by a standard local area network.

PANTS is designed to be transparent to the application as well as the programmer. This transparency allows an increased range of applications to benefit from process migration. Under PANTS, existing multi-process applications that are not built with support for distributed computation can now run on multiple nodes by starting all the processes of the application on a single node and allowing PANTS to migrate the individual processes of the application to other

nodes. As far as the application is concerned, it is running on a single computer, while PANTS controls what cluster computation resources it is using.

The PANTS design provides a method for minimal inter-node communication and for fault tolerance. In a Beowulf system, the network is most often the performance bottleneck. With this in mind, PANTS keeps the number of message sent between machines low and also uses a protocol that does not exchange messages with nodes that are busy with computations. Built-in fault tolerance allows the cluster to continue functioning even in the event that a node fails. In the same way, nodes can be added or removed from a cluster without dramatic consequences.

2. The Design and Implementation of PANTS

2.1 Beowulf and Distributed Applications

In general, a Beowulf cluster is a collection or cluster of personal computers connected by a private network (Sterling et al, 1998). The principal goal of the development of Beowulf clusters is to build low cost, scalable systems using relatively inexpensive personal computers and networks in order to support distributed parallel applications. Our Beowulf cluster consists of seven Alpha personal computers, connected by a private Ethernet. Many Beowulf clusters are much larger (Becker and Merkey).

There are many libraries designed to simplify the creation of distributed parallel applications for Beowulf clusters:

- PVM, Parallel Virtual Machine, is a runtime message-passing system that is easy to use, portable, and widely popular on parallel systems. It has been designed so that users without system-administration privileges could install the software and run parallel jobs from their shell accounts (PVM).
- MPI, Message Passing Interface, provides a complete library specification for message-passing primitives and has been widely accepted by vendors, programmers, and users (MPIF).
- DIPC, Distributed IPC, provides distributed program developers with semaphores, messages and transparent distributed shared memory (Karimi).
- BPROC, Beowulf Distributed Process Space, allows a node to run processes that appear in its process tree even though the processes are not physically on the node itself (Hendriks).

While these libraries are very effective for writing distributed applications, they also require that the programmer explicitly implement the communications between nodes, and write the applications for a specific library. One goal of PANTS is to overcome these limitations by creating an environment where load sharing of multi-process applications can take place automatically, without the need for the application programmer to think about the details of working in a cluster environment.

2.2 PANTS and Process Migration

Process migration can be preemptive or non-preemptive. In preemptive process migration, processes can be migrated while they are running. In non-preemptive process migration, processes are only eligible for process migration when they are first initiated. There is more overhead in

supporting preemptive process migration since the run-time state of the process must be migrated. On the other hand, preemptive process migration would seem to have more potential to improve performance, since there are more opportunities to migrate processes. Previous research has suggested (Eager et al, 1998) that non-preemptive process migration can provide significant performance improvement with lower overhead than preemptive process migration.

A preliminary version of PANTS (Moyer) provided a tool for distributed applications on an Intel-based Beowulf cluster through preemptive process migration. It was implemented by making changes to the Linux kernel. A crucial step was the use of EPCKPT, a Linux process checkpointing utility (Pinheiro). Thus, processes at a busy node could be stopped in mid-computation, checkpointed, and moved to another, less busy, node, without intervention or even the knowledge of the user.

After the development of this preliminary version, there were two issues we wanted to address. First, the use of kernel modifications meant that the PANTS software might have to be modified with each new kernel release. To avoid this version chasing, we wanted to eliminate the need for kernel modifications. Second, since there were many Intel-specific aspects to EPCKPT, the use of the EPCKPT utility restricted the use of the system to Intel systems. In order to make PANTS platform-independent, we wanted to remove the dependency on EPCKPT. This required us to restrict our system to non-preemptive process migration.

2.3 The Load-Balancing Algorithm

An important aspect of any process migration scheme is how to decide when to transfer a process from a heavily loaded node to a lightly loaded node. We call this the load-balancing algorithm. Many load-balancing algorithms for distributed systems have been proposed; there is a summary in (Wills and Finkel, 1995). We have implemented a variation of the multi-leader load-balancing algorithm proposed in (Wills and Finkel, 1995).

In this algorithm, one of the nodes is required to be the leader. The leader can be any node in the cluster, and is chosen randomly from among all of the nodes. The leader has three basic responsibilities: accept information from lightly-loaded ("available") nodes in the cluster, use that information to maintain a list of available nodes, and return an available node to any client that requests it. In the current project, for fault tolerance, we implemented an election procedure to select a new leader in case the leader node fails.

An available node is one that is lightly loaded, as measured by CPU utilization. When any node in the cluster becomes available, it sends a message to the leader, indicating that it is free to accept new work. If a node becomes unavailable, for example if it begins a new computation, it sends another message to the leader. Thus, the leader always knows which nodes are available at any time. If a node wants to off-load work onto another node, it need only ask the leader for an available node, then send the process to that node.

The actual implementation is a variation of the multi-leader policy described in (Wills and Finkel, 1995) and implemented in (Moyer). In many other load-balancing algorithms, either nodes that are

available or nodes that need to off-load work broadcast their status to all the nodes in the cluster. These broadcast messages frequently need to be handled by machines that are heavily loaded ("busy machines"). The multi-leader algorithm avoids these "busy-machine messages" by sending messages only to the leader multicast address.

We modified the Wills-Finkel policy (Wills and Finkel, 1995) to simplify the implementation and improve fault tolerance, at the cost of a small increase in the amount of network traffic. In PANTS, there are two multicast addresses. One of the addresses is read only by the leader; both available nodes and nodes with work to off-load send to the leader on this address. Because the leader is contacted via multicast, the leader can be any node in the cluster, and leadership can change at any time, without needing to update the clients. Available nodes receive on the other multicast address. This address is only used when the leader needs to discover, for the first time, which nodes are available. Because multicast is used to communicate with available nodes, busy nodes are not even aware of the traffic.

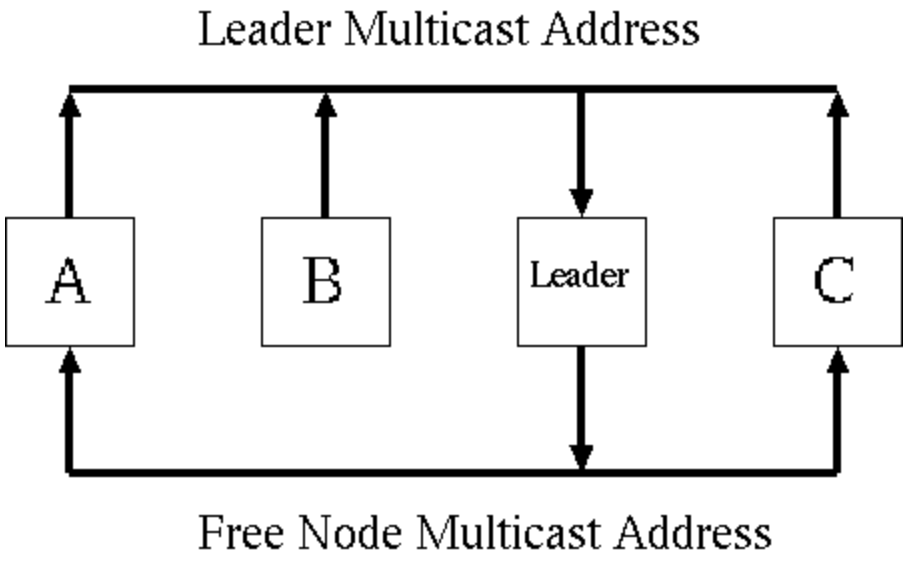

Figure 1: PANTS Multicast Communications

Figure 1 depicts the multicast communication among PANTS nodes. There are four nodes in this Beowulf cluster, one of which is the leader. Nodes A and C are "free" nodes, having little computation load and Node B is a "busy" node. All Nodes can communicate with the leader by sending to the leader multicast address. The leader communicates with all free nodes, A and C in this example, by sending to the free node multicast address. Node B is not "bothered" by messages to the free nodes since it is not subscribed to that multicast address.

2.4 PANTS Implementation

PANTS is composed of two major software components, the PANTS daemon and PREX, which stands for PANTS remote execute. The PANTS daemon is responsible for coordinating the

available resources in the cluster. It communicates among nodes to determine which nodes are available to receive processes. PREX intercepts the execution of a process, queries the PANTS daemon for an available node and remotely executes the process to distribute load among the nodes in the cluster.

Using the multicast policy described in Section 2.3, the PANTS daemon can respond quickly to a request from PREX for an available node.

PREX is made up of a library object called `libprex.o` and a remote execution program called prex. The library object is designed to intercept programs when they are initially executed and then send them to the prex program.

The way libprex works with the C library allows it to intercept processes transparently. To enable libprex, the Linux environment variable LD_PRELOAD is set to reference the `libprex` library object. Doing this causes the library functions in libprex to override the usual C library functions. When programs call the execve function to execute a binary (for example, when a user executes a binary from a command line shell), our version of `execve()` is used instead of the original one. Inside of our `execve()` function, the real C library `execve()` is invoked to execute `prex`, whose arguments are the process name, followed by the original arguments for that process. `prex` can then use `rsh` to remotely execute the process.

When prex is invoked for a process, the process is checked for migratability, which is determined by flags set within the binary. Several user-defined bits within the flag area of the binary's header signal whether the process should be migrated. If the binary is migratable, `prex` queries the local PANTS daemon to determine if the local machine is busy with other computations. If the process is not migratable, or the local machine is not busy, the binary is executed on the local machine by calling the original version of `execve()`. If the local machine is busy, `prex` queries the local PANTS daemon for an available node. If a node is returned, prex calls `rsh` to execute the process on that node. If all nodes are busy, the process is executed locally.

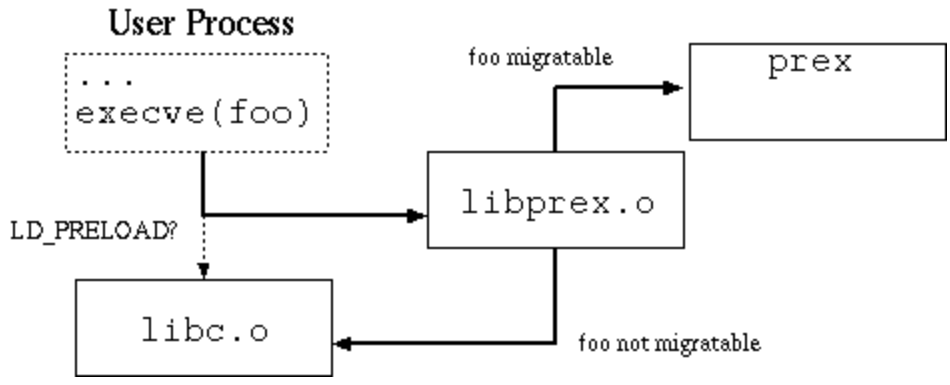

Figure 2: PREX Functionality

Figure 2 depicts the functionality of PREX. When a process calls `execve()` and the environment variable LD_PRELOAD is set, `libprex.o` intercepts the libc.o version of `execve()`. If the executable file is migratable, `libprex.o` invokes `prex` to communicate with the PANTS daemon to find a free node and migrate the process. If the executable is not migratable, `libprex.o` invokes the normal `libc.o` version of `execve()`.

One of the more difficult responsibilities of PANTS is determining whether or not each node is busy with a computation or available to receive a task. The original version of PANTS used a periodic check of the percentage of CPU utilization to make this decision. However, CPU load is not the only possible way to measure load. A process can require large amounts of memory, or I/O activity, which can limit its performance. The current version of PANTS considers CPU, memory, I/O and interrupts in making load decisions. For each load component, a threshold is chosen, and any node with utilization greater than the threshold of any one component is considered "busy," while others are "free." More details on PANTS load measurements can be found in (Nichols and Lemaire, 2002).

3. Results

The PANTS system has been fully implemented and is running on our Beowulf cluster of Alpha processor workstations. Our cluster consists of seven 64-bit Alpha 600 MHz workstations connected on a private subnet via a 100 Mbps switch. Each machine has between 64 and 512 MB of RAM and at least one SCSI hard drive. The PANTS system was thoroughly tested and found functionally correct and small scale testing showed a nearly linear speed-up with PANTS for a computation-intensive numerical application.

In order to further evaluate the performance and transparency of PANTS, we used compilation of the Linux kernel as a benchmark and ran it on our Beowulf cluster. In order to enable PANTS migration, we marked the `gcc` binary as migratable. The Linux kernel version used was 2.4.18 (the same Linux kernel version our cluster was running), with 432 files with a mean file size of 19 Kbytes.

Our first test was a compile where the Linux source tree was stored on the local hard disk without PANTS running. The second test was a compile where the Linux source tree was stored on the NFS mounted disk. The third test was a compile where the Linux source tree was stored on the NFS disk and PANTS was running, but was forced to never choose to migrate the process. The fourth test was a compile where the Linux source tree was stored on the NFS disk and PANTS was running with migration enabled. We ran each test 5 times and computed the average compile time in seconds. The results are depicted in Figure 3.

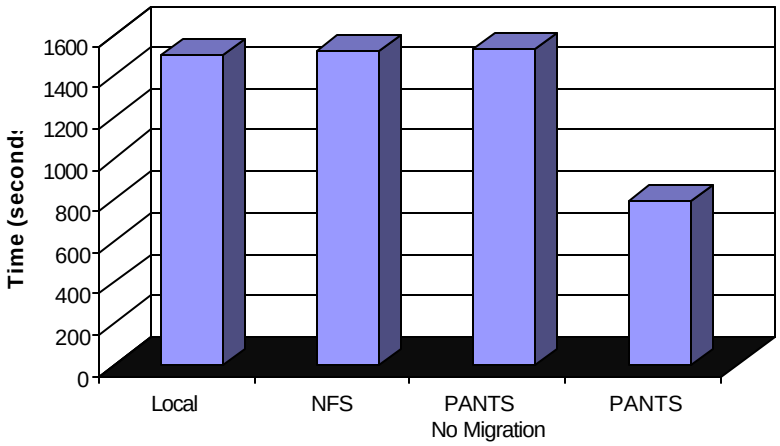

Figure 3: Linux Kernel Compile Time

The local compile took 1520 seconds (about 25 minutes). Accessing the Linux source tree over NFS added about 20 seconds to the total compile time, approximately a 2% overhead. Adding PANTS without migration added an additional 5 seconds to the NFS compile time, approximately a 0.5% overhead. PANTS, fully enabled with migration, reduced the compile time to about 670 seconds, approximately a 55% reduction in compile time.

We have also demonstrated that PANTS is compatible with the distributed interprocess communications package DIPC (Karimi). The use of DIPC requires the programmer to include DIPC-specific code in the program, and thus compromises our transparency goal of not requiring any special code for the distributed application. However, the use of DIPC does allow the programmer to use a wider range of interprocess communication primitives, such as such as shared memory, semaphores, and message queues, than would otherwise not be available (Szelag et al, 2001).

4. Future Work

Although PANTS shows significant potential in distributed processing, there are a number of ways that we see PANTS growing into a more mature cluster environment.

Preemptive migration is the act of moving a process from one node to another when it is already running. It has been shown in (Barak et al, 1996) that there are situations where preemptive process migration can give a significant performance benefit over a simple remote execution scheme. Specifically, when distributed applications are executed on a shared network of workstations, rather than on a single-user cluster of dedicated nodes, preemptive migration allows more flexible use of idle workstations. As discussed above, the original version of PANTS implemented preemptive migration for the Intel platform only, and in the current version we decided to implement the non-preemptive, but platform independent, PREX. BPROC, the Beowulf Distributed Process Space, supports preemptive process migration on many platforms (Hendriks). Thus, the capability for

preemptive migration could be added back to PANTS without affecting platform independence, by borrowing generously from BPROC.

A project with similar goals to our project is the MOSIX system (Barak). MOSIX provides preemptive process migration in a MOSIX Linux cluster. The process migration is transparent to the programmer. As in our system, a user can start multiple processes on a single node and allow the process migration system to relocate the processes on other nodes to improve performance. In the future, we hope to compare the performance and capabilities of PANTS with MOSIX.

5. Conclusions

Distributed computation has grown in popularity recently, earning attention from scientists and even corporations. Accompanied with the dramatic growth of Linux, Beowulf clusters provide a cost effective solution to today's large computation needs. PANTS and its use of transparent load sharing takes another step in the direction of ever more powerful cluster technology. PANTS is designed to run on any system running a modern distribution of Linux regardless of the underlying hardware architecture. Transparency, reduced busy node communication, and fault tolerance make PANTS a viable solution for more effective use of a Beowulf cluster.

Acknowledgements

This research was supported by equipment grants from Alpha Processor, Inc. and from Compaq Computer Corporation. The authors would also like to thank the following WPI students who have contributed to various phases of the PANTS project: Jeffrey Moyer, Kevin Dickson, Chuck Homic, Bryan Villamin, Michael Szelag, David Terry, Jennifer Waite, Marc Lemaire and Jim Nichols.

Bibliography

Barak, A., "MOSIX – Scalable Cluster Computing for Linux", Online at:
`http://www.cs.huji.ac.il/mosix/`

Barak, A., Braverman, A., Gilderman, I. and Laden, O. (1996), "Performance of PVM with the MOSIX Preemptive Process Migration Scheme", In *Proceedings of the 7th Israeli Conf. on Computer Systems and Software Engineering*, June.

Becker, D. and Merkey, P., "The Beowulf Project", Online at: `http://www.beowulf.org/`

Eager, D., Lazowska, E. and Zahorjan, J. (1998), "The Limited Performance Benefits of Migrating Active Processes for Load Sharing", In *Proceedings of ACM Measurement & Modeling of Computer Systems (SIGMETRICS)*, pp. 63-72, May.

Hendriks, E., "BPROC: Beowulf Distributed Process Space", Online at:
`http://www.beowulf.org/software/bproc.html`

Karimi, K., “Welcome to DIPC”, Online at: http://www.gpg.com/DIPC/

Moyer, J., “PANTS Application Node Transparency System”, Online at:
http://segfault.dhs.org/ProcessMigration/

MPIF, “Message Passing Interface Forum Home Page”, Online at:
http://www.mpi-forum.org/

Nichols, J. and Lemaire, M. (2002), “Performance Evaluation of Load Sharing Policies on a Beowulf Cluster”, *Major Qualifying Project MQP-MLC-BW01*, Spring. Online at:
http://www.cs.wpi.edu/~claypool/mqp/pants-load/

Pinheiro, E., “EPCKPT - A Checkpoint Utility for Linux Kernel”, Online at:
http://www.cs.rochester.edu/u/edpin/epckpt/

PVM, “PVM: Parallel Virtual Machine”, Online at:
http://www.epm.ornl.gov/pvm/pvm_home.html

Sterling, T., Salmon, J., Becker D. and Savarese, D. (1998), *How to Build a Beowulf*, MIT Press, Cambridge, Massachusetts USA.

Szelag, M., Terry, D. and Waite, J. (2001), “Integrating Distributed Inter-Process Communication with PANTS on a Beowulf Cluster”, *Major Qualifying Project MQP-DXF-0021*, Spring. Online at:
http://www.cs.wpi.edu/~claypool/mqp/pants-dipc/

Wills, C. and Finkel, D. (1995), “Scalable Approaches to Load Sharing in the Presence of Multicasting”, *Computer Communications*, 18(9), pp. 620-630, September.