Putting PANTS on Linux: Transparent Process Migration in a Beowulf Cluster [*]

Kevin Dickson, Chuck Homic, Bryan Villamin,
Mark Claypool, and David Finkel

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609 USA

April 17, 2000

Introduction

PANTS is the PANTS Application Node Transparency System. It provides automatic and transparent load sharing on a Beowulf cluster of Linux computers. PANTS manages the resources of the cluster for the user and executes processes remotely to share the computation load among the nodes in the cluster.

The benefits of cluster computing are well known. A large class of computations can be broken into smaller pieces and executed by the various nodes in a cluster. Sometimes, however, it can be beneficial to run an application on a Beowulf cluster that was not designed to be cluster-aware. This is one of the main goals of PANTS.

PANTS was 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, not built with cluster computing in mind, can now run on multiple nodes by invisibly migrating the individual processes of the application. As far as the application is concerned, it is running on a single computer, while PANTS controls what resources it is using.

The PANTS design also contains a method for minimal inter-node communication and fault tolerance. In a Beowulf system, the network is most often the performance bottleneck. With this in mind, PANTS keeps the number of

[*]This research has been supported by equipment grants from Alpha Processor, Inc. and from Compaq Computer Corporation.

message that move between machines low and also uses a protocol which does not exchange messages with nodes busy with computation. 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.

Beowulf and Distributed Applications

A Beowulf is a collection or cluster of personal computers that are connected to each other via Ethernet. The main idea behind the Beowulf project was to build low cost but scalable parallel computer system using relatively inexpensive personal computers.

There are many libraries designed to simplify the creation of parallel applications for Beowulf. PVM, Parallel Virtual Machine, is a runtime message-passing system which 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. MPI, Message Passing Interface, provides a complete library specification for message-passing primitives and has been widely accepted by vendors, programmers, and users. DIPC, Distributed IPC, provides distributed program developers with semaphores, messages and transparent distributed shared memory. BPROC, Beowulf Distributed Process Space, allows a node to run processes which appear in its process tree even though the processes are not physically on the node itself.

While these libraries are very effective for writing distributed applications, they also require that applications be written for a specific library. The goal of PANTS is to overcome this limitation, by creating an environment where load sharing of multi-process applications can take place automatically, without the need to think about the details of working in a cluster environment.

The first version of PANTS, created by Jeff Moyer [4], also provided a tool for distributed applications, through process migration. It successfully demonstrated transparent preemptive migration, by making changes to the Linux kernel. It was able to migrate running processes through the use of EPCKPT, a process checkpointing utility by Eduardo Pinheiro [5]. Thus processes at a busy node could be stopped in mid-computation and moved to another, less busy, node, without intervention or even the knowledge of the user.

The current project is directed at continuing the development of PANTS. In particular, 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, the use of the EPCKPT utility restricted the use of the system to Intel systems, since there were many Intel-specific aspects to EPCKPT. In order to make PANTS platform-independent, we removed the dependency on EPCKPT, and developed a system that would only migrate processes at the time of process initiation.

PANTS Concept

PANTS is composed of two major components, the PANTS daemon and PREX. 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 daemon for an available node and remotely executes the process to distribute load among the nodes in the cluster.

PANTS Daemon

An instance of the PANTS daemon runs on each node, and collects and shares load information. This information is made available to PREX when it is needed. Our implementation of the PANTS daemon is based on the daemon developed in [4]. See Figure 1 for a representation of a PANTS communication.

One of the nodes is required to be the leader. The leader can be any node in the cluster, and is chosen randomly among all of the nodes. It has three basic responsibilities: accept load information from each of the 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.

An available node is one that is not busy working on a computation. 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 offload 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 multicast leader policy described in [6], and implemented in [4]. This policy was designed to minimize the number of "busy machine messages," that is the number of messages that need to be handled by a node busy with computation. We modified the Wills-Finkel policy [6] 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 only used by the leader. 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. The other multicast address is for available nodes. 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.

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

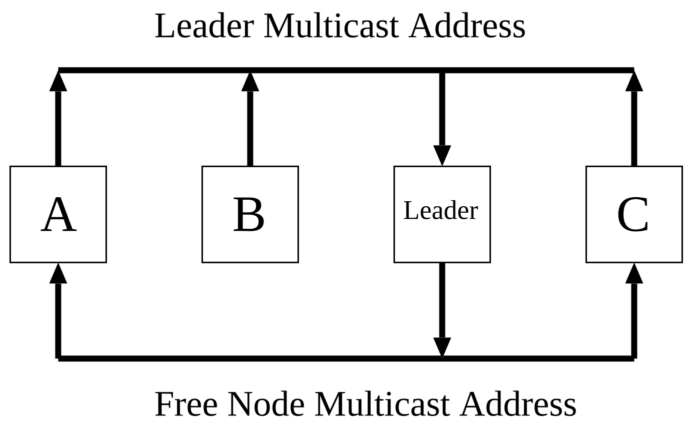

Figure 1: **PANTS Multicast Communication. The above figure depicts the multicast communication among PANTS daemons. There are four nodes in this Beowulf cluster, one of which is the leader. Nodes A and C are "free" nodes, having little computation load and Node B is a "busy" node. All Nodes can communicate 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.**

PREX

PREX, which stands for PANTS remote execute, 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. When `prex` receives the pathname of an executable binary, the binary is first checked for its ability to be migrated. If deemed migratable, it is executed remotely via `rsh`, after a query to the PANTS daemon returns the address of the node to which the process will be sent. The flow of execution using PREX is shown in Figure 2.

The way `libprex` works with the C library allows it to intercept processes transparently. To enable `libprex`, the 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, 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 isn't migratable, or the local machine isn't 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.

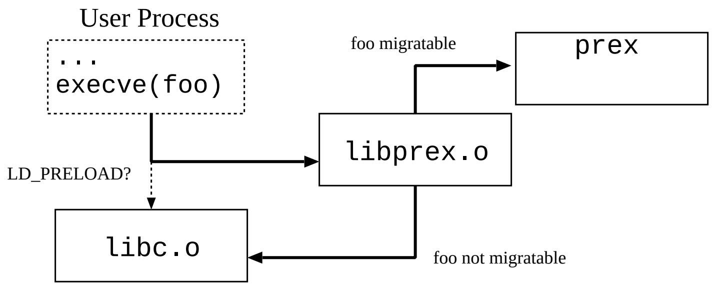

Figure 2: **PREX Functionality. The above figure depicts the functionality of PREX. When a process calls `execve()` and the environment variable `LD_PRELOAD` is set, `libprex.o` intercepts the `libc.o` version of `execve()`. If the executable file is migratable, `libprex.o` invokes `prex` to communicate 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()`.**

Results and Future Work

The PANTS system has been fully implemented, and is running on our Beowulf cluster of Alpha processor workstations. The system has been thoroughly tested, and is functionally correct. Although we have not yet performed extensive performance tests, small scale testing has shown nearly linear speed-up with PANTS with a computationally-intensive numerical application.

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.

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. Currently, PANTS uses a periodic check of the percentage of CPU utilization to make this decision. A threshold is chosen, and any node with CPU utilization greater than the threshold is considered "busy." Others are "free." However, changing this threshold, or changing how often load is measured, can have a significant impact on the overall performance of the cluster. Choosing the correct threshold and testing period, then, will be an important goal to maximize the effectiveness of a PANTS cluster. However, CPU load is not the only kind of "load" possible. A process can also require large amounts of memory, or I/O activity, which can limit its performance. A study of which variables to measure, and how to evaluate them may also be important to maximize performance.

Preemptive migration is the act of moving a process from one node to another when it is already running. It has been shown in [1] 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 [2]. Thus, the capability for preemptive migration could be added back to PANTS, without affecting platform independence, by borrowing generously from BPROC.

The current PANTS implementation uses `rsh` to execute processes remotely. Therefore, the only method of communication to a remote process is through stdin/stdout. Often, this is sufficient, but to be truly transparent, PANTS will have to offer distributed IPC services, such as shared memory, semaphores, and message queues. DIPC, Distributed Inter-Process Communication, provides these capabilities [3]. Integrating DIPC into PANTS will allow a much wider range of distributed applications to benefit from executing on a PANTS cluster.

Conclusion

Distributed computation has grown in popularity recently, earning attention from scientists and even corporations. Accompanied with the dramatic growth of Linux, Beowulf systems provide a cost effective solution to today's large computation needs. PANTS and its use of transparent load sharing makes 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 architecture. Transparency, reduced busy node communication, and fault tolerance make PANTS a viable solution for more effective use of a Beowulf system.

References

[1] Amnon Barak, Avner Braverman, Ilia Gilderman and Oren Laden. "Performance of PVM with the MOSIX Preemptive Process Migration Scheme." *Proc. 7th Israeli Conf. on Computer Systems and Software Engineering*, June 1996.

[2] Erik Hendriks. *BPROC: Beowulf Distributed Process Space.* `http://www.beowulf.org/software/bproc.html`

[3] Kamran Karimi. "DIPC." `http://www.gpg.com/DIPC/`

[4] Jeffrey Moyer. *PANTS Application Node Transparency System.* `http://segfault.dhs.org/ProcessMigration/`

[5] Eduardo Pinheiro. *EPCKPT - A Checkpoint Utility for Linux Kernel.* `http://www.cs.rochester.edu/u/edpin/epckpt/` *(Mirror site)*

[6] Craig E. Wills and David Finkel. "Scalable Approaches to Load Sharing in the Presence of Multicasting." *Computer Communications*, 18(9):620-630, September 1995.