Design and Implementation of a Process Migration System for the Linux Environment

Nalini Vasudevan * Prasanna Venkatesh †

Abstract

This paper reviews the field of process migration by summarizing the key concepts and giving an overview of a high level process migration algorithm and also provides a unique alternative implementation of our own for the linux environment. Design and implementation issues of process migration are analyzed in general, and these are used as pointers in describing our implementation. The primary aim of this paper is to build a user space process migration tool which would obviate the need for kernel support. This paper aims to provide an insight into the difficult task of actual migration for performance gain.

Keywords: process migration, fault resilience, load distribution, distributed systems

1 Introduction

Process migration is the act of transferring an active process between two machines and restoring the process from the point it left off on the selected destination node.The purpose of this is to provide for an enhanced degree of dynamic load distribution, fault resilience, eased system administration, and data access locality. The potential of process migration is great especially in a large network of personal workstations.In a typical working environment,there generally exists a large pool of idle workstations whose power is unharnessed if the potential of process migration is not tapped.Providing for a reliable and efficient migration module that helps handle inconsistencies in load distribution and allows for an effective usage of the system resource potential is highly warranted. As high-performance facilities shift from supercomputers to networks of workstations, and with the ever-increasing role of the World Wide Web, we expect migration to play a more important role and eventually to be widely adopted. This paper reviews the field of process migration by summarizing the key concepts and giving an overview of the most important implementations and by providing a unique alternative implementation of our own. Design and implementation issues of process migration are analyzed in general, and these are used as pointers in describing our implementation.

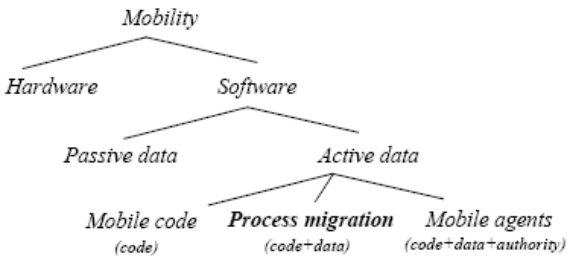

Figure 1: Process Migration and Mobility

Process migration, which is the main theme of this paper primarily involves transferring both code and data across nodes. It also involves transfer of authorisation information, for instance access to a shared resource, but the scope is limited to being under the control of a single administration. Finally mobile agents involve transferring code, data and complete authorisation information, to act on the behalf of the owner on a wide scale, such as the World wide internet.

2 Related Work

The future of process migration is extremely encouraging. Significant research and development has been conducted in process migration and closely related areas. Different streams of development may well lead to a wider deployment of process migration. [2] does a survey on process migration and highlights the importance of process migration. A process is suspended on a machine and resumed on another machine. Remote Procedure Calls, Copy by reference are the important

*Yahoo! Software Development India Pvt. Ltd. 4th floor, "Esquire Center", 9, M.G. Road, Bangalore, India - 560 001. Tel: +91 94481 07482 Email: nalinivasudevan@yahoo.com

†Yahoo! Software Development India Pvt. Ltd. 4th floor, "Esquire Center" , 9, M.G. Road, Bangalore, India - 560 001. Tel: +91 98458 55097 Email: prasanna.venkatesh@yahoo.com

mechanisms used in process migration as explained in [4] and [21]. The transfer of executing software between machines has been investigated by various distributed operating systems research and is termed process migration ([11],[12],[13],[25]). Process migration has been the subject of a consider able amount of research, and there have been a number of experimental implementations as highlighted in ([1],[8],[9],[11]). There are various approcaches to process migration and the mechanisms that can be used as the basis for implementing process migration in a distributed setting have been discussed in [3],[6],[7],[8],[14] and [18]. One path is in the direction of LSF, a user-level facility that provides much of the functionality of full-fledged process migration systems, but with fewer headaches and complications. The checkpoint/restart model of process migration has already been relatively widely deployed. Packages such as Condor, LSF and Loadleveler are used for scientific and batch applications in production environments. Those environments have high demands on their computer resources and can take advantage of load sharing in a simple manner. A second path concerns clusters of workstations as discussed in [12] and [24]. Recent advances in high speed networking have reduced the cost of migrating processes, allowing even costly migration implementations to be deployed. A third path, one closer to the consumers of the vast majority of today's computers (Windows systems on Intelbased platforms), would put process migration right in the home or office. One can imagine a process starting on a personal computer, and migrating its flow of control into another device in the same domain. Such activity would be similar to the migratory agents approach currently being developed for the Web .
Process migration is basically designed to share the load among various systems without yielding on other performances. A lot of research related activities can be found in [23], [24] and [23]. An operating system that provides process migration and transparent remote execution [9], [17], [19] and [20] facilitates with the aim of taking advantage of idle processors within a cluster. The virtualization([5]) and process migration provided can provide a plethora of advantages from fault tolerance to remote resource sharing. But, it has always faced significant technical challenges and are highlighted in [16]. In this paper we try to overcome the challenges and design an optimal system for the linux environment..

3 Overview of Process Migration

3.1 Goals

The goals of process migration are very closely tied to the applications that use migration, as described in the next section. The primary goals include:

- **Harnessing resource locality:** Processes which are running on a node which is distant from the node which houses the data that the processes are using tend to spend most of their time in performing communication between the nodes for the sake of accessing the data. Process migration can be used to migrate a distant process closer to the data that it is processing, thereby ensuring it spends most of its time doing useful work.

- **Resource sharing:** Nodes which have large amount of resources can act as receiver nodes in a process migration environment.

- **Effective Load Balancing:** Migration is particularly important in receiver-initiated distributed systems, where a lightly loaded node announces its availability thereby enabling the arbitrator to provide its processing power to another node which is relatively heavily loaded.

- **Fault tolerance:** This aspect of a system is improved by migration of a process from a partially failed node, or in the case of long running processes when different kinds of failures are probable. In conjunction with checkpointing, this goal can be achieved.

- **Eased system administration:** When a node is about to be shutdown, the system can migrate processes which are running on it to another node, thereby enabling the process to go to completion, either on the destination node or on the source node by migrating it back.

- **Mobile computing:** Users may decide to migrate a running process from their workstations to their mobile computers or vice versa to exploit the large amount of resources that a workstation can provide.

3.2 Applications

The type of applications that can benefit from process migration include:

- **Distributed applications** can be started on certain nodes and can be migrated at the application level or by using a system wide migration facility in response to things like load balancing considerations.

- **Multiuser Applications**, can benefit greatly from process migration. As users come and go, the load on individual nodes varies widely. Dynamic process migration can automatically spread processes across all nodes, including those applications that are not enhanced to exploit the migration mechanism.

- **Standalone Applications**, which is preemptable, can be used with various goals in mind. Such an application can either migrate itself, or it can be migrated by another authority. It is difficult to select such applications without detailed knowledge of past behavior, since many applications are short-lived and do not execute long enough to justify the overhead of migration

- **Long running applications**, which can run for days or weeks on end can suffer various interruptions, for example partial node failures or administrative shutdowns. Process migration can relocate these processes in the event of the occurences of any of the events mentioned above.

- **Migration-oriented Applications** are applications that have been coded to explicitly take advantage of process migration. Dynamic process migration can automatically redistribute these related processes if the load becomes uneven on different nodes, e.g. if processes are dynamically created, or there are many more processes than nodes.

- **Mobile applications** are the most recent example of the potential use of migration; for instance, mobile agents and mobile objects. These applications are designed with mobility in mind. Although this mobility differs significantly from the kinds of process migration considered elsewhere in this paper, it uses some of the same techniques: location policies, checkpointing, transparency, and locating and communicating with a mobile entity.

3.3 System Requirements

To support process migration effectively, a system should be able to provide the following functionality:

- Exporting/importing the process state: The system must provide some type of export/import interfaces that allow the process migration mechanism to extract a process's state from the source node and import this state on the destination node. These interfaces may be provided by the underlying operating system, the programming language, or other elements of the programming environment that the process has access to. State includes processor registers, process address space and communication state, such as open message channels in the case of message-based systems, or open files and signal masks in the case of UNIX-like systems.

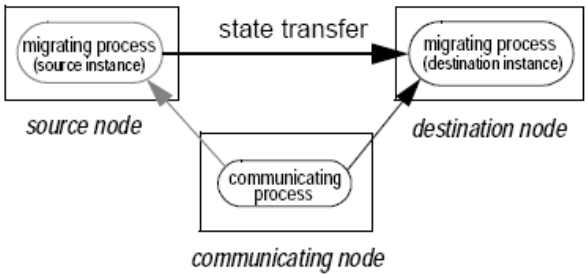

Figure 2: State transfer

- Naming/accessing the process and its resources: After migration, the migrated process must be accessible by the same name and mechanisms as it was before migration as though migration never occurred. The same applies to its resources, namely open files, threads etc.

3.4 Load Measurement Metrics

The load information is typically represented by means of the following metrics: the CPU usage, the memory availability, the average turnaround time etc. A process load is typically characterised by its CPU usage, memory usage, communication, file usage etc. Load information management deals with using these to select the process to be migrated and to choose the destination node.

3.5 Distributed Scheduling

This aspect mainly deals with allocation of nodes to processes. There are a plethora of strategies that are proposed, of which few are mentioned here:

- A sender-initiated policy is activated on the node

that is overloaded and that wishes to off-load to other nodes. A sender-initiated policy is preferable for low and medium loaded systems, which have a few overloaded nodes. This strategy is convenient for remote invocation strategies.

- A receiver-initiated policy is activated on underloaded nodes willing to accept the load from overloaded ones. A receiver-initiated policy is preferable for high load systems, with many overloaded nodes and few underloaded ones. Process migration is particularly well-suited for this strategy, since only with migration can one initiate process transfer at an arbitrary point in time

- A symmetric policy is the combination of the previous two policies, in an attempt to take advantage of the good characteristics of both of them. It is suitable for a broader range of conditions than either receiver-initiated or sender-initiated strategies alone.

- A random policy chooses the destination node randomly from all nodes in a distributed system. This simple strategy can result in a significant performance improvement

3.6 The Virtualisation concept

To provide for user space process migration while guaranteeing transparency, it is imperative that the processes do not realise that they are working in a migration enabled environment. Primarily, this involves building a uniform interface that the processes interact to which empowers the system to achieve the objective of transparency. This interface is sometimes termed the virtual interface and the very idea is called virtualisation.

4 User Space Implementation for the Linux Environment

4.1 The Generic Migration Algorithm

Although the implementations can vary, the primary steps in process migration can be summarised in the following steps:

1. A migration request is issued to a remote node. After negotiation, migration has been accepted.

2. A process is detached from its source node by suspending its execution, declaring it to be in a migrating state, and temporarily redirecting communication as described in the following step.

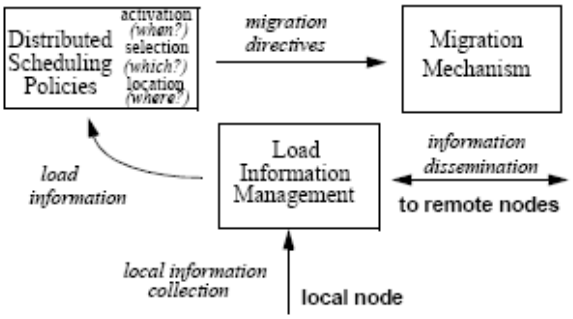

Figure 3: The Migration Process

3. Communication is temporarily redirected by queuing up arriving messages directed to the migrated process, and by delivering them to the process after migration. This step continues in parallel with steps 4, 5, and 6, as long as there are additional incoming messages. Once the communication channels are enabled after migration (as a result of step 7), the migrated process is known to the external world.

4. The process state is extracted, including memory contents, processor state (register contents), communication state (e.g., opened files and message channels) and relevant kernel context. The communication state and kernel context are OS dependent.

5. A destination process instance is created into which the transferred state will be imported. A destination instance is not activated until a sufficient amount of state has been transferred from the source process instance. After that, the destination instance will be promoted into a regular process.

6. State is transferred and imported into a new instance on the remote node. Not all of the state needs to be transferred; some of the state could be lazily brought over after migration is completed

7. Some means of forwarding references to the migrated process must be maintained. This is required in order to communicate with the process or to control it. This concludes step 3 enabling all communication to the original process to be permanently redirected to the new process.

8. The new instance is resumed when sufficient state has been transferred and imported. With this step, process migration completes. Once all of the state has been transferred from the original instance, it may be deleted on the source node.

4.2 Components

4.2.1 Central Server

The centralized server locates the appropriate machine when an overload signal is received. Every machine on the network communicates with this module to first establish its identity and then to communicate regarding the load status.

4.2.2 Load Balancer

The Load Balancer keeps track of the relative loads of the machine on the network and chooses the appropriate machine when an overload signal is received.

4.2.3 Checkpointer

This Checkpointer sends a signal to the process chosen by the Load Balancer for pre-emption and making it dump a core file. Thereafter, this module traverses the /proc directory to locate the attributes of the process selected for preemption. It then creates a file named filedescriptors which stores details about all the open files used by the process. These are used as checkpointing information

4.2.4 Restarter

The restarter uses the Linux machine's ptrace system call to get and set the values of the following attributes of the process

4.2.5 File Transferrer

The FileTransferrer establishes a UDP connection with the main server and transfers the aforementioned files to that server when the overload message is sent. The following files are sent by this module The core dump The filedescriptors file

4.2.6 Load Calculator

The Load Calculator is a daemon that runs on every process on the network to calculate the load at regular intervals. The calculation formula is a parametric equation based on various parameters. The total load on the system is calcualed as the sum of the loads of individual process. The memory and the cpu usage are the most significant parameters in the calculation of load. The /proc file system has one directory per process. The directories contain the relevant information regarding memory and cpu usage for all processes in the system. When the load exceeds a certain threshold value, this module sends an overload signal to the main server. This in turns begins the process of preemption, checkpointing and scheduling.

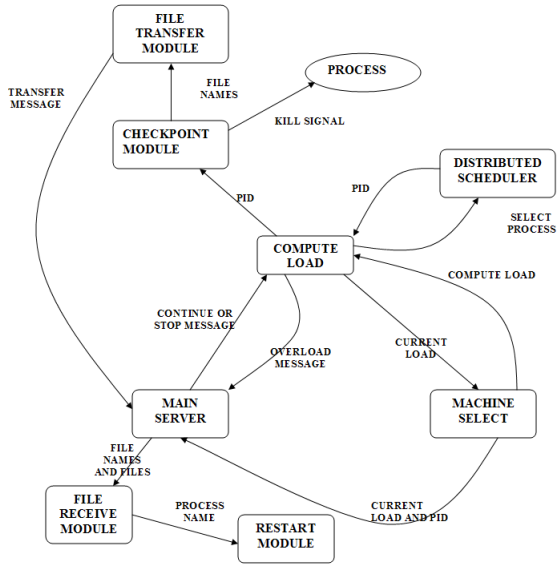

Figure 4: Process Migration and its modules

4.3 The Implementation

Every node on the network is registered with the central network. Additional nodes can be added by editing a configuration file. Every node has the load computation module running on it. This module periodically computes the current load on the node by parsing the /proc files and compares it against a predefined load factor (which depends on the application). When the load exceeds the preset threshold, the load computation module immediately sends a signal to the central system. The central server sends all the nodes on the network. On finding a suitable node using the random selection process, the server responds back with the appropriate IP address to the request-

ing node(RN). The RN, sets up a connection with that node, referred to as the destination node(DN). This is accomplished by having another server run on all the machines. After this, the RN sends a SIGQUIT to the overloading process and makes it dump the core file. This file is used to extract process state like register contents, currently executing address etc. The open file descriptors are saved which is written to a file. The file descriptors file and the extracted information are sent across to the destination node.

On the destination node, a new process is forked. The ptrace system call is used to provide the parent with the power to modify its contents. The relevant files are again opened and the file pointers are set to the appropriate location. Also, the register contents are modified and the parent relinquishes control. The new process is allowed to continue, while the old one is killed.

4.4 Results

The existing process migration tools largely work in the kernel / system space thereby creating a need for operating system support for migration. They cannot handle many conditions like opening of files and uncaught signals etc.
The following diagrams give a self explanation of the states of the nodes before and after the transfer for a threshold load of 0.4.
The following features are available with this imple-

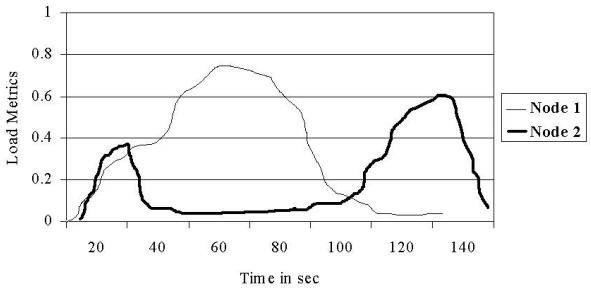

Figure 5: Before Migration (Threshold=0.4)

mentation of process migration:

1. It can migrate processes which have open files without any problem.

2. It takes care of processes spawning other processes

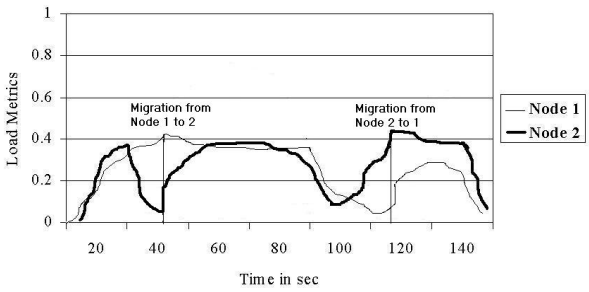

Figure 6: After Migration (Threshold=0.4)

3. It is a scalable architecture. This means that the no of nodes can be dynamically increased.

4. It has a transparent architecture, thereby obviating the need to recompile the user program for migration support.

5. It runs completely in the user space.

In addition to the above features, the implementation supports :

- Dynamic load distribution, by migrating processes from overloaded nodes to less loaded ones,
- Fault resilience, by migrating processes from nodes that may have experienced a partial failure,
- Improved system administration, by migrating processes from the nodes that are about to be shut down or otherwise made unavailable, and
- Data access locality, by migrating processes closer to the source of some data.

5 Conclusion

Despite these goals and ongoing research efforts, migration has not achieved widespread use. One reason for this is the complexity of adding transparent migration to systems originally designed to run stand-alone, since designing new systems with migration in mind from the beginning is not a realistic option anymore. Another reason is that there has not been a compelling commercial argument for operating system vendors to support process migration. Checkpoint-restart approach offers a compromise here, since they can run on more loosely coupled systems by restricting the

types of processes that can migrate. In spite of these barriers, process migration continues to be an object of widespread research and as mentioned above many successful attempts have been there in the past. The current attempt is one more approach towards chieving the ultimate goal of computational speedup.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19), pages 164-177. ACM Press, 2003.

[2] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. ACM Computing Surveys, 32(3):241-299, 2000.

[3] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M.Rosenblum. Optimizing the migration of virtual computers. In Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI-02), December 2002.

[4] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, 2002.

[5] Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Gribble. Constructing services with interposable virtual hardware. In Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04), 2004.

[6] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: A system for migrating computing environments. In Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI-02), pages 361-376, December 2002.

[7] Jacob G. Hansen and Asger K. Henriksen. Nomadic operating systems. Master's thesis, Dept. of Computer Science, University of Copenhagen, Denmark, 2002. Hermann Hrtig, Michael Hohmuth, Jochen Liedtke, and Sebastian Schnberg. The performance of micro-kernel-based systems. In Proceedings of the sixteenth ACM Symposium on Operating System Principles, pages 66-77. ACM Press, 1997.

[8] Michael L. Powell and Barton P. Miller. Process migration in DEMOS/MP. In Proceedings of the ninth ACM Symposium on Operating System Principles, pages 110-119. ACM Press, 1983.

[9] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In Proceedings of the tenth ACM Symposium on Operating System Principles, pages 2-12. ACM Press, 1985.

[10] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. ACM Trans. Comput. Syst., 6(1):109-133, 1988.

[11] Fred Douglis and John K. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. Software - Practice and Experience, 21(8):757-785, 1991.

[12] A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. Journal of Future Generation Computer Systems, 13(4-5):361-372, March 1998.

[13] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The Sprite network operating system. Computer Magazine of the Computer Group News of the IEEE Computer Group Society, ; ACM CR 8905-0314, 21(2), 1988.

[14] Jacob G. Hansen and Eric Jul. Self-migration of operating systems. In Proceedings of the 11th ACM SIGOPS European Workshop (EW 2004), pages 126-130, 2004.

[15] D. Nichols, Using idle workstations in a shared computing environment, Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, ACM, Austin, TX, November 1987, pp. 512.

[16] E. Zayas, Attacking the process migration bottleneck, Proceedings of the Eleventh ACM Symposium.on Operating Systems Principles, Austin, TX, November 1987, pp. 1322.

[17] M. Theimer, Preemptable remote execution facilities for loosely-coupled distributed systems, Ph.D. Thesis, Stanford University, 1986.

[18] Y. Artsy and R. Finkel, Designing a process migration facility: the Charlotte experience, IEEE Computer, 22, (9), 4756 (1989).

[19] A. D. Birrell and B. J. Nelson, Implementing remote procedure calls, ACM Transactions on Computer Systems, 2, (l), 3959 (1984).13. Computer Science Division, University of California, Berkeley, UNIX Users Reference Manual,4.3 Berkeley Software Distribution, Virtual VAX-11 Version, April 1986.

[20] M. Litzkow, Remote UNIX, Proceedings of the USENIX 1987 Summer Conference, June 1987.

[21] E. Zayas, The use of copy-on-reference in a process migration system, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, April 1987. Report No. CMU-CS-87-121.

[22] D. L. Eager, E. D. Lazowska and J. Zahorjan, The limited performance benefits of migrating active processes for load sharing, ACM SIGMETRICS 1988, May 1988.

[23] D. L. Eager, E. D. Lazowska and J. Zahorjan, Adaptive load sharing in homogeneous distributed systems, IEEE Trans. Software Engineering, SE-12, (5), 662675 (1986).

[24] E. H. Baalbergen, Parallel and distributed compilations in loosely-coupled systems: a case study,Proceedings of Workshop on Large Grain Parallelism, Providence, RI, October 1986.

[25] F. Douglis and J. Ousterhout, Process migration in the Sprite operating system, Proceedings of the 7th International Conference on Distributed Computing Systems, IEEE, Berlin, West Germany, September 1987, pp. 1825.