Process Migration

DEJAN S. MILOJÍČIĆ

HP Labs

FRED DOUGLIS

AT&T Labs–Research

YVES PAINDAVEINE

TOG Research Institute

RICHARD WHEELER

EMC

AND

SONGNIAN ZHOU

University of Toronto and Platform Computing

Process migration is the act of transferring a process between two machines. It enables dynamic load distribution, fault resilience, eased system administration, and data access locality. Despite these goals and ongoing research efforts, migration has not achieved widespread use. With the increasing deployment of distributed systems in general, and distributed operating systems in particular, process migration is again receiving more attention in both research and product development. 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 survey reviews the field of process migration by summarizing the key concepts and giving an overview of the most important implementations. Design and implementation issues of process migration are analyzed in general, and then revisited for each of the case studies described: MOSIX, Sprite, Mach, and Load Sharing Facility. The benefits and drawbacks of process migration depend on the details of implementation and, therefore, this paper focuses on practical matters. This survey will help in understanding the potentials of process migration and why it has not caught on.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; D.4.8 [**Operating Systems**]: Performance—*measurements*; D.4.2 [**Operating Systems**]: Storage Management—*distributed memories*

General Terms: Design, Experimentation

Additional Key Words and Phrases: Process migration, distributed systems, distributed operating systems, load distribution

1. INTRODUCTION

A process is an operating system abstraction representing an instance of a running computer program. Process migration is the act of transferring a process between two machines during its execution. Several implementations have been built for different operating systems, including MOSIX [Barak and Litman, 1985], V [Cheriton, 1988], Accent [Rashid and Robertson, 1981], Sprite [Ousterhout et al., 1988], Mach [Accetta et al., 1986], and OSF/1 AD TNC [Zajcew et al., 1993]. In addition, some systems provide mechanisms that checkpoint active processes and resume their execution in essentially the same state on another machine, including Condor [Litzkow et al., 1988] and Load Sharing Facility (LSF) [Zhou et al., 1994]. Process migration enables:

- **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.

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 approaches offer 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 attract research. We believe that the main reason is the potentials offered by mobility as well as the attraction to hard problems, so inherent to the research community. There have been many different goals and approaches to process migration because

of the potentials migration can offer to different applications (see Section 2.3 on goals, Section 4 on approaches, and Section 2.4 on applications).

With the increasing deployment of distributed systems in general, and distributed operating systems in particular, the interest in process migration is again on the rise both in research and in product development. As high-performance facilities shift from supercomputers to Networks of Workstations (NOW) [Anderson et al., 1995] and large-scale distributed systems, we expect migration to play a more important role and eventually gain wider acceptance.

Operating systems developers in industry have considered supporting process migration, for example Solaris MC [Khalidi et al., 1996], but thus far the availability of process migration in commercial systems is non-existent as we describe below. Checkpoint-restart systems are becoming increasingly deployed for long-running jobs. Finally, techniques originally developed for process migration have been employed in developing mobile agents on the World Wide Web. Recent interpreted programming languages, such as Java [Gosling et al., 1996], Telescript [White, 1996] and Tcl/Tk [Ousterhout, 1994] provide additional support for agent mobility.

There exist a few books that discuss process migration [Goscinski, 1991; Barak et al., 1993; Singhal and Shivaratri, 1994; Milojičić et al., 1999]; a number of surveys [Smith, 1988; Eskicioglu, 1990; Nuttal, 1994], though none as detailed as this survey; and Ph.D. theses that deal directly with migration [Theimer et al., 1985; Zayas, 1987a; Lu, 1988; Douglis, 1990; Philippe, 1993; Milojičić, 1993c; Zhu, 1992; Roush, 1995], or that are related to migration [Dannenberg, 1982; Nichols, 1990; Tracey, 1991; Chapin, 1993; Knabe, 1995; Jacqmot, 1996].

This survey reviews the field of process migration by summarizing the key concepts and describing the most important implementations. Design and implementation issues of process migration are analyzed in general and then revisited for each of the case studies described: MOSIX, Sprite, Mach, and LSF. The benefits and drawbacks of process migration depend on the details of implementation and therefore this paper focuses on practical matters. In this paper we address mainly process migration mechanisms. Process migration policies, such as load information management and distributed scheduling, are mentioned to the extent that they affect the systems being discussed. More detailed descriptions of policies have been reported elsewhere (e.g., Chapin's survey [1996]).

This survey will help in understanding the potential of process migration. It attempts to demonstrate how and why migration may be widely deployed. We assume that the reader has a general knowledge of operating systems.

Organization of the Paper

The paper is organized as follows. Section 2 provides background on process migration. Section 3 describes the process migration by surveying its main characteristics: complexity, performance, transparency, fault resilience, scalability and heterogeneity. Section 4 classifies various implementations of process migration mechanisms and then describes a couple of representatives for each class. Section 5 describes four case studies of process migration in more detail. In Section 6 we compare the process migration implementations presented earlier. In Section 7 we discuss why we believe that process migration has not caught on so far. In the last section we summarize the paper and describe opportunities for further research.

2. BACKGROUND

This section gives some background on process migration by providing an overview of process migration terminology, target architectures, goals, application taxonomy, migration algorithms, system requirements, load information management, distributed scheduling, and alternatives to migration.

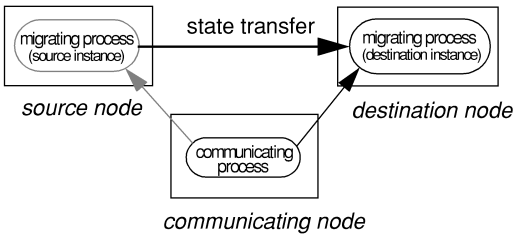

Fig. 1. High Level View of Process Migration. Process migration consists of extracting the state of the process on the source node, transferring it to the destination node where a new instance of the process is created, and updating the connections with other processes on communicating nodes.

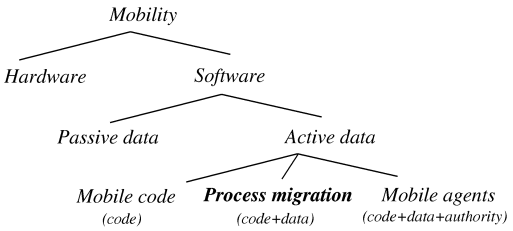

Fig. 2. Taxonomy of Mobility.

2.1. Terminology

A *process* is a key concept in operating systems [Tanenbaum, 1992]. It consists of data, a stack, register contents, and the state specific to the underlying Operating System (OS), such as parameters related to process, memory, and file management. A process can have one or more threads of control. Threads, also called lightweight processes, consist of their own stack and register contents, but share a process's address space and some of the operating-system-specific state, such as signals. The *task* concept was introduced as a generalization of the process concept, whereby a process is decoupled into a task and a number of threads. A traditional process is represented by a task with one thread of control.

Process migration is the act of transferring a process between two machines (the *source* and the *destination* node) during its execution. Some architectures also define a *host* or *home* node, which is the node where the process logically runs. A high-level view of process migration is shown in Figure 1. The transferred state includes the process's address space, execution point (register contents), communication state (e.g., open files and message channels) and other operating system dependent state. *Task migration* represents transferring a task between two machines during execution of its threads.

During migration, two instances of the migrating process exist: the *source instance* is the original process, and the *destination instance* is the new process created on the destination node. After migration, the destination instance becomes a *migrated process*. In systems with a home node, a process that is running on other machines may be called a *remote* process (from the perspective of the home node) or a *foreign* process (from the perspective of the hosting node).

Remote invocation is the creation of a process on a remote node. Remote invocation is usually a less "expensive" operation than process migration. Although the operation can involve the transfer of some state, such as code or open files, the contents of the address space need not be transferred.

Generally speaking, mobility can be classified into hardware and software mobility, as described in Figure 2. Hardware mobility deals with mobile computing, such as with limitations on the connectivity of mobile computers and mobile IP (see [Milojičić et al., 1999] for more details). A few techniques in mobile computing have an analogy in software mobility, such as security, locating, naming, and communication forwarding. Software mobility can be classified into the mobility of passive data and active data. Passive data represents traditional means of transferring data between computers; it has been employed ever since the first two computers were connected. Active data can be further classified into mobile code, process migration and mobile agents. These three classes represent incremental evolution of state transfer. Mobile code, such as Java applets, transfers only code between nodes. Process migration, which is the main theme of this paper, deals primarily with code and data transfer. It also

deals with the transfer of authority, for instance access to a shared file system, but in a limited way: authority is under the control of a single administrative domain. Finally, mobile agents transfer code, data, and especially authority to act on the owner's behalf on a wide scale, such as within the entire Internet.

2.2. Target Architectures

Process migration research started with the appearance of distributed processing among multiple processors. Process migration introduces opportunities for sharing processing power and other resources, such as memory and communication channels. It is addressed in early multiprocessor systems [Stone, 1978; Bokhari, 1979]. Current multiprocessor systems, especially symmetric multiprocessors, are scheduled using traditional scheduling methods. They are not used as an environment for process migration research.

Process migration in NUMA (Non-Uniform Memory Access) multiprocessor architectures is still an active area of research [Gait, 1990; Squillante and Nelson, 1991; Vaswani and Zahorjan, 1991; Nelson and Squillante, 1995]. The NUMA architectures have a different access time to the memory of the local processor, compared to the memory of a remote processor, or to a global memory. The access time to the memory of a remote processor can be variable, depending on the type of interconnect and the distance to the remote processor. Migration in NUMA architectures is heavily dependent on the memory footprint that processes have, both in memory and in caches. Recent research on virtual machines on scalable shared memory multiprocessors [Bugnion, et al., 1997] represents another potential for migration. Migration of whole virtual machines between processors of a multiprocessor abstracts away most of the complexities of operating systems, reducing the migrateable state only to memory and to state contained in a virtual monitor [Teodosiu, 2000]. Therefore, migration is easier to implement if there is a notion of a virtual machine.

Massively Parallel Processors (MPP) are another type of architecture used for migration research [Tritscher and Bemmerl, 1992; Zajcew et al., 1993]. MPP machines have a large number of processors that are usually shared between multiple users by providing each of them with a subset, or partition, of the processors. After a user relinquishes a partition, it can be reused by another user. MPP computers are typically of a NORMA (NO Remote Memory Access) type, i.e., there is no remote memory access. In that respect they are similar to network clusters, except they have a much faster interconnect. Migration represents a convenient tool to achieve repartitioning. Since MPP machines have a large number of processors, the probability of failure is also larger. Migrating a running process from a partially failed node, for example after a bank of memory unrelated to the process fails, allows the process to continue running safely. MPP machines also use migration for load distribution, such as the *psched* daemon on Cray T3E, or Loadleveler on IBM SP2 machines.

Since its inception, a Local Area Network (LAN) of computers has been the most frequently used architecture for process migration. The bulk of the systems described in this paper, including all of the case studies, are implemented on LANs. Systems such as NOW [Anderson et al., 1995] or Solaris [Khalidi et al., 1996] have recently investigated process migration using clusters of workstations on LANs. It was observed that at any point in time many autonomous workstations on a LAN are unused, offering potential for other users based on process migration [Mutka and Livny, 1987]. There is, however, a sociological aspect to the autonomous workstation model. Users are not willing to share their computers with others if this means affecting their own performance [Douglis and Ousterhout, 1991]. The priority of the incoming processes (processing, VM, IPC priorities) may be reduced in order to allow for minimal impact on the workstation's owner [Douglis and Ousterhout, 1991; Krueger and Chawla, 1991].

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.