CHAPTER 1

Introduction to Process Migration & Load Balancing

1.1 Introduction
1.2 The Concept of Load Balancing
1.3 Process Migration Basics
1.4 Process Migration Mechanism
1.5 Process Migration Challenges
1.6 Objectives of the Proposed Work
1.7 Problem Description
1.8 The OptiMigrator (OM)
1.9 Concluding Remarks
CHAPTER – 1

Introduction to Process Migration and Load Balancing

1.1 Introduction

The invention of distributed systems has directed the era of computing towards a new horizon. Distributed systems have brought the terminologies such as process migration and load balancing at newer heights. The work described in this dissertation is primarily associated with the terminologies such as process migration, load balancing and distributed systems. Therefore, this chapter builds an introductory platform and discusses various issues which are associated with these terminologies.

In the beginning of this chapter, we introduce the concept of distributed systems in brief. In the subsequent parts of this chapter, the terminologies such as process migration and load balancing have been discussed. The chapter also describes various classifications and related issues of the process migration technique. Then, the chapter describes in brief the process migration mechanism. The challenges faced in the area of process migration have been described thereafter. The research problem and problem description have been highlighted the end of the chapter. Finally, the chapter concludes with an introduction to the “OptiMigrator (OM)”.

Distributed Computing Systems

Use of distributed computing systems is increasing day by day as a consequence of the availability of powerful hardware at lower cost and innovations in networking and communication technologies. Some of the primary benefits of distributed systems include the strength of resource sharing to provide the users with a wealthy collection of resources which are usually spread across the member workstations.
Chapter 1: Introduction to Process Migration & Load Balancing

Optimized Process Migration for Load Balancing in Distributed Systems

of a particular network and the opportunity on load balancing of network’s workstations. The last decade has observed the development of tightly-coupled parallel multi-processor systems which have been trendy in high performance computing environments [J. M. Smith, 01]. Moreover, it is generally observed that, the hardware has been upgrading more quickly than the supporting system software and the operating systems. There exist number of drawbacks with such parallel computing systems like a limited support or lacking of fault tolerance and limited or negligible support in quick hardware upgrading as the addition of new hardware components to the parallel systems do require modifications in underlying system software and operating system policies and this causes the system reboots and therefore causes discontinuity in the availability of the system. In the long run, a number of limitations may be faced by such configurations such as difficulty in upgradation e.g. (a) addition of more processors on the board which is already having two or more processors, (b) the whole board is to be removed from the computing system if one processor crashes or requires maintenance causing the whole system reboot. By connecting simpler, smaller and cheaper processors and other components, a computing system may be assembled with the same processing capacity as that of one utilizing a highly complex, large and powerful processor but for a fraction of the price [M. Liu, 09].

On the other side, the advancements in microelectronic technology and communication technology has resulted in the availability of fast, inexpensive processors and cost-effective “loosely-coupled systems” also known as “Distributed systems”; in which software components located at networked nodes communicate and cooperate their actions to achieve the features like – information sharing, resource sharing, better flexibility, higher throughput and higher reliability. It is practically possible to either insert additional cost-effective workstations to or remove existing workstations from the distributed
systems. It means that the limitations of the parallel computing architectures do not exist with distributed systems [J. D. Smith, 03], [M. Kozuch, 02].

[A. S. Tanenbaum, 07] defines a distributed system as a “collection of independent computers that appear to the users of the system as a single computer”. Moreover, according to [P. K. Sinha, 04], in the loosely-coupled systems i.e. the networked workstations (Figure 1.1), each of the workstation has its own processor(s), processes and other resources such as the address spaces; and the software components located at networked workstations communicate and co-operate their actions. Some of the primary important benefits of the distributed systems include the strength of resource sharing to provide the users with a wealthy collection of resources which are usually spread across the member workstations of a particular network and the load balancing of network’s workstations.

![Figure 1.1 The abstract view of distributed systems](image)

Furthermore, according to [M. Liu, 09], a computing technique practiced with the distributed systems enables the noteworthy benefits such as- scalability, resource sharing, fault tolerance, affordability of computers and availability of network.
Chapter 1: Introduction to Process Migration & Load Balancing

1.2 The Concept of Load Balancing

Although the total computational capacity of the networked workstations may on paper appear much larger than that of the original uniprocessor computing system, often, workstation-users will repeatedly execute their corresponding applications on their own systems, which gradually become either overloaded or entirely idle; it is frequently observed that, in a computing environment with a number of hosts connected by networks, there is high probability that some of the workstations are heavily loaded, while others are almost idle. This in effect causes utilization of only a fraction of the overall network’s theoretical power [S. Malik, 00].

Thus, the discussion made so far indicates that the performance gains may be achieved by transferring processes from the currently heavily loaded workstations to either the idle workstations or the lightly-loaded workstations. Such a technique of sharing of computing power among the processes across the networked workstations for the objective of performance improvement of the overall distributed system by redirecting the workload among the available workstations is known as “Load sharing” or “Load balancing” [A. S. Tanenbaum, 94]. The term load sharing indicates the process of ensuring that no workstation remains idle while work is under queue at other workstations in the network. Another term, load balancing, describes a system in which efforts are made to equalize the load at each workstation; this varies in the more general load-distribution implied by load sharing. According to [M. Nuttall, 97], in practice, the theoretical distinction is not very important; as most of the algorithms fall somewhere between the two terms, do more than ensuring that no host remains idle. Some of the benefits of the load balancing technique in the context of distributed systems have been described below.
Benefits of load balancing

1. Higher throughput
A major weakness of networked workstations is that there is a lot of distributed processing power that goes unused. Instead of queuing more and more processes thereby causing the overloading of a few workstations, it is wise to divert/redirect some of these processes to either idle workstations or lightly-loaded workstations. Such type of process-redirection may lead the migrant processes towards early completion as they will have to wait less to avail the resources on the idle or lightly-loaded workstations. Thus, the overall system will attain a greater throughput.

2. Maximum Resource Utilization and Return on Investment
As mentioned above, the processing power and other resources remain unutilized on the idle workstations or the lightly-loaded workstations, thereby resulting in the under-utilization or wastage of such resources which is against the objective of the ‘Return on Investment’. But, the ‘Return on Investment’ may be improved by bringing some of the processes from the heavily-loaded workstations to the workstations where the computing power goes unused (i.e. the workstations generating less Return on Investment); and thereby causing the maximum utilization of such computing power and consequently causing increase in the return on investment.

The technique of load sharing or load balancing has been studied using a number of different approaches for long time, which can be roughly partitioned into two broad categories: (i) static load balancing and (ii) dynamic load balancing [S. Sharma, 08].

In static load balancing, workload transfer decisions are made probabilistically without taking into consideration the current actual state of the system [P. K. Sinha, 04]. Static load balancing is simple and effective when the workload can be sufficiently well characterized
beforehand, but it may fail to adjust the fluctuations in the system load. In distinction, dynamic load balancing attempts to balance the system’s load dynamically as jobs arrive. We have proposed the dynamic load balancing algorithms in this work.

We suggest algorithms to provide significant reduction in the load of heavily used workstations in chapter 6. We propose algorithms for two techniques: server-managed load balancing to serve periodic load-balancing policy and client-initiated load-balancing to serve partial fault-tolerance and the non-periodic load-balancing policy.

### 1.3 Process Migration Basics

The issue of efficient resource utilization in the network of workstations can be addressed by distributing load of a workstation among other workstations in the network using process migration. The type of load balancing performed by us in this work is carried out using the technique of process migration. The technique of process migration can result into efficient overall system utilization by making use of the idle workstations [J. Basney, 00].

The main objective of this work is to efficiently send a process from one workstation to another workstation in the network to achieve load-balancing in a network of workstations. The figure 1.2 represents the phenomenon of shifting a process from one workstation to another one which is called process migration. This proposed work is intended to design algorithms and to implement the proposed algorithms for optimized and featured process migration to support load-balancing in the network of workstations.

In fact, there are two main types of process migration techniques: the non-preemptive process migration and the preemptive process migration.
Sometimes they are also known as static and dynamic process migration respectively [D. S. Milojicic, 00].

The term preemptive (dynamic) process migration refers to the practice of relocation of an already executing process from its origin workstation to some another workstation, followed by its continued execution on the destination workstation. It is different from the concept behind the ‘rsh facility’ available in Linux which is associated with the selection of another workstation for creation and execution of a newly submitted process on the destination workstation – also known as non-preemptive (static) process migration. The static process migration is applicable to the newly submitted processes (i.e. the processes which have been only created and have not started execution yet); though it is not preferred for the partially executed process as the process has to re-execute from the beginning after migrating on the destination workstation.

Preemptive process migration is considerably more difficult to employ than the non-preemptive process migration, since it involves migration of a process in a state of execution. Preemptive process migration minimally requires suspending the executing process on origin workstation, extracting its state, relocating the extracted state to the destination workstation, reconstructing the state on the destination workstation and resuming the process’s execution on the destination workstation [N. A. Joshi, 11].

Both of these techniques have merits and demerits:

(i) There is no need of re-executing the portion of the process which has already been executed on the source workstation. The dynamic process migration mechanism does not re-execute the process on destination workstation from the beginning, whereas the static process migration mechanism re-executes the process on the destination workstation; and
The other types in which the process migration mechanism can be classified are weak process migration and strong process migration. The weak process migration mechanism refers to migration of the program executable only (and not the data segment), i.e. only the code segment gets migrated to the destination workstation. The Java Applet is an example of weak migration. Main characteristic of weak-migration is that after migrating to the destination workstation, the migrated program starts its execution from the beginning. Simplicity
is the advantage of weak process migration mechanism as it requires only the binary executable program’s availability on the destination workstation. So the migration mechanism does not have to worry about migration of the other memory segments except the code segments occupied by the process.

On the contrary, the strong process migration requires migration of all the memory segments occupied by the process. The benefit of strong process migration is that the remigration of the process which was migrated earlier is possible, meaning that a running process can be stopped, subsequently transferred to another workstation, and then resume execution where it left off. Noticeably, strong process migration is much more powerful than weak process migration, but also much harder to implement [K. Noguchi, 07], [I. Satoh, 01].

Depending upon the requirements and the goal behind process migration, the process migration mechanism may be further classified as passive migration and self-governing migration or sometimes a combination of these two types. In case of passive migration mechanism, the process migration is controlled by some external supervisor, where, the external supervisor could be some load-balancing software or the system administrator. This can be most suitable for the purposes like fault tolerance and dynamic load balancing.

**User-level process migration and kernel-level process migration**

The user-level process migration mechanisms carry out the process migration without altering the underlying operating system’s kernel. Due to this reason, the mechanisms are not as complex as kernel-level process migration mechanisms. The major difficulty involved in user-level process migration implementation is, they cannot access the features which are available to only kernel-layer and hidden from
the user-layer; therefore, they cannot deal with migration of certain process-state information of the process, thereby limiting the applicability of the process migration to all processes.

The kernel-level process migration mechanisms carry out the process migration by altering the kernel; the mechanism may modify the existing features to make them suitable for process migration and add new capabilities to the existing kernel. This increases efficiency, and makes more kernel-level features available to the process migration mechanism. The major weak point of the kernel-level (i.e. system-level) process migration is that such mechanisms are applicable to the specific kernels for which they have been implemented (as the operating system’s kernel implementation varies from version to version across the upgrading hardware architectures); thereby tumbling the process migration’s availability and interoperability across different kernels and architectures [J. C. Sancho, 05].

**Homogeneous process migration and Heterogeneous process migration**

The homogeneous process migration mechanism is applicable to the homogeneous environment where all the workstations have the same operating system. However, in some of mixtures of networks the process migration is desired among the workstations which are running different operating systems. In such cases, heterogeneous process migration mechanism becomes mandatory.

The heterogeneous process migration mechanism performs the process migration irrespective of the differences in the underlying operating systems being run by the workstations; although it is more difficult to implement since it must consider the workstations and the features and structures specific to the different operating systems [A. J. Ferrari, 00]. For example, (i) the operating system's 'system call’
facility (which is supported by some operating system A) might be either supported using different mechanism or not supported at all by the destination workstation’s operating system B; (ii) it may not be easy or very difficult to implement the process migration mechanism in case of unavailability of the implementation information or documentation of a particular operating system (such as closed-source operating systems); (iii) different type of data structures and data types might be implemented differently on different operating systems, e.g. the size of the integer data type, even some of the data types which are present in one operating system might not be available with the same name or totally absent on some other operating system [N. Meyer, 03], [A. J. Ferrari, 98].

**Benefits of Process migration**

1. **Load sharing**
   Process migration and load sharing can be applied to a system of interconnected computers to distribute the workload of heavily used workstations among the idle workstations or the lightly-loaded workstations. This leads to availability and utilization of a plenty of distributed processing power that usually is unused and wasted. Instead of overloading a few workstations with a lot of tasks, it makes sense to allocate some of these tasks to under-used workstations to improve performance. This results in tasks to be completed quickly and possibly in parallel, without affecting the performance of the other workstations in the system [N. A. Joshi, 12], [N. G. Shivaratri, 92].

2. **Increased fault tolerance**
   Process migration may improve fault tolerance of a system. For example, the process could be migrated from a partially-failed workstation to some other healthy workstation in the network to preserve continuity and reliability in the availability and execution of the process.
Consider long-running processes whose workstations are to be disconnected from the network or powered-off later after some time before the particular process reaches to its completion. In such circumstances, the specific process of interest which is required to be available even after either disconnecting the workstation from network or powering-off the workstation could be migrated to some other running workstation. Thus, process migration makes the process available even after machine failure, disconnection or power-off. Consider certain category of faults which can be notified in advance, e.g. the operating system may notify a process by sending a message that the system is about to shut down. In this case, the process may be migrated to some other running workstation to continue its execution [N. Vasudevan, 06].

Moreover, the process state checkpointing systems perform incremental checkpointing to the secondary memory storage device; which can be useful in case of system or process failure [J. C. Sancho, 05].

3. Better resource utilization
Certain workstations may have better capabilities (e.g. more amount of primary memory or higher processing power) which can be utilized by migrating the process to the workstation which possesses such capabilities. In this case, initially the process can be started on the originating workstation and then migrated to a more powerful machine when it becomes available, thereby availing the better capabilities to the migrated process [N. A. Joshi, 12].

4. Bringing the process near data
Another area of applications where process migration could be beneficial is – it could be used to migrate a distant process closer to data that it is processing; thereby ensuring that it spends most of its time in doing a useful work; and not spending most of its time
performing communication between workstations for sake of accessing the required data [D. S. Milojicic, 00].

There are several other benefits of process migration as the process migration mechanism involves the activity of process checkpointing; which itself has certain advantages:

5. **Crash recovery and rollback transaction**
   Process can be made to return to a previously checkpointed state. This can become beneficial to long-running applications especially to scientific computing applications. A specific type of checkpointing, called incremental checkpointing, may help to reduce the checkpointing overhead.

6. **System administration**
   The facility of checkpointing and resuming the checkpointed image later (on same workstation or on different one), can be of great advantage to system administrators. They can checkpoint processes before shutting down a workstation of the network or an isolated computer system and restart them after the workstation or computer system is up again.

7. **Importance of process migration**
   When applications execute on networked workstations, the practice of process migration mechanism is an advantageous feature for load sharing; although these days the cluster systems offer a means to set aside and employ dedicated workstations so as to execute their specific applications. In such conditions, the modern compiler, the programmer, the application manager, or the system administrator may be able to share the data and computation at compile time among pre-decided well-qualified workstations thereby not necessitating process migration to achieve the features such as access locality and load sharing.
In general, the amount of overhead related to process migration is much larger than generating and sending messages in the message passing systems. Therefore, the following questions are likely to be raised- “When does the benefit prevail over the overhead of using process migration?” Or “What is the importance of process migration?” [K. Noguchi, 07] considers the uncertainty at compile time as a major issue in such queries.

This uncertainty could exist either in the quantity of execution or in the dimension and weight of the data structures being used or in the workload of the networked workstations. In absence of availability of such pre-required information, it may become difficult or nearly impossible for a programmer or a compiler to carry out the task of load distribution in terms of data and programs distribution across the networked workstations. The applications in the domain of fluid dynamics, computer geometry and graphics are the examples.

Consider, one more example of the applications which run in the GRID environment [I. Foster, 99]. The workstation’s speed and network delays among workstations are uncertain and unstable since GRID makes use of network resources. For this purpose, the distribution of data may be preferred at runtime, thereby requiring process migration.

As well, it is difficult to determine the incremental allocation pattern and the irregular access pattern in the applications which use the data structures like linked lists, trees, directed acyclic graphs, or any combination of them; thereby preventing the compiler and the programmer to know the correct size and shape of data structures at compile time and thereby requiring the availability of the process migration mechanism.
Although process migration is such a beneficial mechanism, it is still primarily a research area and has not reached to the production phase yet. Some of the reasons for this are:

- **Lack of support from operating system**
  The open source operating systems like LINUX and the parent operating system UNIX are not designed for the purpose of process-state checkpointing and migration mechanisms. Although, the LINUX operating system is available with an open-source license, it is very difficult to add the facilities like load balancing, process-state checkpointing and process migration in a straightforward way without applying significant modifications to the already vastly used operating system’s kernel. Therefore, lack of operating system’s features for supporting process migration has steered research towards this area.

- **Lack of commercial demand**
  [H. Zhong, 01] claims that process checkpointing is primarily used for high performance distributed computing systems; there is not yet a large demand in the commercial computing market.

- **Transparency and reliability**
  In order to be available for general use, the features like load balancing, process-state checkpointing and process migration must be transparent and reliable. These all features are still research-problems, since the available process migration mechanisms are not considered transparent and reliable for general purpose processes.

### 1.4 Process Migration Mechanism

As mentioned earlier, the dynamic process migration mechanism initially involves the procedure of determining the current state of the
victim process (i.e. the process which is to be migrated). This procedure is known as process state checkpointing or process checkpointing. Therefore, the desired process which is to be migrated is checkpointed first on the originating workstation and the generated checkpointed process image is migrated to destination workstation where the process is resumed from the execution point where it was stopped on the originating workstation.

The process checkpointing mechanism involves the steps of determining the current state of the desired process; the state of the process consists of mainly two sub-state components: static state and dynamic state.

The static state of the running process corresponds to the binary program image which is executed by the running process. This generally corresponds to the binary executable files or library files. The library files can be linked either statically or dynamically. In short, the static state of a particular process remains same for all the n number of checkpointing attempts for a process.

On the other side, the dynamic process state can be summarized in brief as a collection of run-time information of a process [D. P. Bovet, 05] like –

- **Process credentials**: such as process-id and user-id;
- **The memory regions**: (both the physical memory regions and the logical memory regions) represents the address space of the process. It is portioned into number of segments such as the code segment, data segment, stack segment and the heap segment occupied by the process including their location in memory and current state;
- **Context Data**: the current-values held in the CPU registers for the process such as the data registers, the program counter, the
instruction register and such other registers; which may vary in numbers and types, depending on the workstation architecture;

- **System calls**: the checking of whether the process is executing any system call or not, and if yes, then the system call details such as system call number;
- **Files**: the set of all types of files being used by the process;
- **Signals**: the set of pending signals to be processed by the process;
- **Accounting information**: such as total execution time elapsed, total waiting time elapsed, time limits;
- **Current execution-state**: value of the process such as running state, blocked state, ready state, terminated state;

and other such dynamic details. In short, the dynamic state of the process is of varying nature in real as it continuously keeps on changing from time to time while the process is under execution. Once the process checkpointing is finished, the resultant binary image is migrated to the destination workstation, where it is made to resume further its execution from the context where it was checkpointed on the originating workstation.

### 1.5 Process Migration Challenges

A number of basic problems exist in synthesizing a process migration scheme. The following are some of the challenges to be dealt with:

(1) Process state capturing & transfer: How to collect the information about current state of the process to be migrated? (Such process information to be collected involves - object code of the process, memory regions occupied by the process, contents of CPU registers at the time of migration, files being used by the process and environment variables used by the process.)

How to transfer this information to the target host? Ideally, the complete process state information is to be encapsulated and
shipped to another workstation in the network.

(2) System calls: What should happen to the system call while the process was executing inside it during process checkpointing? After migration should the system call execution be skipped, or restarted or should it be continued from the place where it was checkpointed? In fact, the migrated process should resume the system call’s execution on the destination workstation.

(3) Name spaces: what should happen to original process-id after migration? Should its value remain intact or should it be reshaped to target host environment?

(4) File system: How does the underlying file-system support the migration? Are files assumed to be accessible from any point in the network? A transparent file-system is required.

(5) Scheduling: After migration, when the migrated process should execute? Whether the migrated process should be given lower priority or should it be treated normally on target host? What should happen with the preemptive/non-preemptive process scheduling? For our work, we assume that the migrated process will have normal priority.

1.6 Objectives of the Proposed Work

The process migration and the load balancing work has been described with the following objectives in this thesis –

- To design and develop a dynamic process-migration software (ProcessMigrator) for the Linux operating system.
- To design and implement a load information management package (LoadPortal) for the Linux operating system.
To design and implement a load-balancing package (LoadBalancer) for the Linux operating system with the help of Process-migration package and the Load-calculation package.

### 1.7 Problem Description

The work described in the thesis deals with development of the OptiMigrator to perform checkpointing of a process on source Linux workstation, transferring the checkpointed process state information to the destination workstation, restoring the process state and resuming the process execution on the destination Linux workstation. The development of load information management software module (LoadPortal) is intended to determine and communicate the workload information of the Linux-based workstations to the central load management server.

The suggested load-balancing software module (LoadBalancer) coordinates the above mentioned two packages ProcessMigrator and LoadPortal. It is responsible to carry out process migration decisions and operations with the help of the above listed two packages. We have implemented the solution for the Fedora Linux operating system due to following reasons:

1. The Linux operating system enables researchers to work with both of the operating system layers: the user-layer and the kernel-layer.
2. Solution suggested for the Fedora Linux can be used for other Linux flavors with no or minor modifications.
3. The Fedora is vastly used flavor among the existing flavors of Linux operating system.
4. Linux is the most preferred operating system among contemporary open source operating system platforms.
1.8 The OptiMigrator (OM)

As an outcome of the development of process migration and load balancing algorithms and their implementation, we present a functioning software package named “OptiMigrator (OM)” in this thesis.

The OptiMigrator (OM) software package incorporates various functionalities such as server-managed load balancing, client-initiated load balancing and process migration. It not only includes these functionalities but also meets certain design goals. Some of its functionalities have been described in the seventh chapter.

1.9 Concluding Remarks

An attempt is made in this chapter to introduce the concepts of process migration and load balancing. This chapter justifies the significance of the process migration mechanism by describing some of its benefits. Various classifications of the process migration mechanism such as – preemptive and non-preemptive process migration, user-level and kernel-level process migration, strong and weak process migration, homogeneous and heterogeneous process migration – have been described in this chapter. The challenges appearing in the area of process migration have been described in this chapter. The overall process migration mechanism also has been summarized in this chapter.

Moreover, the chapter explains importance of the load balancing technique by listing its usefulness and benefits in the distributed system. The chapter also notes the resemblance between the two related terms load balancing and load sharing.
The chapter finally formulates the research problem statement for which various solutions have been designed, implemented and presented in this thesis. The chapter concludes with an introduction to our solution software ‘OptiMigrator (OM)’, whose design and implementation has been presented in this thesis.