CHAPTER 2

Research Contributions in the Field of Process Migration

2.1 Introduction
2.2 User-level Process Migration
2.3 Kernel-level Process Migration
2.4 Significant Remarks on Literature Survey
2.5 The Work
2.6 Concluding Remarks
CHAPTER – 2

Research Contributions in the Field of Process Migration

2.1 Introduction

Researchers have experimented many ways to automate load balancing with the help of various process migration mechanisms. This chapter observes the work done in past in the area of process state checkpointing and process migration in the computing environments. Some of the motivating research contributions in the area of process state checkpointing and process migration have been presented in this chapter. The later part of this chapter describes the work proposed in this dissertation.

2.2 User-level Process Migration

This section presents an overview of the significant research contributions in the field of user-level process migration.

2.2.1 Condor

The Condor system, developed by the Condor Team, Computer Science Department at the University of Wisconsin, is a scheduling system for an environment of UNIX workstations which are networked through a high speed LAN and provides the facility for checkpointing and migration [M. Livny, 00]. The central idea of Condor’s job scheduling system is to carry out a job to vacate non-idle workstation and migrate to an idle workstation. To achieve this, the Condor system checkpoints the process’s state, and it happens just before the process vacates the origin workstation; this checkpoint is then used as a starting point for the process after migration [D. Thain, 05]. Condor gives a program the ability to checkpoint itself by providing a checkpointing library which contains a signal handler for
SIGTSTP, the Condor checkpoint signal. It provides its own set of library functions which perform the required task of extracting the state of the process and restarting it on the remote workstation [J. Meehean, 05], [M. Litzkow, 97].

Although it has been found useful for computation-intensive long running jobs, the program’s code must be re-linked with the Condor checkpointing libraries to make it a suitable candidate to get the benefits of checkpointing and process migration. Thus, existing applications whose source code is not available cannot be re-linked and therefore cannot be checkpointed and hence cannot be migrated; which severely limits its usefulness.

One more serious limitation with Condor is that it does not support alarms, timers and sleeping, e.g. the `sleep()`, `alarm()` and `getitimer()` system call are not supported with Condor [Condor]. Moreover, Condor’s manual says: “Any system call performed on the destination workstation is sent over the network, back to the process originating workstation which actually performs the system call on the submit machine, and the result is sent back over the network to the migrated process on the destination workstation.”; this requires availability of the victim process (in the form of shadow process) on the originating workstation even after completion of process migration.

### 2.2.2 UTOPIA

The UTOPIA system developed at the Computer Systems Research Institute, Toronto – Canada is a load-sharing facility. An additional advantage of UTOPIA is that it supports heterogeneous process migration. The system consists of Load Information Manager (LIM) which is used to measure and exchange load information. There is a Remote Execution Server (RES) which provides mechanisms for transparent remote execution. There is also a Load Sharing Library (LSLIB) which is used for developing load sharing applications.
Such applications are of two types: one that directly uses the LSLIB interface and others which use such applications as their execution context. The other applications do not require re-linking for achieving remote execution, but the child process created by them on the remote node cannot be remotely executed. Therefore, the user code must be re-linked with the checkpointing library to make it a candidate of migration. Thus, current applications whose code is unavailable, they cannot be linked again and so they cannot be checkpointed and therefore cannot be transferred to some remote workstation; which limits its usefulness.

Moreover, some applications have to be rewritten (and therefore recompiled) to utilize the advantage of remote execution. Other applications which are not rewritten get only limited advantage as their child process cannot be remotely executed. Thus, the utility of the system is decreased as the opportunity for load sharing is lost across the network.

The serious limitation of the UTOPIA is that it supports only non-preemptive process migration (i.e. static process migration), means the process cannot be migrated once it gets started on the originating workstation [S. Zhou, 93].

### 2.2.3 MPVM

The MPVM (Migratable PVM) utility, developed at the Oregon Graduate Institute of Science and Technology, is a transparent migration version of PVM. The PVM (Parallel Virtual Machine) is a software system that allows a network of heterogeneous workstations to be viewed as a single virtual machine with distributed shared memory. In the original implementation of PVM, processes were allocated to processors statically. Migration allowed the usage of idle workstations in a dynamic manner. MPVM is implemented at the user layer.
Even though MPVM strives to achieve transparency, it has a serious drawback due to the fact that process state information like process ids that are known and used by the application process change upon migration [J. Casas, 95].

### 2.2.4 REXEC

REXEC, developed at Computer Science Division, University of California is a decentralized, secure remote execution facility. It provides high availability, scalability, transparent remote execution, dynamic cluster configuration, decoupled node discovery and selection, a well-defined failure and cleanup model, parallel and distributed program support, and strong authentication and encryption.

However, REXEC does not support process migration and is limited for the 2.2.5 version of the Linux kernel. It also lacks in re-linking with a C library of remote execution related functions [B. Chun, 00].

### 2.2.5 Research Contribution at University of Virginia

Ferrari, Grimshaw and Chapin also implemented process state capture and recovery through process introspection mechanism for the UNIX 4.3 BSD running on DEC VAX machines. They modified the ‘C shell’ to remotely execute the selected types of given commands. A Load Information Manager (LIM) component constantly monitors system loads and performs job placement. A Load Balancing Manager (LBM) module exists on each host to support remote execution.

Although the suggested work supports the objective of load sharing, it supports only static migration; it does not fulfill dynamic process migration, i.e. the process under execution cannot be checkpointed and therefore cannot be migrated to the destination workstation [A. J. Ferrari, 00].
2.2.6 rsh
The UNIX rsh facility supports the execution of a subset of shell commands on a remote machine. One of the rsh options enables the execution of a program on the remote machine [LINUX].

The UNIX rsh facility provides only static process migration. Processes already under execution cannot be checkpointed and therefore cannot be migrated to some remote workstation. Another problem with rsh is that it requires the user to pick a suitable destination workstation for offloading.

2.2.7 Libckpt
The libckpt tool is a library implementation, which was developed to carry out process user-level checkpoint-restart for the UNIX processes [J. S. Plank, 95]. (Library implementations are the set of libraries or header files which are to be used or included by the applications who desire self-checkpointing. Such applications must be compiled and linked with reference to these checkpointing libraries.)

The libckpt technique possesses some unique optimizations to reduce the size of checkpoint-image files. With libckpt, those memory pages which have not been used are not written to the checkpoint image file. On the contrary, the modified pages are marked as dirty; so during the checkpoint mechanism follow-up, the libckpt writes only the dirty pages to the checkpoint image file. This reduces the time required to checkpoint the memory regions occupied by the victim process; hence it reduces not only the memory requirement of the libckpt library implementation but also the smaller sized checkpoint-image file gets migrated faster to the destination workstation. This is how the libckpt optimizes the checkpointing mechanism. It also facilitates a feature of synchronous checkpointing, which allows an application to carry out checkpointing operation at particular time during the execution of a process.
The serious limitation with the libckpt mechanism is that, the libckpt facility imposes a number of constraints on the application which is to be checkpointed:

- It requires significant changes to the application’s source code.
- It requires renaming of the application’s main routine file.
- Moreover, changes made to the source code must be transformed into the object code also, i.e. the application code is required to be not only re-compiled but also statically linked to the libckpt library; this prevents the libckpt from checkpointing applications whose source files are unavailable, such as commercial binary files.

Another considerable limitation of the libckpt library is that during the process-restart phase from the checkpointed-image file, it cannot restore segments which are mapped in by the application through mmap.

2.2.8 Libtckpt

The Libtckpt utility is also a library implementation, which additionally performs user-level checkpointing of multithreaded applications using either Linux threads or Solaris threads [W. R. Dieter, 01]. To carry out checkpointing of multithreaded applications, the Libtckpt adds one more thread to the application; the thread is responsible to carry out checkpointing of the application which is to be checkpointed and synchronize the threads of application.

Like the Libckpt library (discussed above), the Libtckpt also necessitates changes to the application’s source code such as the application must not only include a specific Libtckpt header file but also invoke its initialization routines; this may always not be conceivable for some applications as the source code of some third party legacy applications and some commercial applications may not be available.
2.2.9 CoCheck

The CoCheck (Consistent Checkpointing) utility is a user-level checkpointing utility which was developed by George Stellner. The Cocheck utility checkpoints parallel application which is running on network of workstations. Once checkpointed, these applications can be migrated to achieve a more balanced load across a network [G. Stellner, 96]. After completion of process migration, CoCheck does not leave any residual dependencies (or the shadow process) on the originating workstation of the process. However, like other previously discussed user-level checkpointing libraries, the CoCheck utility also does require modification of source code, recompilation and relinking of the application source code.

2.2.10 CKPT

The CKPT (Process Checkpoint Library) utility was developed by Victor C. Zandy. It is a set of libraries and programs which are responsible to carry out user-level process checkpointing. Main advantage of the CKPT utility over other user-level checkpointing systems is that it does not require re-compilation of the already compiled and linked applications which are to be migrated. It is also possible to inject CKPT into running processes. The CKPT utility inserts code into the process; this enables the process to checkpoint itself. The CKPT redirects the checkpointed-image to the disk file which can be resumed for execution later. It provides a facility to properly checkpoint the address space and the process-environment [V. C. Zandy, 04].

However, the CKPT utility does not support checkpointing and migration of the process identifiers and other process credentials, which can be considered a serious limitation. Moreover, the CKPT usage involves re-linking the process with the CKPT runtime environment.
2.2.11 Esky

The Esky (A slightly portable checkpoint) package developed by David Gibson is a system for checkpointing and resuming processes in the UNIX operating system. It works for LINUX kernel 2.2 [D. Gibson, 99].

However, to checkpoint a process the process must be guided to execute under the control of the Esky software’s monitor, i.e. it must be known in advance by the programmer or the user that the process will be checkpointed and migrated (to achieve load sharing and balancing) before it gets launched, this could be inconvenient and inefficient. Other limitation with Esky is that in order to operate, it periodically receives the SIGALARM signal, so unfortunately this solution is not generous because in many cases signal handlers (such as SIGALARM) interfere with the application to be checkpointed or the resource manager.

2.2.12 Xen

The Xen hypervisor is a layer of software running directly on computer hardware replacing the operating system, thereby allowing the computer hardware to run multiple guest operating systems concurrently. It provides the powerful open source industry standard for virtualization. It offers a powerful, efficient, and secure feature set for virtualization of x86, x86_64, IA64, ARM, and other CPU architectures. It supports a wide range of guest operating systems including Windows, Linux, Solaris, and various versions of the BSD operating systems [P. Barham, 03].

However, the hardware virtualization approach provided by Xen allows checkpointing and restarting of only the entire operating system environment, and it cannot provide checkpointing and restarting of small sets of processes. That leads to higher checkpointing and restart overhead [A. Mirkin, 08].
2.3 Kernel-level Process Migration

This section highlights significant research contributions in the field of kernel-level process migration.

2.3.1 Amoeba

Amoeba is a distributed operating system which was developed at the Vrije Universiteit, Amsterdam. It is designed to take a collection of machines and make them act together as a single integrated system. In general, users are neither aware of the number and location of the processors that run their commands, nor of the number and location of the file servers that store their files. To the casual user, the Amoeba system looks like a single old-fashioned time-sharing system. It provides dynamic load balancing through the means of homogeneous process migration. The kernel of the Amoeba operating system contains comparatively little functionality, with most traditional operating system functions carried out in server processes. The kernel is responsible to accomplish process scheduling, memory management, inter-process communication, and input/output device handling.

A trouble in checkpointing and migrating the process state arises when the process is executing in a system call. In this case, the process includes system call return addresses and system call parameters. During the process of checkpointing the system call’s details, Amoeba approaches two strategies. It either pre-terminates the system call under execution or waits for the system call’s completion. Unfortunately, neither approach is ideal for the purpose of the true process migration. Thus, the Amoeba operating system does not maintain consistency in the operation of checkpointing the system call’s details during the introspection of the process state [C. Steketee, 96].
2.3.2 V Distributed System
The V Distributed System developed at Stanford University is an operating system designed for a cluster of workstations that are connected by a high-speed network. High performance and efficiency are its main principles. Each workstation runs a separate copy of the kernel which cooperates to provide a single system abstraction. This cluster of workstations is called a V domain. It is a message-oriented kernel which provides uniform local and network IPC. The V system provides preemptive process migration among the “pool of processors” i.e. idle workstations.

The main drawback of the V distributed system is that it uses broadcast messages to ensure network-wide unique process identifiers; this may affect scalability of the whole V domain [D. Cheriton, 84], [D. Cheriton, 88].

2.3.3 MOSIX
The MOSIX is a high performance cluster computing system, for the cluster of Pentium based PCs which are connected by high speed LAN. The MOSIX operating system was developed at the Hebrew University of Israel to provide the features like kernel-level process checkpoint-restart and the preemptive process migration mechanisms to achieve the goal of load-balancing. It serves well the feature of resource sharing. It was generated as a result of the number of modifications made to the UNIX operating system [A. Barak, 93].

MOSIX has a limitation that it does not allow system administrator to redirect the checkpointed process image to secondary storage media. One more critical limitation of the MOSIX is that it implements only limited process migration because after process migration, the migrated process still leaves residual dependencies on its originating workstation and therefore depends on the continuous availability of the originating workstation. (As discussed in chapter 1, after
migrating to destination workstation, the migrated process should neither leave any residual dependency nor any associated execution on the originating workstation.) Thus, with MOSIX, the benefit of fault tolerance with process migration is not supported.

### 2.3.4 BLCR

The BLCR (Berkeley Lab Checkpoint/Restart) system is developed at Lawrence Berkeley National Laboratory. It is a dynamically loadable kernel module that supports efficient kernel-level checkpointing on a variety of LINUX systems. It can be used either as a stand-alone system for checkpointing applications on a single workstation, or as a component by a scheduling system or parallel communication library for checkpointing and restarting parallel jobs which are running on multiple workstations. BLCR is kernel-based; provides automatic support for checkpointing the widest range of program features without building distributed operating system functionality or other cluster awareness into the kernel; it provides a user-level callback interface to allow libraries and applications to support behaviors which are not automatically handled in the kernel's own checkpoint logic. It provides a facility to its users to specify whether the process state is saved locally or remotely via the ioctl system call [J. Duell, 03].

One requirement with BLCR process migration is that user applications that may need to be checkpointed must be loaded with the user-level BLCR checkpoint library, which registers a signal handler for the checkpoint signal, enforcing the applications to include the BLCR header files and compile them against the BLCR library. This requires application re-linking and availability of complete source-code of the application for the stage of recompilation.

Moreover, to checkpoint the memory regions occupied by the process, BLCR uses the already existing VMADump module (the Virtual Memory Area Dumper is part of Beowulf project). The process which is
to be checkpointed has to directly invoke the VMADump module to checkpoint itself by writing the process state to a file descriptor. This requires that the process to be checkpointed must be made aware about the working and library of VMADump. This does not allow to perform the passive process migration in order to be beneficial to the load-balancing mechanism. Thus, this approach lacks transparency and flexibility [E. Hendriks, 02].

2.3.5 Sprite
The Sprite software is a network operating system, which was developed at the university of California at Berkeley. In Sprite, the kernel itself provides the process migration facility, thereby featuring the characteristic of transparency. The Sprite network operating system provides homogeneous process migration for the Sun workstations (running the Sprite OS) which are connected through a fast Ethernet. In Sprite, processes are migrated during two occasions - when a resource intensive program is about to start, or during eviction from a remote host [F. Douglass, 90]. In order to support process migration, the Sprite network OS is built up with a high performance sophisticated network file system and caching mechanisms [M. Nelson, 88]. Sprite uses a globally named uniform file system, thus file access is completely transparent across workstations.

Although the Sprite system leaves the least residual dependencies on the originating workstations, it does not eliminate all residual dependencies, because the system forwards some system calls to the originating workstation for execution. This is against the feature of fault tolerance [M. P. Nuttall, 97].

2.3.6 EPCKPT
The Epckpt utility was developed at Federal University of Rio de Janeiro by Eduardo Pinheiro [E. Pinheiro, 98]. It was developed in the form of a Linux kernel patch that installs a default signal handler in
the Linux kernel. With EPCKPT, the process checkpointing task is performed by sending a specific signal to the process which is to be checkpointed; in sequence the associated signal handler gets activated which the current process is checkpointed.

It has the prominent features like transparency in the checkpoint-restart mechanism. It sends the checkpoint image directly to the destination workstation, thereby reducing the disk I/O overhead. However, this feature has a drawback, as the generated checkpoint image is not stored on the persistent storage; so, the availability of checkpoint-image cannot be expected for later or repetitive usage.

The EPCKPT utility has some drawbacks also. One of the drawbacks is that it adds excessive information logging to the kernel. For instance, in order to know the names of opened files, it patches the open and close system calls. It also patches mmap, fork, exit, etc. and introduces a lot of overhead and makes the implementation rather complex. Moreover, to reduce the checkpointing overhead, it also adds one more system-call, collect_data. By default, no information is logged. If a user wants to checkpoint one process, he has to call this system call before it starts to initiate the logging. Without starting the logging mechanism, the process can't be checkpointed. Thus, the user has to know in advance whether he will checkpoint and migrate a process before he launches the process. This could be very inconvenient and inefficient [H. Zhong, 01].

2.3.7 CRAK
The CRAK: Linux Checkpoint Restart As Kernel Module is an innovative and transparent checkpointing module; developed by Hua Zhong and Jason Nieh at the Department of Computer Science, Columbia University basically for the Linux kernel versions 2.2.x. and 2.4.x. The CRAK system provides transparent migration of Linux networked applications and computing environments without
modifying, recompiling, or relinking applications [H. Zhong, 01]. It supports process migration in the homogeneous environment.

Unlike other packages discussed earlier, the CRAK system does not leave any residual dependency (i.e. the stub) on the originating workstation, thereby providing true process migration. Also, unlike other kernel-level implementations, the CRAK is implemented through the dynamically loadable kernel modules, thereby not insisting the existing operating system setup patching nor recompilation and relinking. But, the usage of dynamically loadable kernel modules (LKMs) has restricted the applicability of the CRAK for a few kernels, e.g. some of the services such as kernel symbols exported to LKMs in kernel version 2.4 are not available to the higher versions; this makes CRAK not useful to the higher kernel versions.

Moreover, some of the kernel-level features and kernel symbols are restricted to accessibility in the kernel space only and not exported at all in the LKMs. This limits working scope of LKMs; e.g. assigning a particular process identifier value to a certain process (which is not supported by CRAK) necessitates working with the kernel-layer instead of the dynamically loadable kernel modules.

Moreover, after migrating to the destination workstation, the migrant process’s behavior remains ambiguous if it resumes execution from inside some system call, i.e. if the process was migrated while it was within some system call, it might have behaved ambiguously on the destination workstation. Furthermore, although the CRAK system does not require existence of any type of residual dependencies after migration is over on the originating workstation. It takes more time in checkpointing the memory regions occupied by the process which is to be migrated, this affects the performance criteria of the optimized load balancing technique. In fact, the time required to perform checkpointing should be as less as possible in order to carry out the
load balancing practice in the optimized manner.

### 2.3.8 ZAP

The ZAP software is a system for migration of computing environments, developed by Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh at the Computer Science Department, Columbia University [S. Osman, 02]. The ZAP software improves on the CRAK (discussed above) by inserting a virtualization layer named as POD (PrOcess Domain). It enables process migration of legacy applications without modifications, recompilation and re-linking of the application’s source code and libraries [S. Osman, 02].

As the Zap virtualization mechanism is integrated with a checkpoint-restart mechanism, it empowers processes within a POD to be migrated as a single entity to another workstation. The PODs are independent and self-contained entities, and they do not leave any residual dependency on the originating workstation of the process. As the POD namespace is private, only processes within the POD can see the namespace. Being private, POD masks out resources that are not contained within it. Processes inside a POD appear to one another as normal processes, although the processes outside a POD do not appear in the namespace and are therefore not able to interact with the processes inside the POD.

In the ZAP system, the virtualization layer is responsible to deal with the resource conflicts (such as Process Id) that arise during process migration between workstations having different persistent states. However, the ZAP tool achieves these features by intercepting every system call (issued by a process within the POD) at the virtualization layer, which introduces some run-time overhead. Moreover, it will become difficult for the system administrator to communicate with and control the process that is within the POD. Also, the PODs are limited to migration as an entire POD as opposed to migration of a
few processes of choice.

Furthermore, from the beginning, a new process must be created within the POD, if it is supposed to be migrated in future. This puts limitations on the already running processes which are not part of the POD, as they will not be able to take advantage of POD migration. Other serious problem with the POD could be, if one process damages the POD execution, then the whole group of processes in the same POD may get affected.

### 2.3.9 AutoPod

The AutoPod system was built at Columbia University; which provides a simple self-managed infrastructure for the self-maintenance of operating system. The AutoPod system is responsible to exclusively allow operating system updates in the unscheduled way while preserving application service availability during system maintenance [S. Potter, 05]. In order to achieve this objective, the AutoPod system involves checkpoint-restart mechanism. The AutoPod is based on a virtual machine abstraction called a pod (PrOcess Domain).

Again, as discussed in the ZAP package, the pod namespace is private, and the processes which are outside the pod (such as admin or root) are not able to control or interact with the processes inside the pod [R. Baratto, 04]. One more overhead with the pod to be considered is, in order to checkpoint or to migrate a particular process, the entire pod is required to be checkpointed or migrated.

### 2.3.10 CHPOX

The CHPOX (CHeckPOinting for linuX) software is a set of the Linux kernel modules and user-space programs for transparent dumping of specified processes into disk file and restarting them later. It is based on CRAK [H. Zhong, 01], EPCKPT [E. Pinheiro, 98] and VMADump; it was developed by Olexander P. Sudakov. It creates a new entry in the
/proc pseudo file system and also a new kernel signal such as SIGSYS [O. sudakov, 07]. As CHPOX is developed in the form of dynamically loadable kernel modules, it doesn’t require recompilation and re-linking of existing applications.

It is based on VMADump that writes memory regions of a process to disk file [E. Hendriks, 02]; as VMADump is of nature of application-initiated, the CHPOX system may not be applicable to the server-managed load-balancing. Application-implemented checkpoint has some weaknesses. Sometimes, it will be difficult or not possible to modify the application’s source code, such as legacy applications and third party scientific applications. Another serious drawback of this technique is the lack of a generalized restart mechanism. It may be impossible for an automated system to restart an arbitrary application. Moreover, the CHPOX system can checkpoint only non-interacting processes, and sometimes it may behave ambiguously with the process which involves dynamically-linked libraries on the destination workstation. Later, it was added to the [OpenMosix] project, which had been officially declared as closed in March, 2008.

2.3.11 CryoPID

CryoPID was spawned out of a discussion on the Software suspend mailing list about the complexities of suspending and resuming individual processes. CryoPID provides facilities to capture the state of a running process in Linux, and such a captured state of the process can be used to resume the process on another workstation. It provides a homogeneous technique and supports a user-level process migration facility [CryoPID].

However, the problem of lack of consistency and resource conflict still exists in CryoPID after migrating to the destination workstation; the migrated process loses its original process-identifier value i.e. the pid value which the process was assigned on the originating workstation.
before migration took place. In fact, after migration, during resumption of the migrated process, it should be allocated the same pid value, which it possessed on the source workstation.

In addition, the behavior of system calls is found to be ambiguous with CryoPID. When a victim process (i.e. the process which is to be migrated) is running some system call on the source workstation and at the same time if CryoPID migrates that process, then the process will behave ambiguously after migrating to the destination workstation. It may simply skip the system call and resume execution from the just next instruction which appears after system call. In this dissertation, we have suggested mechanisms to overcome the above two deficiencies in the CryoPID system.

2.3.12 Virtual Machine Systems
The Virtual machine systems are designed to provide a software-emulated hardware environment. Different virtual machines are separated from one another. Specialty of the virtual machine systems is that the image (i.e. the current state) of the entire virtual machine can be saved to disk file and resumed later. Oracle’s VirtualBox [VirtualBox], Disco [E. Bugnion, 97], and VMware’s VmW [VmWare] are the examples of Virtual Machine Systems.

However, the virtual machine systems can only do checkpointing on a per-Virtual Machine basis, which is too coarse-grain. The entire virtual machine’s current state image has to be saved to the disk (typically several hundreds of megabytes), even if only a small part of the memory is actually being used. It results in very expensive migration operations [H. Zhong, 01]. Moreover, achieving the migration of virtual machine systems based processes is much costlier and difficult to attain as the migration mechanism has to not only consider the similarities and differences but also has to migrate the whole image (i.e. current state) of source workstation to some remote
workstation.

2.4 Significant Remarks on Literature Survey

Though much research has been done in this area, most of the work remains inaccessible for the computing environments built using contemporary hardware and operating systems. Implementations discussed are – either too old or not publicly available, require use of operating systems which are not supported by modern hardware, or simply do not meet the functional requirements demanded by practical use in real world settings.

As an outcome of this study, we have found either inefficiency in techniques available or lack of kernel-mode support. Moreover, it is felt that there is a need to contribute research to (i) checkpoint the memory regions which are occupied by an executing process, (ii) migrate the system call being currently run by the process and (iii) resume the migrated process with the same process-id which the process had possessed before migration took place.

Some of the process migration systems use a page migration technique that transfers a checkpoint-image page of a process to the destination workstation only when the page is actually accessed. We have not considered such a practice, but simply not only checkpoint but also migrate all the memory regions (occupied by the process) at the time of checkpointing of the process for the following reasons:

- It requires the victim process to be alive at its originating workstation even after it has migrated and resumed execution on the destination workstation, i.e. we have to leave some residual dependency still on the originating workstation of the process; which we do not want at all to achieve the benefit of fault tolerance.
Chapter-2: Research Contributions in the Field of Process Migration

- The page transfer implementation becomes far more complex; also it varies from platform to platform.

Moreover, in order to make offloading as handy as possible for administrators and system users, we preferred to facilitate an automatic mechanism to keep track of lightly loaded workstations and to select destination workstations to carry out process migration and load balancing.

The process migration and load-balancing mechanisms proposed in this thesis extend the work suggested in the [H. Zhong, 01] and the [CryoPID] in order to overcome some of their limitations and weaknesses, to supplement some of the missing functionalities which are required to fulfill the proposed process migration mechanism and to work on the contemporary kernel-versions.

2.5 The Work

In this study, new algorithms have been designed for effectively carrying out the mechanisms such as process migration (from one workstation to another workstation) and load-balancing. Having studied some of the existing process migration mechanisms, we have designed algorithms to checkpoint and restore certain state information of a process which is under execution.

The process state information comprises of the memory occupied by the process; our contribution includes identification and checkpointing of these memory regions’ details and contents in an effective and optimized manner.

During execution on source machine, the running process possesses certain process credentials. Our contribution includes development of mechanisms for preserving the original value of some of the process
credentials and restoring the same values even after the resumption of the migrated process on the destination machine.

At the time of checkpointing a particular process, the process might be executing some system call; our contribution includes implementation of a mechanism to checkpoint the system call details and to resume the process-execution in the system call during process-state restore on the remote system.

Thus, the research contributions (i.e. OptiMigrator) presented in this thesis include development of algorithms and their implementations to carry out optimized process migration:

- to capture the process state information (on source machine) such as –
  - memory-regions’ contents possessed by the process,
  - details of the system call which is being currently executed by the process,
- to resume the migrated process with the system call (which the process was executing on source workstation before migration took place) on the destination workstation,
- to resume the migrated process with the same process-id (which the process was assigned on the source workstation), and
- to balance the load of the workstations of a network.

Moreover, unlike the weaknesses and limitations of the existing process-migration and load-balancing mechanisms which have been discussed earlier in this chapter, we have suggested algorithms and their implementations for load-balancing and process migration mechanisms with the following salient characteristics:

- The suggested algorithms support the preemptive process migration.
Chapter 2: Research Contributions in the Field of Process Migration

- The suggested work supports homogenous process migration.
- No modifications are required to existing applications.
- The suggested solution does not require the application programmer to include any customized library in his application program development.
- The described algorithms do not require recompilation and re-linking of existing application source code.
- The suggested solution does not require existence of any shadow process (of the migrated process) on source workstation; i.e. it does not leave any residual dependency on originating workstation of the process.
- The described solution directs the checkpointed image to the disk file instead of directly migrating the image to the destination workstation; the image stored on the disk may be used several times even after the rebooting the workstation.
- The work does not impose changes to the already installed and running system software such as compilers, linkers and loaders.
- Memory regions checkpointing is performed optimally.
- The suggested solution supports the passive process migration mechanism in order to support the load-balancing mechanism.

The characteristics and design goals listed above are deliberated in next chapter.

2.6 Concluding Remarks

A study about the two types of process migration technique – the user-level process migration and the kernel-level process migration – is carried out in this chapter. A literature survey about the existing kernel-level techniques such as Amoeba, V Distributed System, MOSIX, BLCR, Sprite, EPCKPT, CRAK ZAP, AutoPod and CHPOX along with their proficiencies and weaknesses, is presented in this chapter.
The chapter also presents a study about the well-known user-level process migration techniques, such as Condor, CKPT, Esky, UTOPIA, MPVM, REXEC, rsh, Libckpt, Libtckpt, CoCheck and Xen along with their capabilities and limitations thereafter.

In addition, the chapter describes points out significant remarks on literature survey by insisting on the necessity of more research contributions especially for (i) checkpointing of the memory regions, (ii) migration of the system calls, and (iii) migration of the process credentials. Finally, the chapter concludes with the discussion on the process migration mechanism.