ISSN No. 0976-5697

Volume 4, No. 2, Jan-Feb 2013

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

User-Level Process Migration Mechanism

Dr. Narayan A. Joshi*
Assistant Professor, Computer Science Department,
Institute of Science & Technology for Adv. Studies & Research, Vallabh Vidyanagar, India
narayan.joshi@yahoo.com

Dr. D. B. Choksi
Professor,
G. H. Patel Post Graduate Department of Computer Science & Technology, Vallabh Vidyanagar, India
dbchoksi@yahoo.com

Abstract: In networks of workstations, running processes lead to a situation, in which some of the nodes are highly loaded whereas other nodes may remain lightly loaded or almost idle. It may lead to reduction in throughput and lot of computational power offered by the lightly loaded or the idle processors. This computational power goes unused while the jobs are striving for the availability of processor-cycles. The issue of efficient utilization of computing resources is noteworthy in the networked workstations such as local area networks as well; which can be tackled by spreading the running processes among the connected workstations by means of the mechanism such as process migration. The application of process migration mechanism may consequent into the proficient utilization of the overall networked computing environment through the utilization of the lightly loaded workstations or the almost idle workstations. The paper discusses characteristics of many user level process migration mechanisms.

Keywords: process migration, load balancing, process checkpointing, user-level process migration, distributed systems

I. INTRODUCTION

The implementation of distributed computing systems is emerging day by day as an outcome of the availability of advanced computing hardware resources at reduced cost and innovations in the communication and networking technologies. One of the primary objectives of distributed systems is its capability of resource sharing in order to serve its users with an augmented combination of computing resources which are generally distributed around the network's workstations. Another important advantage with distributed systems is that the concept of load balancing may be implemented on such a network of workstations.

The earlier years have has perceived the realization of parallel computing systems which have been in practices in high performance computing applications [7]. Moreover, it is often observed that, the hardware has been upgrading faster than the relevant system software including the operating systems. Moreover, some of the shortcomings do exist with parallel systems, for example deficiency of fault tolerance and negligible support in fast upgradation of hardware as the augmenting of modern hardware components to the parallel systems requires rebooting of the running system and upgradation in underlying system software and operating system policies and therefore results into discontinuity in the availability of the system. Moreover, the developments in computing technology and networking technology has led to the availability of powerful, inexpensive CPUs and economical loosely-coupled computing environments called distributed systems; in which software modules sited at interconnected workstations collaborate their jobs to accomplish the characteristics like – resource sharing, information sharing, higher throughput, better flexibility and greater reliability. Moreover, in the network formed through such distributed systems it becomes easier to supplement more nodes or to disjoin the surplus or malfunctioning workstations. These noteworthy features of distributed systems overcome the aforesaid limitations of the parallel computing systems [6], [10]. However, the total computing capacity of the networked workstations may theoretically appear to be larger than that of the single isolated workstation. It is frequently observed that in such networks of workstations, frequently seen that some of the nodes are highly loaded while others are left as lightly loaded or nearly idle; which results into application of merely a fraction of the total theoretical computing power of the network [13], [14]. But, on the other side, in such circumstance, the technique of load sharing or load balancing may become useful to achieve better performance in the network of workstations. We may implement the load balancing technique by migrating some of the already running processes from the overloaded nodes to the idle or lightly loaded nodes of the network [12].

Section 2 of this paper highlights the mechanism of process migration. There are numerous types of process migration techniques; the section 3 enlists some of them. The paper mainly focuses on the user-level process migration mechanisms, which have been discussed in the section 4. In the later part of the paper, remarks on the study of the user level techniques have been briefed in the section 5.

II. THE TECHNIQUE OF PROCESS MIGRATION

In the network of workstations, it is common that some workstations remain lightly loaded or nearly idle to whom the mechanism of migration of already running process may produce better utilization of the overall computing environment [4]. In addition to the advantage of load sharing, the process migration mechanism also serves the features like efficient utilization of computing resources and fault tolerance and enhanced system management. The mechanism involves migration of process from its originating workstation to some other workstation; the phases involved are on the source workstation, checkpointing of the current state information of the process which is to be migrated, putting the checkpointed state information to a persistent storage device, sending the state information to the destination workstation on which the process is to be resumed and after migrating to the destination workstation, restoring and resuming the migrated process on destination node. The migration can be either static or dynamic. Static migration involves transfer of only newly submitted process which has not begun its execution, whereas dynamic migration involves migration of partially executed processes; compared to static migration, the dynamic process migration is a quite challenging mechanism. The scenario of dynamic process migration mechanism is represented in figure 1 [12].

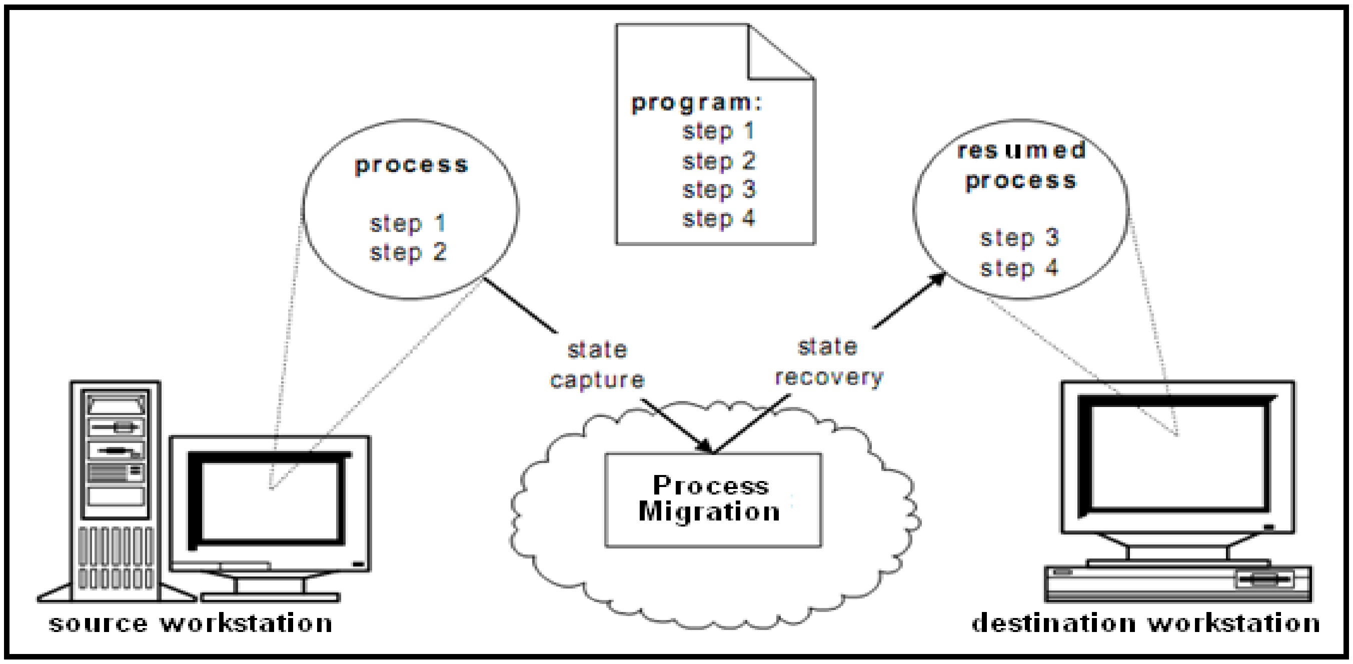

Figure 1. Process Migration Mechanism

III. PROCESS MIGRATION FLAVOURS

When a process itself performs self-migration, the migration is called active migration; otherwise when some other process performs migration of the process which is to be migrated, the technique is called passive migration. When migration occurs between same platforms, the technique is called homogeneous migration, otherwise it is known as heterogeneous migration. When associated file is only transferred to the destination workstation, it is called weak migration; otherwise the static and dynamic process migration, strong and weak process migration. Moreover, the process migration techniques can be classified into two more significant types called kernel-space process migration and user-space process migration.

The user-space process migration techniques perform migration of the process without changing the code of the kernel of the operating system; which makes implementation of such techniques simpler. The primary and major difficulty in user-level process migration mechanism implementation is - such mechanisms cannot access the features which are accessible to only kernel-space and which are out of sight in the user-space; therefore, they cannot deal with migration of certain state information of the process which is to be migrated, thereby limiting the applicability of the process migration to all processes [12], [13]. Whereas, the kernel-space techniques utilize the services available to underlying operating system's kernel in order to perform migration of process from its originating workstation to the destination workstation. However, such techniques certainly require modification of the kernel; which makes the scope and availability of such techniques limited to particular kernels for which it has been implemented [5], [12]. However, a good amount of research has taken place in the areas of user-space and kernel-space process migration. Here we describe some of the known mechanisms in the domain of user-level process state checkpointing and process migration.

IV. USER-SPACE PROCESS MIGRATION

This section focuses on some of the significant research contributions in the field of user-space process migration mechanism.

A. Libckpt:

The Libckpt utility is developed as library routines in the form of includable header files. It is useful to perform the checkpointing of processes on UNIX platform [9]. As Libckpt is implemented in the form of library, it is applicable to self-checkpointing processes, i.e. it supports the active process migration. Furthermore, utilization of Libckpt requires recompilation and relinking of the application's source files in association with the Libckpt header files. An important characteristic is that the generated checkpoint-image file consumes less space that results into faster file transfer which leads the mechanism towards better efficiency. However, it not only requires modifications in the application source code but also it requires renaming of some of its source files. Moreover, the applied changes to the source files must be transformed into final application executable files also; which may become problematic in case of infeasibility of modifications of the application's source code.

B. Libckpt:

As like as the Libckpt utility (discussed above), the Libtckpt utility is also developed in the form of a set of libraries which can perform active process migration. An additional capability that it possesses is that it can checkpoint multithreaded processes involving the Linux and Solaris threads [16]. In order to achieve its goal, the Libtckpt adds one more thread of execution to the application which is to be migrated. However, being a library implementation, the Libtckpt library also requires modifications to the application's source files in addition to the execution of the initialization routines.

C. Ckpt:

The Process Checkpoint Library package is a set of libraries and programs which are capable to perform user-space process checkpointing. The technique developed by Zandy does not require modification and recompilation of the pre-compiled and pre-linked application's source and binary files. As well, it is also possible to inject the library to the applications which already are under execution. Moreover, the checkpointed image file can be resumed later. It supports checkpointing of environment of the process. But, migration of the credentials of a process is problematic with the Process Checkpoint Library [15].

D. Esky:

The Esky technique is a system is capable to checkpoint and resume the processes in the UNIX environment [2]. One of the characteristic of Esky is that the process (which is to be migrated) must be directed to run in the ambience of the monitor provided by the Esky software. This requires that the application programmer or the user must know well in advance that the process could be migrated later, which may create inconveniency during its usage. The other point to be considered while working with Esky is that the signal handers for the signals such as sigalarm may not function transparently as the Esky also takes help of sigalarm signal.

E. REXEC:

The REXEC utility facilitates a secure and decentralized execution facility. Apart from enabling the feature of migration, it also supports the features such as scalability, transparency, availability, discovery of decoupled nodes, dynamic cluster configuration, authentication, encryption and support to the parallel applications. However, one of the limitation with REXEC is it lacks in relinking with the remote execution related requirements and its usage applicable for certain version of kernel [1].

F. Condor:

The Condor system is a scheduling system for an environment of UNIX workstations which provides the feature of checkpointing and migration [11]. The Condor's job scheduling system migrates job from some non-idle node to some idle node. In order to attain such characteristic, the Condor system performs first checkpoints the process on its originating node, transfers the checkpointed image to some idle node and makes arrangements on that idle node such that the migrated process image resumes its execution [3]. Moreover, it enables the process to perform self-checkpointing with help of its library which can be activated through the checkpoint signal provided by Condor [8]. However, the Condor system requires recompilation and relinking of the application's source files.

V. CONCLUDING REMARKS

Although abundant work has been accomplished in the domain of process state checkpointing and migration, a major part of the work stays incongruous or difficult to get to the computing environments produced by means of recent hardware and system software including operating systems and networks connected using modern networking technologies. The discussed mechanisms are either publicly unavailable or involve usage of operating systems which are not supported by contemporary hardware or too old or, simply do not meet the functional requirements demanded in real world scenarios.

VI. REFERENCES

[1] B. N. Chun and D. E. Culler, "REXEC: A Decentralized, Secure Remote Execution Environment for Clusters", CANPC '2000: Proceedings of the 4th International Workshop on Network-Based Parallel Computing, pages 1-14, London, UK, 2000. Springer-Verlag

[2] D. Gibson, "Checkpoint/restart for Solaris and Linux", 1999, Available from http://ccnuma.anu.edu.au/cap/cap/reports/report99/node14.htm

[3] D. Thain, T. Tanenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience", ACM Journal of Concurrency and Computation: Practice and Experience, Vol. 17 Issue 2 -4, February-April, 2005

[4] J. Basney, M. Livny and P. Mazzanti, "Utilizing Widely Distributed Computational Resources Efficiently with Execution Domains", Journal of Computer Physics Communications, Vol. 140, 2000

[5] J. C. Sancho, F. Petrini, K. Davis, R. Gioiosa and S. Jiang, "Current practice and a direction forward in checkpoint/restart implementations for fault tolerance", Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing, 2005

[6] J. D. Smith, "Fault Tolerance using Whole-Process Migration and Speculative Execution", M. S. thesis at California Institute of Technology, 2003

[7] J. M. Smith, "A Survey of Process Migration Mechanisms", Technical Report CUCS-324-88, , Columbia University

[8] J. Meehean and M. Livny, "A Service Migration Case Study: Migrating the Condor Schedd", Midwest Instruction and Computing Symposium, April 2005

[9] J. S. Plank , M. Beck, G. Kingsley and K. Li, "Libckpt: transparent checkpointing under Unix", USENIX 1995 Technical Conference Proceedings

[10] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume", Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, IEEE CS Press, 2002

[11] M. Livny and J. Basney, "Managing Network Resources in Condor", Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), 2000

[12] N. A. Joshi, "Development of Algorithms for Optimized Process Migration for Load Balancing in Distributed Systems", PhD Thesis, Sardar Patel University, 2012

[13] N. A. Joshi and D. B. Choksi "Process Migration Techniques", International Journal of Information & Computing Technology, Vol. 2 Issue 2, Nov. 2012

[14] S. Malik, "Dynamic Load Balancing in a Network of Workstations", 95.515F Research Report, 2000

[15] V. C. Zandy, "CKPT: A Checkpoint Library under UNIX", 2004, http://www.cs.wisc.edu/~zandy/ckpt

[16] W. R. Dieter and J. E. Lumpp; "User-level Checkpointing for Linux Threads Programs", Proceedings of FREENIX Track: USENIX 2001 Annual Technical Conference,2001