International Journal of Engineering Science Invention (IJESI)
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org ||Volume 7 Issue 4 Ver. III|| April 2018 || PP. 01-05

A Survey of Process Migration Mechanisms in Distributed Computing Systems

Dr.Shamsudeen.E
Department of Computer Applications E.M.E.A. College of Arts and Science, Kondotty, Malappuram, India.
Corresponding Author: Dr.Shamsudeen.E

ABSTRACT: *The process migration in distributed system is the ability of the system to transfer process from one processor to another across the network. This paper presents how the various systems like MOSIX, V, MPVM, Sprite, and Condor are dealing this issue. Location transparency, dynamic process migration, single system image, autonomy, scalability are maintained while migration occurs. The paper also presents what are the causes of the process migration.*
KEYWORDS: *Availability, Load Balancing, Process Migration, Transparency*

Date of Submission: 31-03-2018 Date of acceptance: 16-04-2018

I. Introduction

Process, a program in execution can be migrated from one processor to another node usually in order to balance load across the distributed system network

.

1.1 Reasons of the process migration

Migration [1] of the process (Fig.1) is the ability of the distributed [2] operating system due to variety of reasons like,

1. Load balancing
2. Communication performance
3. Availability
4. Utilizing special capability

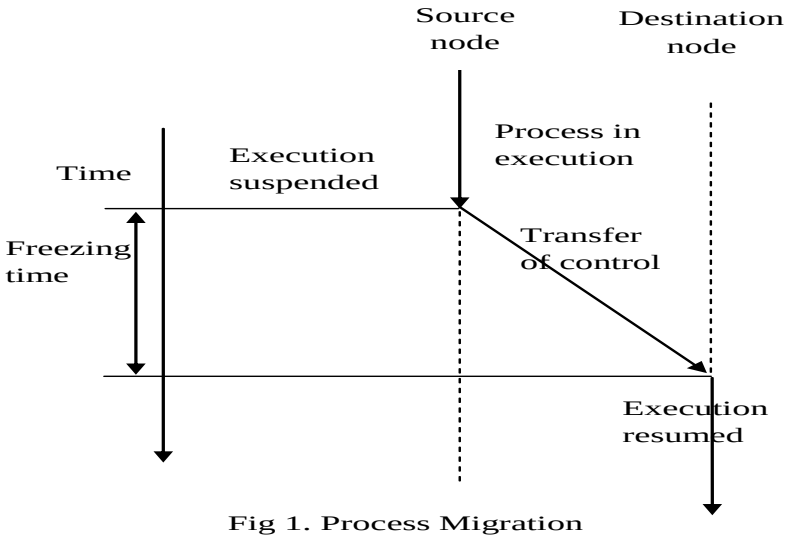

Fig 1. Process Migration

1.1.1 . Load balancing

Moving process from one heavily loaded to lightly loaded processor. This load balancing can have much to do with the performance of the distributed system. Many distributed system like Eager [3], Lazowska [3], Zahorjan [4], MOSIX [5], Sprite [6] [7], V [8], Condor [9] [10] and Mach [11] implemented process migration with high performance so as to reduce the average response time processes and hence higher throughput.

1.1.2. Communication performance

If requests are sending through the communication channel to the processor where the process resides, then the traffic on the communication channel will also very high. This high traffic affects the performance of the entire system. This can be tackled by moving the processes to the node where the intensively interacting process resides.

More over the process that needs more remote data can be moved to the remote node where remote data largely available.

1.1.3. Availability

Long running processes can be moved to the processors which are idle or underutilized. Processes reside on a failure node can be migrated to other healthy nodes.

1.1.4 Utilizing special capability

Process can take advantage of unique hardware or software capabilities. That is, the processes residing node is less fast and the software available in the node is less capable for faster response time can be moved to the node where both software and hardware are highly faster and to do the processes' requirements.

1.2. Types of Process Migration

There are two types of process migration, they are,

1 Homogeneous process migration
2 Heterogeneous process migration

1.2.1. Homogeneous process migration

Homogeneous process migration [12] involves migrating processes in a homogeneous environment where all systems have the same hardware and software but not necessarily the same resources or capabilities. In homogeneous fashion process migration can be performed either at the user-level or the kernel level

1.2.1.1. User-level Process Migration

User-level process migration techniques support process migration without changing the operating system kernel. User-level migration implementations are easier to develop and maintain but have two common problems:

i. They cannot access kernel state which means that they cannot migrate all processes.
ii. They must cross the kernel/application boundary using kernel requests which are slow and costly.

1.2.1.2. Kernel Level Process Migration

Kernel level process migration techniques modify the operating system kernel to make process migration easier and more efficient. Kernel modifications allow the migration process to be done quicker and migrate more types of processes. Unfortunately, many older implementations have high overhead, long freeze times, and still cannot migrate all processes.

1.2.2. Heterogeneous process migration

Heterogeneous process migration [12] is process migration across machine architectures and operating systems. Obviously, it is more complicated than the homogeneous case because it must consider machine and operating specific structures and features, as well as transmitting the same information as homogeneous process migration including process state, address space, and file and communication information. Heterogeneous process migration is especially applicable in the mobile environment where is highly likely that the mobile unit and the base support station will be different machine types. It would be desirable to migrate a process from the mobile unit to the base station and vice versa during computation.

II. Case Studies

In this section, we analyse several typical migration implementations. The systems we discuss are,

2.1. MOSIX

MOSIX is a general-purpose Multicomputer Operating System which integrates a cluster of loosely connected, independent computers (nodes) into a single-machine UNIX environment. The main properties of MOSIX are its high degree of integration and the possibility of scaling the configuration to a large number of nodes

The design goals of MOSIX are

1. Single System Image [5]. It provides a single view of the file system and network transparency is provided at the user level.

2. Autonomy [5]. Kernel is replicated in each processor and thus autonomic. It makes its own control decision independent of other nodes.

3. Scalability [5]. Probabilistic algorithms are used to minimize system management and network overhead.

4. Dynamic Configuration [5]. Node may join and leave a cluster at will without significant adverse effect.

In MOSIX the kernel is divided into three layers as in the fig 2.

- The upper kernel
- The linker
- The lower kernel

The lower-kernel is machine-dependent and operates autonomic. That is, it provides the services like disk access and others. While the upper kernel is machine independent and it provides standard system call interface and it keeps knowledge of whereabouts of all objects it handles.

The linker is responsible for inter-node communication, data transfer, process migration and load balancing.

In MOSIX, the process migration is done with a handshaking between the source processor and destination processor. During migration, only user areas of the migrating process are transferred. The dirty pages are too migrated while clean pages are paged in whenever there a page fault occurs at the destination node.

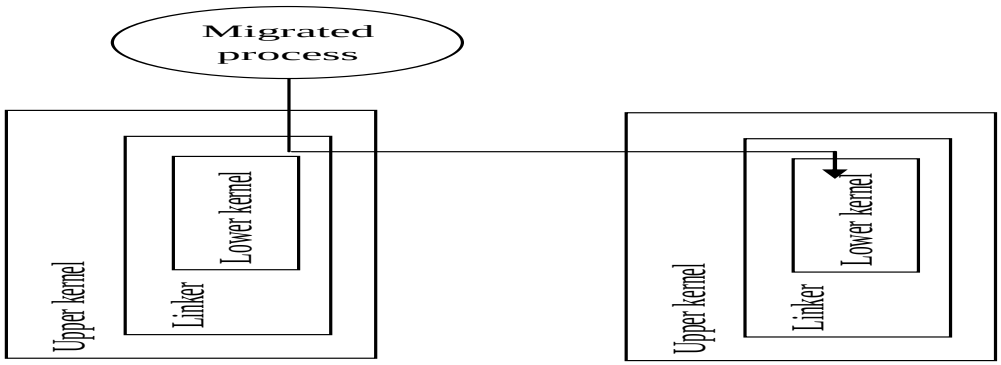

Fig. 2: MOSIX architecture

2.2. Sprite

It is a time-sliced distributed system with a group of personal workstations. Each host runs a copy of the Sprite kernel and work closely together using a remote-procedure-call (RPC) mechanism. Processes in Sprite can access files or devices on any host, and data can be cached across the network while consistency is maintained with the one-copy semantic.

The design goals process migration in Sprite is:

1. Utilize Idle Hosts [6]. Idle hosts are plentiful even at the busiest time of the day.

Utilization of these otherwise wasted computing resources can give a boost in performance to other loaded hosts.

2. Exclusive Use of Workstations by their owner [6]. Computer resources privilege is given to the local user. Migrated processes are evicted back to their source nodes whenever the workstation owner log into his/her machine.

3. Kernel RPC [6]. Sprite uses protected kernel RPC as a form of interprocess communication. Redirection of communication channels is not as obvious as it is in the message-passing kernel.

In Sprite each process associated with its designated home node, regardless of its physical location. That is, sprite provides location transparency to the migrated process. Host-specific system calls are accomplished by kernel-to-kernel RPCs. Other calls, such as memory allocation and the distributed file system-related calls are handled locally.

2.3. System V

System V is a distributed operating system that is run on a cluster of networked workstations (see figure 3). Every host runs a small identical kernel plus some service modules and run-time libraries. The implementation scheme of V provides transparency, minimal interfaces and residual dependencies on the source host.

The design goals V's process migration are,

1. Network Transparency [8]. Network transparency is implemented with the use of V IPC primitives and global naming to provide communication channels between a program and the operating system.

2. Minimal Interference to the system [8]. Process migration is regarded as an atomic transfer of process states between two hosts without interfering other parts of the system.

3. No Residual Dependencies [8]. Process states are not left over on the source node once the process migration has been committed to protect programs executing on behalf of different (remote) users on the same workstation.

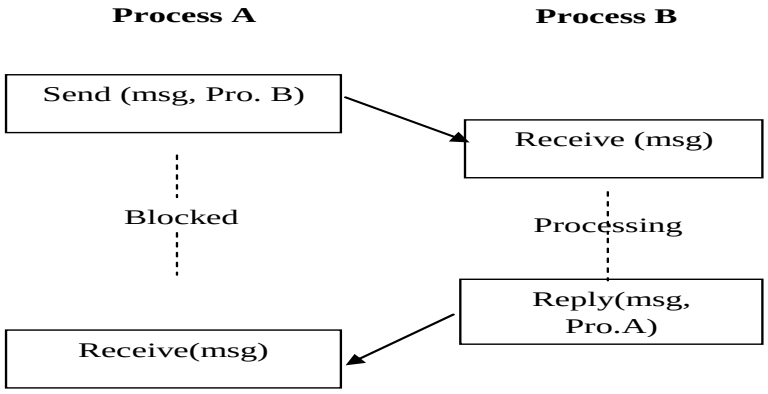

Fig. 3. Interprocess Communication in V

System V uses a logical host to keep the processes and its address space and it is identified by a data structure (logical-host-id, local-index). In order to accomplish the process migration the logical host is migrated in which the process is running. The system V uses the mechanism called 'precopy' to transfer the address space of the migrated process.

Here a fixed timeout value is set with system calls so as to improve triggering of critical operations. System V also resolves the problem of residual dependencies by putting the condition that the execution context of a program is either resided in its address space or in a server which is globally accessible.

2.4. Condor

Condor is a system that identifies idle machines in a network and offloads processes to those machines. Here each workstation has a local scheduler and a back ground queue which holds the jobs submitted by the user and a centre co-ordinator is present on one work station. Each station keeps information regarding its jobs. The centre coordinator uses a polling mechanism to see which processors are idle and allocate jobs.

The design goals of process migration in Condor are:

1. Maximize Computation Utilization [9]. Idle workstations are potential destinations for process migration because otherwise the computation cycles may be wasted.

2. No Modification to Kernel [9]. UNIX systems are proprietary and access to the internals of the system is not possible at the time Condor is proposed.

Condor does not really taking to account affinity based scheduling. Whether such a policy would make sense in a loosely coupled system where all remote resources are freed up and no residual dependencies.

2.5. MPVM

In MPVM, a unit of work is known as a task and it uses global scheduler which is basically a centralised resource manager. The global scheduler decides the task when and where to be migrated.

The design goals of MPVM are,

1. The location transparency [13]. it is accomplished with the help of global scheduler(GS)
2. Dynamic allocation of hosts [13]. Migration is done during work creation and eviction or when a node under heavily loaded.

Idle nodes are identified by the global scheduler in a manner somewhat similar to Condor. Moreover, MPVM does not support cache or I/O affinity.

III. Conclusion

This paper presented what are the needs for a process to be migrated from one node to another in distributed systems. Then it analyses different distributed mechanisms on the basis that how they migrate processes from one processor to other and what are their design goals. MOSIX, Sprite, V, Condor, MPVM distributed mechanisms' process migration is presented in detail.

References

[1]. Tanenbaum, Andrew S. 1993 Distributed operating systems anno 1992. What have we learned so far? Distributed Systems Engineering, 1, 1 (1993), 3-10

[2]. Barak, A. and Litman, A. (August 1985). MOS: a Multicomputer Distributed Operating System Software – Practice and Experience, 15(8)

[3]. D. Eager and E. Lazowska and J. Zahorjan: The Limited Performance Benefits of Migrating Active Processes for Load Sharing. In Conf. on Measurement & Modeling of Comp. Syst., (ACM SIGMETRICS), May 1988

[4]. Frederick Douglas: Transparent Process Migration in the Sprite Operating System (PhD Thesis,University of California, Berkeley), September 1990.

[5]. Amnon Barak and Richard Wheeler. MOSIX: An Integrated Multiprocessor UNIX

[6]. Fred Douglis and John Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation

[7]. Frederick Douglis. Sprite Process Migration: a Retrospective.

[8]. Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable Remote Execution Facilities for the V-System

[9]. Michael Litzkow, and Marvin Solomon. Supporting checkpointing and Process Migration outside the UNIX Kernel.

[10]. Michael litzkow, and Marvin Solomon. The Evolution of condor checkpointing.

[11]. Dejan S Milojicic, Wolfgang Zint, Andreas Dangel and Peter Giese. Task Migration on the top of the Mach Microkernel

[12]. Peter Smith and Norman C. Hutchinson. Heterogeneous process migration: The Tui system. Technical Report TR-96-04, UBC Computer Science Department, Vancouver, B.C., February 1996

[13]. J Casas, D Clark, R Konoru, S Otto, R Prouty, J Walpole; MPVM: A Migration Transparent Version of PVM, OGI Technical Report, Feb 1995

International Journal of Engineering Science Invention (IJESI) is UGC approved Journal with Sl. No. 3822. Journal no. 43302.

Dr.Shamsudeen.E "A Survey of Process Migration Mechanisms in Distributed Computing Systems" International Journal of Engineering Science Invention (IJESI), vol. 07, no. 04, 2018, pp 01-05