EasyChair Preprint

№ 3557

MOSIX: Architecture, Job management, Cluster Management, Filesystem Management

Muhammad Umair Khan

June 5, 2020

MOSIX: Architecture, Job management, Cluster Management, Filesystem Management

Muhammad Umair Khan (14849)
Department of Computer Science
Riphah International University Lahore, Pakistan
Email:umairkhan8120@gmail.com

***Abstract*— MOSIX is a cluster management system offering a single system picture to users and applications. Users can run multiple processes within a MOSIX cluster by allowing them to search for MOSIX resources and assign and migrate automatically between nodes, without changing the interface executable environment and the associated login nodes. Users therefore do not have to change or connect applications to a specific library, modify applications, login or copy files to remote nodes or even know where their programs are going. This report represents an overview of the MOSIX architecture in the form of object models, Task management in clusters, cluster management, and filesystem management.**

INTRODUCTION

MOSIX is a general-purpose distributed system for UNIX. It's a gadget which supports cluster computing. It has kernel, resource sharing algorithms that are tended for high performance, easy scalability and easy to use it for scalable clustering. [1] The significance of the MOSIX technology is the ability of multiple servers and workstations to work supportively as if component of a system that is system. MOSIX algorithm is designed in a way that it can efficiently respond to deviations of resource allocation among different nodes. It does this task by drifting processes from one node to another, load-balancing and to resist memory exhaustion in any node. MOSIX technology is scalable and it challenges to increase the whole performance by dynamic distribution and reallocation of the workload and the resources in the nodes of any size of a computing-cluster [1]. MOSIX in time-sharing environment supports multiple users for the performing both parallel and sequential tasks.

MOSIX can change a cluster of Linux of x86 based servers and workstations to execute like an SMP. The major purpose of MOSIX is that, in case you generate one or multiple processes in login nodes, MOSIX allocates and reallocates your processes among nodes to regulate performance with best possibilities. [1] The basic of MOSIX is a set of management algorithms that constantly monitor the events of the processes vs. the existing resources, in order to return to irregular resource sharing and to take benefit of the resource with top existence.

The MOSIX algorithms use preemptive process mode to provide:

- **Automatic work distribution**-parallel processing or transition from slower to faster nodes to process.
- **Load balance**–even distribution of work.
- Migration of processes from the main memory node to prevent swapping or thrashing.
- Migration of a file server by an intensive I / O operation.
- Migration from a client node to a file server of parallel I / O processes.

First we define MOSIX architecture in the first section and the second section is about cluster job management and the third section discusses file system management and the fourth section explains cluster management and in the fifth section we give an overview of the space allocation and in the last section we discussed some unique feature of MOSIX as shown in the following section:

1- MOSIX ARCHITECTURE

The Architecture of MOSIX was designed by prof. Ammon Barak team at Hebrew University of Jerusalem, and it provides clusters of Linux with better clustering. [2] Now its prolonged with new attribute that can provide a network of clusters of Linux execute as a supportive system of united clusters. Our system design consists of clusters which are independent, e.g., of different groups, whose master desire to distribute their resources computationally, though still upholding control over private resources. The major features of the resultant system are:

- Auto discovery of resources: users do not need to have any knowledge of any specific resource's configuration or status.
- Preventive (transparent) migration processes and automatic load balancing within and across clusters.
- Modify management that responds to changes in resources free and required.
- A secure guest-process run-time environment.

- Support dynamic configuration: partitioning or combination of clusters.
- Precedence over guest processes and guest processes for local run-time.
- Prevention of flooding.
- Dynamic environment support: clusters can at any time be connected or disconnected.

The above four points were got by increasing present cluster of MOSIX within a grid environment, though the latter four are new features that were designed for such environment.

A current way to define an OS is the object model. Examples of objects are processes, files, directories, virtual terminals, pipes, memory segments, disks, terminals, disk blocks and the like possible. The OS, with time, generate objects, alters their state, and typically destroys them. [4]

The perfect split-up between the object and its low level is necessary for the object model. There is a separate program called a' type module' for each type of object, which is the only module which manipulates the internal display explicitly and is responsible for the integrity of those objects. It delivers abstract actions on those objects to the rest of the system.

The demonstration information about type module algorithms are concealed in the type module. The word' operate' is maintained below for the abstract-level operation, while' manipulate' is used for the low-level deployment of the actions. Objects can be either active or passive. Processes in MOSIX are only active objects. Only those objects that can change the mode of other objects and thus change the system mode. [4] Remember that the process can very limitedly change the mode of any other process. Sometimes we denote programs as active instances by using language, but these programs are only active if they are executed by a process.

Each object exists in a single machine in MOSIX. That is, the demonstration of the object resides in one machine's secondary or primary storage and only that machine's kernel can manipulate that object directly. Objects exist in close proximity to their physical complements, e.g. there is a terminal object in the machine to which the resulting physical terminal is bound and there is a pipe object in the machine where the pipe buffer is stored. [4] The process object is the exemption from this pattern; during its action, a process can momentarily span machines.

To be able to function on a remote object on a kernel in one machine, it must have a clear name for that object. Each network-wide system object has a general name that recognizes the object in the network exclusively. These names are not seen by the user and are generated and dynamically demolished by the system. [4]All network objects excluding processes are static, i.e. they are not transferred by machine boundaries. In these static cases, the general name includes the identity of the machines in which the object exists, as well as other information that certifies the object's acquisition and validates the connection as to whether it was made specific to an object.

1.1- INTERNAL KERNEL STRUCTURE

Kernel is the most important part of the operating system. The MOSIX kernel consists of three major sub-systems: the lower kernel, the connector and the upper kernel as shown in Figure 1. The kernel structure is the framework in which the kernel develops dissimilar definite algorithms. [4] These algorithms are not intended to be part of a high-level design; they are therefore only shown in this paper as examples.

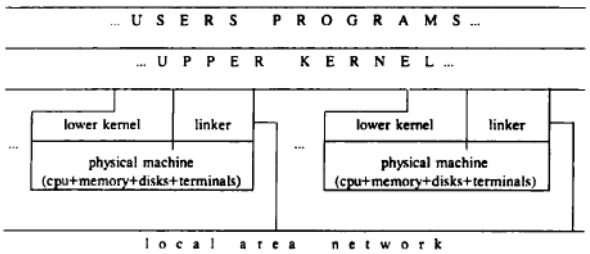

Figure 1. The system structure

1.1.1- LOWER KERNEL

This sub-system deploys the objects that exist in each machine. It is the kernel only component that has full knowledge of local objects and is responsible for their reliability. [4]The connector allows abstract actions to be performed on local objects, but the lower kernel is independent of the originator.

The lower kernel provides life support facilities for processes that exist in the system at the moment. The simple: these facilities are concerned with processor cycles: computing resources and primary memory.[4] It multiplexes the primary memory between the executable processes and the processor, these executable processes offer operations to change their computer-generated address space and coordinate with uncoordinated actions in the upper kernel. The lower kernel does not respond to the machine's ID. This determines the ID of the computer, but only uses it to establish general names for its objects.

1.1.2- LINKER

The connection connects upper kernels and lower kernels as the upper kernel has the ability to request behavior in any device in the system's lower kernel. [4] The communication protocols and the individual hardware are covered in the connector. The connector is the only one with up-to-date network formation knowledge, i.e. which machines are currently running in the network.

The connector is accountable for consistent service. Remotely operations may not constantly spread their target, but failures will be exceptional and the upper kernel never redoes a failed operation.

1.1.3- UPPER KERNEL

Upper Kernel software is the logical extension of the program oprator. Procedures performing operator’s program sometimes route on the upper kernel. Any method or process running on the upper kernel examines only the program of the operator and adds the operator's address space to the process address space. The core responsibility of this software level is to call behavioral systems and preserve the process environment while completely hiding the network from the program of the operator. [4] But the lower kernel directly requested by the upper kernel on the same machine in assured rare cases.

2- JOB MANAGEMENT IN CLUSTER

EnFuzion and MOSIX are two packages representing different cluster management approaches. MOSIX and EnFuzion are two sets of different methods for organizing the cluster. EnFuzion is a user-level line-up arrangement that can communicate to a cluster a programmed number of procedures. [5] Nimrod is a marketable variety, a device that restricts the sweeping application of phase variations. MOSIX is also an operating system (kernel) level program that facilitates preventive process relocation for near-optimal, cluster-wide reserve management, effectively eliminating the cluster track as SMP. Usually, users also use EnFuzion with a predictable cluster operating system, or MOSIX lacking a queue supervisor.

2.1- ENFUZION

EnFuzion is a presentation level set that delivers an extraordinary level environment for the conception, scattering and organization of huge constraint sweep applications. EnFuzion is a marketable software that has been discovered in the Nimrod project to build research ideas. [5] Constraint sweep applications are classified by many workers, each finding a fragment of a more global constraint. A computational model is usually performed several times, whereas there are various constraints on the model. The jobs are released separately from an early sprinkling of files and input restrictions, and a complete aggregation of data, can be distributed into a pool of processors working together. [5].EnFuzion consists of two main modules, one generator and the other generator. [5] The generator proceeds with a computational test frame explanation and shapes a file that shows how the model is going to be route and what the authentic restriction arrangements are. The dispatcher continues a route folder, and works on the connections that are available at the time are configured and routes. This is achieved by moving that job to the successor machine, using a first-come first-serve delivery method. Significantly, when a job begins to be carried out on a knot, it rests there waiting for it to finish. This simpler delivery method, as we know and see earlier, can produce systemically ruthless tasks when changing implementation times differ in one direction.

EnFuzion does not agree that processors share a file system, and so files are unoriginal from the server, which challenges the MOSIX algorithms to move the operation to the knot in which the file resides. Generally, on a particular file system, many file processes are completed by a procedure. [5] MOSIX has significant advantages over additional network file schemes as it allows the use of a native file system. It clearly denies the overhead argument between the process and the file server.

Linking EnFuzion and MOSIX gives the cluster an authoritative stage in which EnFuzion creates, assigns and lines work, and MOSIX succeeds and improves the power circulation within the cluster between knots. In some cases, EnFuzion benefits from the ability of MOSIX to perform preventive relocation of processes, and MOSIX benefits from a queue control system. [5] While EnFuzion mainly assigns procedures consistently through the cluster, it cannot regulate how the procedures will be conducted when they take placeFor example, a procedure with a set of constraints has very different memory requests from those running with a different set of limitations. This is non-deterministic in most cases and two procedures of enormous memory constraints may end up on a knot while a new node has two procedures that have slight memory constraints. This can affect the exchange on a knot and mark the cluster's overall performance. With the proactive relocation of MOSIX protocol, these procedures could be moved to other knots to allow the transfer. Moreover, MOSIX will balance the cluster’s capacity if it is not smooth. This will occur moreover although an EnFuzion route is ending some knots may turn into lazy or the CPU speediness of the cluster’s knots are dissimilar.

3- FILE SYSTEM MANAGEMENT

MOSIX is cluster handling system that cares preemptive procedure relocation. In this how we signify MOSIX Direct File System Access (DFSA), [6] an establishment that can increase the performance of cluster file systems by permitting a travelled procedure to directly access files in its current position. This competence, when pooled with a suitable file system, could considerably increase the I/O performance and decrease the network blocking by transferring an I/O severe procedure to a file server somewhat than the traditional way of carrying the file’s records to the procedure.

DFSA is appropriate for clusters that achieve a pool of common disks between multiple machines. With DFSA, it is imaginable to transfer comparable processes from a consumer knot to file servers for equivalent access to dissimilar files. [6] Any reliable file system can be familiar to work with DFSA. To test its performance, we established the MOSIX File-System (MFS) which allows stable matching procedures on different files. In which we designate DFSA and MFS.

3.1- DFSA

The Direct File System Access (DFSA) is a redirect adjustment that was planned to reduce the extra load of executing file concerned with system-calls of a transferred procedure. This is proficient by execution most of those which system calls in the knot where the process now runs. [6]The main benefits of this methodology are speedy access to files, avoiding network blocking, decreasing the network communicating overheads, and even excluding it entirely when the procedure runs in the similar knot as the file it uses. [6]

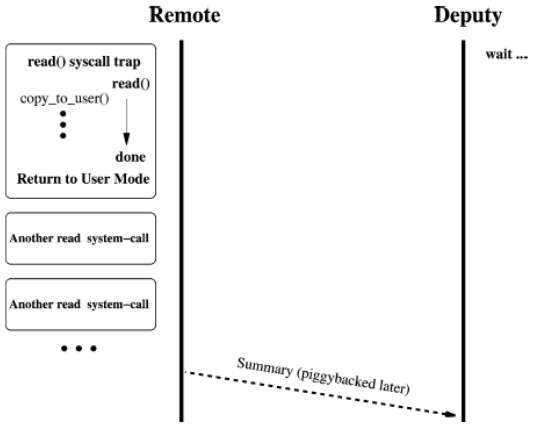

Figure 2. *Read* system-call with DFSA enabled.

DFSA is used to checks the partition which is used by a certain system-call that is confirmed to be straddling on all knots and that its file-system category provisions DFSA. If so, it generally runs file and directory concerned with system calls openly on the current knot, and only extraordinary cases are focused to the

home knot. We can example it as, when a file descriptor is mutual among many procedures.
Read system call on DFSA enabled the sequence of operations shown in figure 2
As we shown Remote-Deputy protocol in figure 1, the whole system-call is executed in the isolated knot, and although the Deputy still wants to be ultimately informed, DFSA avoids the want to interaction the Deputy per system call, and operations short summary are queued that is sent to be the deputy, commonly after a extent of several system calls, and even then, that summary is piggy-backed to further messages.
Any file system can work with DFSA that fulfils the given properties: [6]

- A single-knot stability: the outcomes of any arrangement of read/write processes on a data item by methods running in a set of knots could also happen if those methods were all running in one knot.
- Time-stamps on files and data between files in the similar partition essential be stable and non-decreasing, irrespective from which knot the changes are made.
- It must be ensured that directories/files are not clear when unlinked, any process in the cluster as long as still holds them to open.

3.2- MFS

A single-knot must be requires DFSA file and directory stability between methods that run on different knots because even the similar methods can seems to activate from dissimilar knots. To use DFSA lacking any common hardware, we applied a prototype file system, that is called the MOSIX File System (MFS), that offers a joined vision of all files on all attached file systems of any type on all the knots of a MOSIX cluster as if they were all exclusive a single panel. For example, if one steeds MFS on file /mfs/1456/usr/tmp/myfile before that the /mfs mount-point, formerly the mentions to the file /usr/tmp/myfile on node #1456. [6] This makes MFS both common since /usr/tmp may be of any file system form and accessible (since MOSIX itself is mountable and all MOSIX knots are included). MFS tree structure which is shown in figure 3. [6]

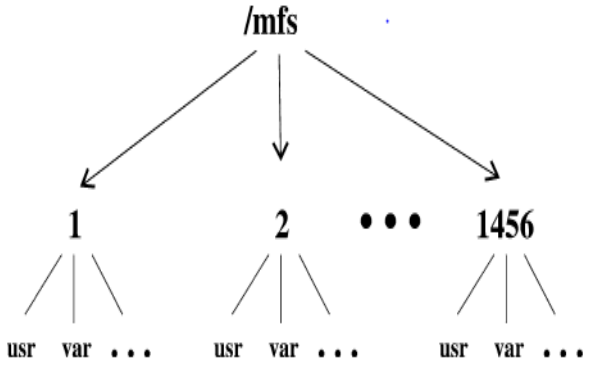

Figure 3. The MFS tree structure.

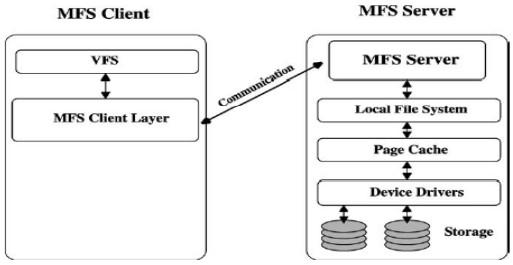

Figure 4. The MFS client–server interaction.

, See figure 4 for MFS uses a client/server model. [6]When a process matters an MFS-related system-call, the native kernel acts as consumer and forwards the demand to the suitable MFS server, client and a server both can used each node. The MFS server entrées its local file system. Dissimilar other file systems, MFS provides a single-node uniformity by preserving only one cache on the server. To device this, the regular disk and manual caches of Linux are used only in the server and are by-passed on the consumers. The main benefit of this methodology is provided that a simple, however mountable scheme for constancy. Additional benefit of the MFS methodology is floating the client–server collaboration to the system-call level which is particularly worthy for huge I/O tasks. Clearly, having no cache on the consumer is a main disadvantage for I/O tasks with minor chunk sizes.
We reminder that MFS does not care great accessibility, E.g. a letdown of a knot avoids any entree to files that were placed in that knot.

4- CLUSTER MANAGEMENT SYSTEM

MOSIX clusters consist on a servers and nodes that connect together and are administered by master and have the similar version of MOSIX, each node in the cluster having information about the availability and status of resources in other nodes.

4.1- CONFIGURATION OF SINGLE CLUSTER:

The main feature in a MOSIX cluster is the determination of the nodes that are participating in the cluster on the way to find the participating nodes, the nodes in the cluster through the IP address will be consecutive from each other, so they will have a constant IP address. [7] There may be some holes in a cluster, but still we can determine the participant nodes by knowing the full range of the cluster. MOSIX also has a feature that automatically detects participating nodes that you just have to run "mos_autoconf", and it will find nodes in the local TCP / IP subnet, but this command will run only if all the nodes are running or if the cluster is too large it will not be able to detect them.
To improve efficiency of process relocation for each category of node, you can define whether the node is remote or near the node you are trying to configure. The reason behind this is that it is better to jam the migrating nodes when the network is slow. This will take CPU time, but will reduce network volume and time. There are some nodes which is known as Aliases Nodes that are connected to multiple networks and will have multiple IPs. Then there will be a possibility that another node message

may arrive at that node with an IP address that is different from the IP used to deliver messages from that node. These nodes allow MOSIX to differentiate between the IP addresses. So, that valid messages can arrive and associate them with one of its configured nodes.

4.2- Configuration of Multiple Cluster:

The MOSIX clusters are configured to work together and also run the same MOSIX version. The MOSIX cloud represents the organization and each cluster of this cloud will have its own owner. [7] It contains information about the status and availability status of all nodes connected together through different groups in each node. Different groups can have a shared environment such as an NFS file system.

Now how to notify one cluster about another partner cluster. There is no need to be aware of all groups in the MOSIX cloud, but you should be aware of the partner groups we will use to send or receive processes. You will identify each cluster with a specific name, this name is for that specific cluster that does not need to be the same across all multi groups and then you can add some details with it. The other method of finding partner clusters is similar to that discussed above in single clusters through their IP addresses. [7]

The partner cluster relates migration between relationship processes, by default this can occur from both directions. Processes from local clusters can move to any participant cluster at any time and can also move from a participant cluster to a local cluster, but there is an option that we can only allow one direction migration. We can also set some priorities, the number between 0 and 65535 is lower, the higher the priority. When a new cluster partner is defined the priority of that cluster will be 50 by default. [7]

4.3- MOSIX Process:

Processes are created by the "mosrun" command, these processes have standard Linux executions when they start but they run in an environment that allows them to be moved from one node to another and the home node to the node where this process will take place.

Configuring process speed: In the previous MOSIX the speed of the process can be detected automatically, but it is difficult to detect the speed with immense diversity in the process and their special characteristics, so you need to know the information of your computer. The process should trigger or measure the performance of real applications. [7]

4.4- Freezing Policies

There is a risk that memory will run out when too many processes are running on its home node. The process will be interchanged and performance will be reduced and in the worst case the swap space will be exhausted and the Linux kernel will start killing these processes. A possible example for this scenario is when some other clusters shut down. Which will force a large number of homecoming processes and the solution of this problem in MOSIX is to freeze such returning processes. They will resume these processes will not consume memory and more resources. [7]

4.5- Disk Space for Freezing

You must inform MOSIX where the frozen process will be located, and you will configure it under the name of the directory, all these freezing processes make the memory image in the director "/ freeze". You must ensure that this directory exists. Otherwise the cold will fail every time or you can also select different directories. [7]

5- Storage Allocation

MOSIX allocate the storage for the processes and the other working which are listed below.

5.1- Swap Space

There should be enough swap space to meet the memory demands for all processes, this process can migrate but these processes are likely to return to their home nodes for several reasons so you should consider the worst case and accordingly must assign swap location.

5.2- MOSIX Files

The MOSIX system creates and maintains a directory "etc / mosix / var" to manage many small files. [7] When there is no available disk to create those files, the Mosix operation will be interrupted. It asked if you want to make it a regular directory or a symbolic link when Mosix is first installed but you can change it later.

5.3- Freezing Space

There are several reasons that a mosquito process can be temporarily frozen manually by using the command or automatically when the load is increased or removed from another cluster. [7] Frozen process memory images are placed on disk by default if the process's memory content does not have enough space to write so it is killed to prevent it from filling up memory space or swap space. Therefore MOSIX placed a directory on disk for these frozen processes.

5.4- Private File Space

Users have an option that they can create files with the process that will migrate with them as well, the file size is smaller by 10MD, then it is kept in memory otherwise they require backing storage on disk and this system. It is the responsibility of the administrator to allocate enough space for them, the system can setup 3 different directories for the private files which are local Of processes, which are 2 for guest process from same cluster and 3 for guest process from different cluster. [7]

6- Unique Feature of MOSIX:

MOSIX provides several unique features to make processing easier and work on remote nodes without informing the user. Some of them are listed below:

1. *Resource discovery*

Resource discovery is performed by providing all nodes in all clusters about resource availability and status. This was done through the Gossip Algorithm, in this algorithm each node

regularly updates its resource status and also monitors the CPU speed, free and used memory and current load on it and it The information is sent to some randomly selected nodes, usually the same cluster nodes. [8]

2. Preemptive Process migration:

We need this for multiple points. For load balancing we require process migration that can move a process from slow to fast node. [8]Processes can be transferred from nodes that run out of free memory and cannot perform memory-related tasks. This can be done automatically or manually, copying the process's memory image and setting its run time environment. The memory image is usually compressed which will improve performance.

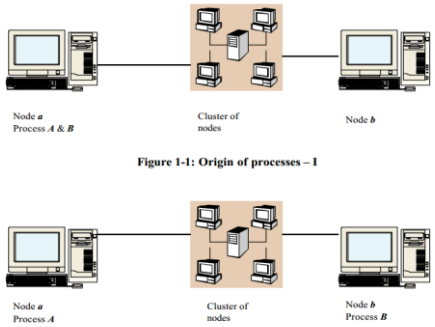

Figure 1-1: Origin of processes – I

Figure3: *MOSIX migrates the process a & b from node a to any free node in the cluster, process b to node b*

3. The run-time environment

The MOSIX software layer is implemented in such a way that applications can run on remote nodes that are away from their home node. This is done by system calls [8] In MOSIX, it's an environment in which the migrated process is also running in their home nodes will not allow the user to know where the program is running.

4. The Priority Method

This method ensures always running high-priority local processes and pushing out all processes with lower priority. Therefore the guest process will always proceed when the process of a home cluster comes in and the owners can determine which cluster they can accept the process from or from which they can block the cluster without blocking the process of the recognized cluster can do. [8] By proper management of priority settings you can share two or more clusters between users of each cluster.

5. Flood Control

It can cause flooding when the user generates a large number of processes with the hope that the program will run it somehow or allow them to occur spontaneously. This can also happen when some clusters are cut off which will cause the large process to return to their home node. [8] Some functions have been performed in MOSIX to prevent load balancing in a way that does not allow migration of the process when there is not enough memory or it sets a limit to limit the number of each local node when Limits become stricter Additional procedures are frozen.

6. Load Balancing

Migration processes are constantly trying to reduce the difference in load between node pairs. If a load imbalance occurs, it responds quickly to this and switches a system from a slower node to a faster node to a better node or the best allocation to a load node that might occur from more than one load node. Results and comparisons show that the algorithm for load balancing is only about 2 percent slower than the optimal allocation.

7. Memory Ushering

There are some hypotheses about the onset of ushering memory in that it can initiate process migration from a node with a free memory to which memory is available [8].

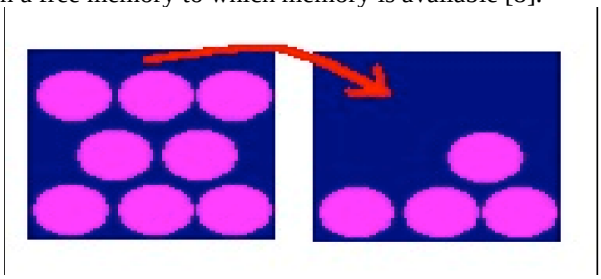

Figure4: We use the memory ushering algorithm when the free memory of a node moves below the threshold value, where the node with the lowest load survives unnecessary migration or that the node under the threshold to reduce the communication process brings

8. Decentralized Control and Autonomy

Every node in the MOSIX framework can work as independent system. Make all its control decisions independently. [8] There is no master slave relationship between nodes, allowing for a dynamic configuration in which any node with minimal interference can connect to or leave the network.

7- CONCLUSION

MOSIX is an operating system that provides gadget for sharing computational sources across clusters. Its most essential function is to provide ease of use by providing the concept of running on the same laptop with multiple processes. By maintaining the run node environment and the interface of the home node for jogging on the distant node. As a result, the consumer will not need to understand the place where the application will run. Another special attribute is computerized resource discovery, dynamic workload distribution through the migrating process, the use of a prioritization technique that will allow the use of available support for the cloud to migrate between nodes. This is very beneficial when allocating huge amounts of nodes to a cluster is unavoidable. Migration of the system is completed in a sequential manner whenever support is not available, leaving any cluster flooded and the use of MOSIX's disruptive configurations.

REFERENCES

[1] R. SUBRAMANIAN, "A TECHNIQUE FOR IMPROVING THE SCHEDULING OF NETWORK COMMUNICATING PROCESSES IN MOSIX," SOUTH GUJRAT, 1998.

[2] "MOSIX Architecture," [Online]. Available: https://www.uninet.edu/umeet/conferencias/DavidSanto/node26.html..

[3] A. Barak, A. Shiloh and L. Amar, "An Organizational Grid of Federated MOSIX Clusters," *IEEE International Symposium on Cluster Computing and the Grid,* p. 8, 2005.

[4] A. BARAK and A. LITMAN, "MOS : A Multicomputer Distributed Operating," Jerusalem.

[5] D. Abramson, A. Barak and C. Enticott, "Job Management in Grids of MOSIX Clusters".

[6] L. AMAR, A. BARAK and A. SHILOH, "The MOSIX Direct File System Access Method for," in *Cluster Computing 7, 141–150, 2004*, Jerusalem, 20014.

[7] A. Barak, MOSIX Cluster Management system, Jerusalem, 2017.

[8] A. Barak and A. Shiloh, "The MOSIX Cluster Management System for Distributing Computing on LINUX CLlusters and Muliti-Cluster Provate Clouds," Jerusalem.