An Approach for Load Balancing using Process Migration in Parallel System

Chandu Vaidya
M-Tech (2nd Year) RKNEC Nagpur (INDIA), WCE Sangli (INDIA)

M.B. Chandak
H.O.D. RKNEC, NAGPUR (INDIA)

ABSTRACT

Given paper contain proposed approached for task scheduling to achieve load balancing which is done on a group of computers. Consideration of process data part by dividing them into number of fixed part & merge into single set that as good as previous original data set. Parallelism an approach for doing jobs in amount of time i.e. very fast. The paper contains dynamic approach for process migration using thread level paradigm. Creating a thread of process into number of task, that leads to reduce total execution time of process. An algorithm is used to calculate PCB for decision purpose to achieve load balancing. We are taking CPU and MEMORY parameter in this approach. Fair share approach is considered to allocating task to every processor using preemption strategy. The MPI is used for process communication. This system has defined to reduce total execution time on onboard & between board times. Open knoppix & MOSIX platform (Middleware) are used to show the results. Prime number calculation code is used to show parallel architecture like SIMD computer. Cluster computing is way of resource managing & scheduling strategy.

General Terms

Cluster server, Middleware, Node, Resources.

Keywords

Cluster computing, MOSIX, MPI, load balancing, threads, Task load. Onboard-time, betweenbord time.

1 INTRODUCTION

The objective is to develop an algorithm for load sharing by inducing parallelism (granular programming)[6]mechanism on a group of interconnected machines. This algorithm is useful only when the cost factor can be underestimated when compared to time. The algorithm developed should be smart enough to migrate thread to other node in the cluster [3] only when the time requirement for completion of process can be reduced by doing so. Traditionally, computer software has been written for serial computation. To solve a problem, an algorithm is constructed and implemented as a serial stream of instruction. These instructions are executed on a CPU on one computer. Only one instruction may execute at a time after that instruction is finished, the next is executed. If load increase or more load is given the time requirement for execution will be more. For reducing the execution time to get output concept of Parallel Computing arises. Parallel computing uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm with the others. A computer cluster is a group of linked computers, working together closely thus in many respects forming a single computer. The components of a cluster are commonly, but not always, connected to each other through fast LAN. Clusters are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability. Load balancing [1][2]is when multiple computers are linked together to share computational workload or function as a single virtual computer. Logically, from the user side, they are multiple machines, but function as a single virtual machine. Requests initiated from the user are managed by, and distributed among, all the standalone computers to form a cluster. This results in balanced computational work among different machines, improving the performance of the cluster systems. Scheduling refers to the way processes are assigned to run on the available CPUs. This assignment is carried out by software known as scheduler and dispatcher. Scheduler and dispatcher operate with the help of a software known as middleware's. Middleware is computer software that connects software components or people and their applications. The software consists of a set of services that allows multiple processes running on noe or more machines to interact. The middleware we are using is MPICH2. MPICH2 is an high performance and widely portable implementation of the Message Passing Interface standard. It efficiently support different computation and communication platforms including commodity clusters, SMPs, massively parallel systems and high-speed networks.

2 BACKGROUND

This system is refined from the concept of executing of tasks using single processor. Uniprocessor system functioning includes preemptive scheduling scheme. We change this by using multiple processors to execute a particular task in proportional manner to reduce time to execute the task in relatively short time. The processors are connected with each other in a Cluster, such that it is viewed as a single coherent entity .Non-preemptive scheduling scheme is used for this project. This improves the performance of execution of tasks as compared to earlier type. This project uses fair scheduling approach for providing fair access to users. This system is an example of a distributed system [8]. This project is a scheduling system that provides allocation of system resources of one or more processor sets among groups of processes. Each of the process groups is assigned a fixed number of shares, which is the number that is used to allocate system resources among processes of various process groups within a given processor set. The described fair share scheduler considers each processor set to be a separate virtual computer.

Cluster computing[9][10] (or the use of computational Clusters) is the application of several computers to a single problem at the same time usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. A Cluster can provide significant processing power for users with extraordinary needs. Animation software, for instance, which is used by students in the arts, architecture, and other departments, eats up vast amounts of processor capacity.

Description:

The main function of client/user is to submit the process in the process pool related with a processor. The processes in the process pool are waiting for the execution. From these processes the higher priority process is selected by using the appropriate scheduler and is given to the Cluster server. The process division is a function that divides the process into the pieces or threads.

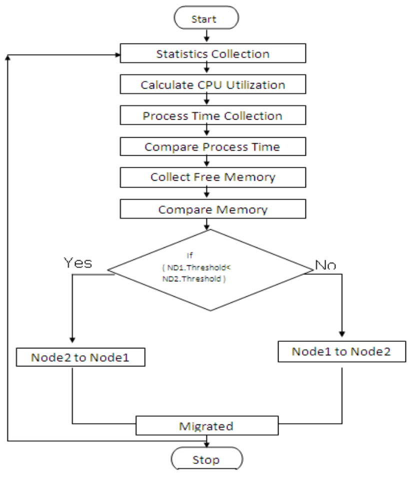


Fig-Overall model.

Thread distribution distributes these threads proportionally among the several nodes in the Cluster network [10]. Thread execution is a function that executes each thread independently on different nodes. The threads are executed using Fair-Share Scheduling[20]. It allocates equal CPU time for each node. While executing threads, the resources required for the execution of that thread on the node, the load on the node and the complexity of each threads are taken into account. Each node may require same or different resources for the execution of the thread. These resources must be provided to each node. Above simple model(fig) show the general idea regarding project. Finally the output from each node is combined and the final output is given to the Cluster server.

2.1 Related Work

Migration can be achieved at various levels in a system. It can be implemented in the operating system, as was the case with many monolithic operating systems, such as MOS(IX) [7][8], Locus [12], Sprite [11],etc. Then there are examples of migration for microkernel's, such as the V kernel [13] and Mach [14] There are also examples of user level migration implementations such as in Condor [15], and on top of UNIX [12][16]. In these systems, migration was designed outside of the kernel, independent of applications.

Plenty of researchers have proposed and most of the, has been implemented scheduling algorithms [17], [18], [19] for parallel and distributed systems, Cluster computing, as well as for Grid computing environment. For a dynamic load-balancing algorithm, it is unacceptable to frequently exchange state data because of the high overheads. In order to reduce the total execution time among cluster. Proposed a decentralized load-balancing algorithm for a cluster environment. Although this work attempts to include the communication time between two nodes during the scheduling process on their model, it did not consider the actual cost for a job transfer. Our approach takes the job migration cost into account for the load-balancing decision. A sender processor collects status information about neighboring processors by communicating with them at every load-balancing instant. This can lead to frequent message transfers among the node.

3 IMPLEMENTATION

Implementation purpose some steps are considered to achieve this project, which are as follows.

3.1 BASIC STEPS

3.1.1 CLUSTERING/HIGH PERFORMANCE

It's first step in our project in which we are going to deal with cluster computing i.e. connect more than one computer together to perform high performance task .the main purpose of cluster computing is resource utilization, where applications have traditionally used parallel or distributed computing platforms. Simple LAN and Cluster are two different things that are application specific. High performance clusters, which are also referred to as computational cluster systems. These systems are normally utilized to support very large data volumes (of computational processing). In such an environment, a parallel file system distributes the processing resources across the nodes, Load balancing clusters distribute the workload as evenly as possible across multiple server or small computer systems, such as web or application servers, respectively.

OpenMOSIX [7], [8] was a free cluster management system that provided single-system image (SSI) capabilities, e.g. automatic work distribution among nodes. It allowed program processes (not threads) to migrate to machines in the node's network that would be able to run that process faster (process migration). It was particularly useful for running parallel and intensive input/output (I/O) applications. Diagram contains the result of cluster configuration (MOSIX) among three nodes with their IP addresses.

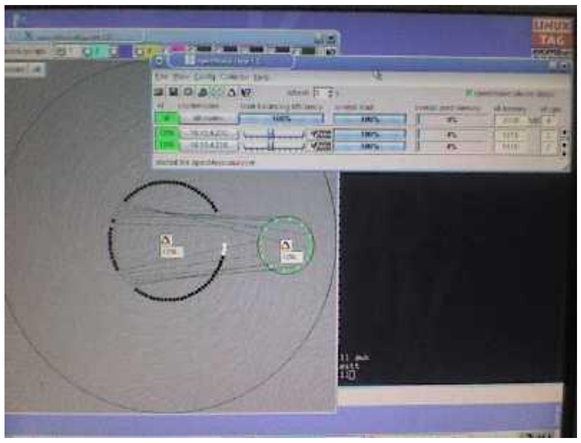
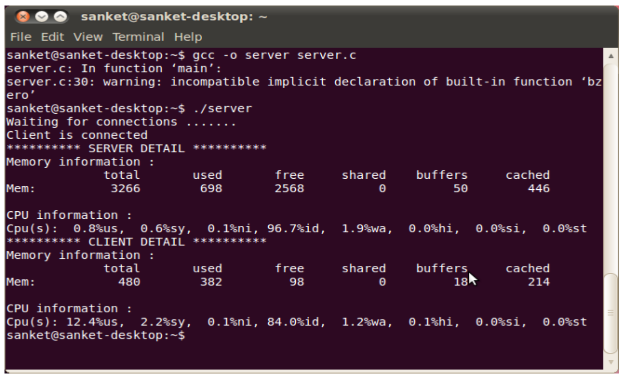


Fig-MOSIX 3 Node Cluster SnapShot.

3.1.2 STATISTICAL COLLECTION

This is second step of our project in which we are going to collect some statistical information in regular interval dynamically, only for a purpose where that collected information can used somewhere for taking decision .Here we got successes to collect statistical information .This information is about the processor and memory related for decision purpose which is over loaded and which under loaded only to load balancing. Main contain is CPU utilization and MEMORY information. There is several way for calculating the processor information like *top* and *free* command. The syntax for this command is and result store in text file for further requirement.

```
system ("top -n 1 | grep Cpu > cpu_stats.txt")
```



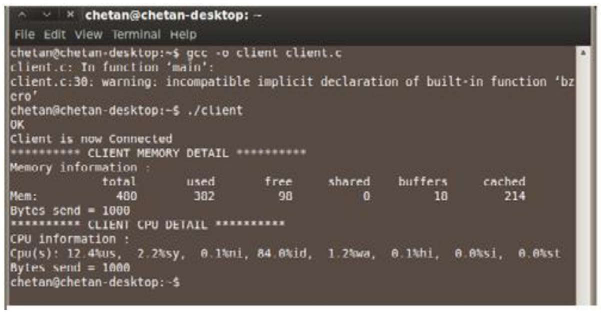
```
sanket@sanket-desktop:~$ gcc -o server server.c
server.c: In function ‘main’:
server.c:30: warning: incompatible implicit declaration of built-in function ‘bz
ero’
sanket@sanket-desktop:~$ ./server
Waiting for connections .......
Client is connected
********** SERVER DETAIL **********
Memory information :
             total       used       free     shared    buffers     cached
Mem:          3266        698       2568          0         50        446

CPU information :
Cpu(s):  0.8%us,  0.6%sy,  0.1%ni, 96.7%id,  1.9%wa,  0.0%hi,  0.0%si,  0.0%st
********** CLIENT DETAIL **********
Memory information :
             total       used       free     shared    buffers     cached
Mem:           480        382         98          0         18        214

CPU information :
Cpu(s): 12.4%us,  2.2%sy,  0.1%ni, 84.0%id,  1.2%wa,  0.1%hi,  0.0%si,  0.0%st
sanket@sanket-desktop:~$
```

Fig-Result of processor(Statastic)

system ("free -m > mem_statc.txt")



```
chetan@chetan-desktop:~$ gcc -o client client.c
client.c: In function ‘main’:
client.c:30: warning: incompatible implicit declaration of built-in function ‘bz
ero’
chetan@chetan-desktop:~$ ./client
OK
Client is now Connected
********** CLIENT MEMORY DETAIL **********
Memory information :
             total       used       free     shared    buffers     cached
Mem:           480        382         98          0         18        214
Bytes send = 1000
********** CLIENT CPU DETAIL **********
CPU information :
Cpu(s): 12.4%us,  2.2%sy,  0.1%ni, 84.0%id,  1.2%wa,  0.1%hi,  0.0%si,  0.0%st
Bytes send = 1000
chetan@chetan-desktop:~$
```

Fig-Result of Memory(Statastic).

3.1.3 *SETTING THRESHOLD/MAKING DECISION*

In this step we are considering two parameter i.e. free memory and CPU utilization .we are setting the CPU utilization up to some percentage like 80% or more, so that we will be in the position to take the decision for process migration that we are going to deal in next step.

Once we get statistical information so we need to take exact decision on that threshold. Generally load balancing purpose we need this mechanism *Collection of statistical information, Decision making, Data migration.*

Decision code

Calculation of the server CPU utilization:

```
if server is overloaded then
        t1=compute the time required to complete task on server
        t2=compute the time required to complete task after sharing threads with the nodes
/* Check t1 with t2 */
if (t1>t2) then
        Migrate threads to nodes
else
        no process migration will occur.
```

3.1.4 *THREAD CREATION*

Process thread creation is a mechanism of separating data part from process. Our idea is somewhat about to use multithreading, hyperthreding. Pthread are used to create such process chunks like child processes. Process distribution on multicore processor is very big deal in today's era. Actually the problem on physical and logical processor threads distribution. We are using one SIMD application to demonstrate this mechanism.

Before understanding a thread, one first needs to understand a UNIX process. A process is created by the operating system, and requires a fair amount of "overhead". Processes contain information about program resources and program execution state, including Process ID, process group ID, user ID, and group ID Environment Working directory, Program, instructions, registers, Stack, Heap ,File descriptors ,Signal actions ,Shared libraries ,Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory).

routin's.

pthread_create (thread,attr,start_routine,arg)
pthread_join()
pthread_exit (status)
pthread_cancel (thread)
pthread_attr_init (attr)
pthread_attr_destroy (attr)
OpenMP+MPI[4][10] is an implementation of multithreading, a method of parallelization whereby the master "thread" (a series of instructions executed consecutively) "forks" a specified number of slave "threads" and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors. Given diagram showing the main process and different threads.

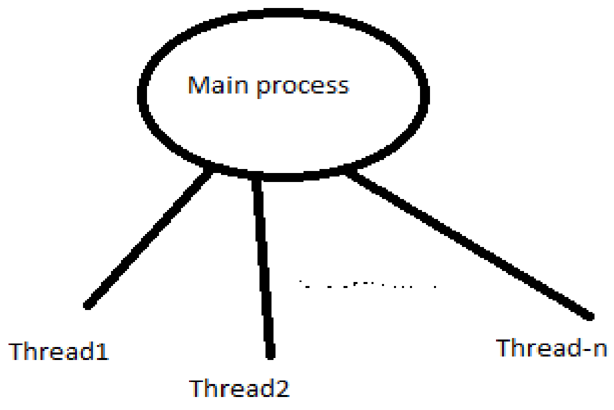


Fig-Thread Creation.

3.1.5 *PRIME NO. GENERATION USING MPI (THREAD CREATION)*

We design this algorithm for thread creation

Algorithm

```
1] Start.
2] Input no. of tasks.
3] Calculate Rank of each Task.
4] i) calculate Stride distance using ,
      stride= 2 * tasks;
   ii) Fork the tasks & assign start to each Task using,
      start = rank  * 2 + 1;
5]for each task,
   i] calculate largest prime no.
   ii] store it in slarge, & increment  prime count;
```

6]synchronize outputs of all task & calculate largest prime no.
7]finalize output & display time as,
i) Total time.
ii) Communication time.
I] On board communication time.
II] Between board communication time.
iii) Execution time.
8] Stop.

3.1.6 *PROCESS MIGRATION*

If we *transfer* the state of a process from one machine to another, we have to *migrated* the process. Process migration is most interesting in systems where the involved processors do not share main memory, as otherwise the state transfer is trivial. A typical environment where process migration *is* interesting is autonomous computers connected by a network. some consideration of process migration are Who initiates the migration, What portion of the process is migrated, State Migration, Address Space Migration ,Considering queue length etc.

Regarding process migration we used PID of that respective process.

Syntax:-"migrate {pid} {hostname or IP-address or node-number}"

Syntax : migrate {{pid}|-j{jobID}} {node-number|IP-address|host}

```
Migration of threads to node:
    n1=node1 threshold
    n2=node2 threshold
    if (n1<n2)
        Migrate process to n1
    else
        Migrate process to n2
```

3.1.7 *PROCESS EXECUTION*

By default, each thread executes the parallelized section of code independently. "Work-sharing constructs" can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both task parallelism and data parallelism can be achieved using OpenMP +MPI.

After process migration in other processor or buddy processor that time we need to consider the scheduling mechanism whether that processor are allowing are not to execute respective thread.

Algorithms used

```
Algorithm schedule process
Input: none
Output: none
{
while (no process picked to execute)
  {
    for (every process on run queue) pick highest priority
    process that is loaded in memory;
        if (no process eligible to execute)
            idle the machine;
/* interrupt takes machine out of idle state */
  }
remove chosen process from run queue;
switch context to that of chosen process, resume its execution;
}
```

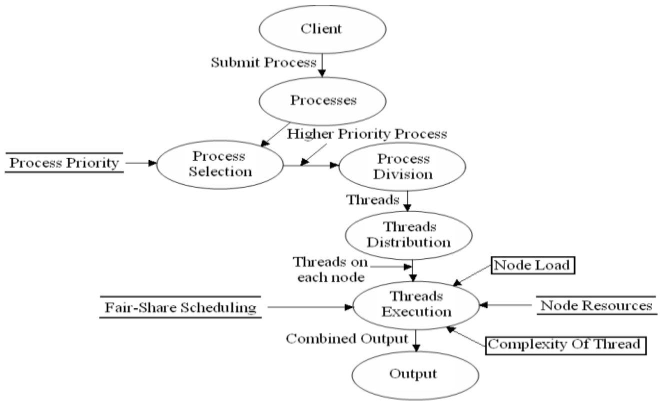


Fig-Execution Sinario.

3.1.8 *COLLECTING BACK & MERGE*

After execution of thread on different processor we need to merge together so data divided part will combine in one uniform set. we can manually migrate the processes (using PID) of all users send them on other nodes and bring them back at home. Like move them to other nodes freeze or unfreeze (continue) them, overriding the MOSIX[7][8] system decisions as well as the placement preferences of users. Even though as the Super-User you can technically do so, you should never kill (signal) guest processes. Instead, if you find guest processes that you don't want running on one of your nodes, you can use "migrate" to send them away (to their home-node or to any other node).

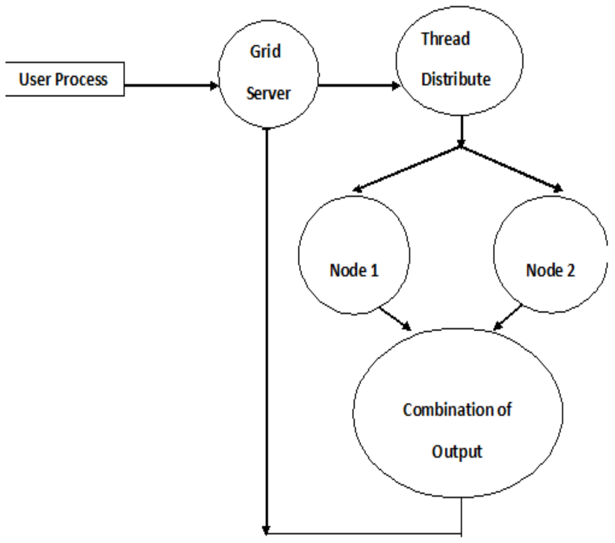


Fig-Combination of Thread

3.1.9 *ANALYSIS*

For Demonstration purposed we select on task that is finding largest prime no. up to 25000000 as well as total number of prime numbers

Time taken by Standalone Computer for above problem = **750.87 sec** we want to reduce this huge execution time by using our approached

Between board communication time is nearly constant for any number of tasks. Because system bus speed between board is nearly constant at any time. **On-board** communication time varies with the no. of tasks.

consider

obt α n
Where,
obt = onboard communication time.
n = number of tasks

. Total communication time (tcm) varies with no. of tasks.

tcm = obt + bbt

Total Execution time is calculated as :

Total time = communication time + Execution time
= (Onboard time + Between Board time) + Execution time

Consider,

Tob=On Board time
N=no. of tasks
Texe= execution time
Tob α N
Texe α 1/N

Our simple model contains the basic architecture of the project that indicate how the flow of project goes. We conclude that implementation of task scheduling which lead to fair share process allocation and load balancing as well as the total execution time. We have been able to collect the information of all nodes in Cluster environment. We have been able to perform load balancing by considering available resources such as free memory and CPU utilization for migratable processes. This increases the performance of the Cluster by decreasing the execution time for the processes. For testing purpose we have select Prime number generation program using MPI programming its. good enough to success this approached. Deployment of our approached on kernel code so that we will in the position to developed one component module that will beneficial to someone.

3.1.10 DEPLOYMENT ON KERNEL CODE

In this step we decided two possible method Daemon tool which is a system that load at start time of system and another one is direct deployment of code on kernel. Deployment purposes their so many methods are available. Regarding compilation of kernel code cross tools are available in market. We need not to compile entire kernel code, only the component that we want to merge that need to compile or rebuilt. Advance Packet Tool (APT) is also available to code deployment.

1. Daemon tool

2. Kernel Module

The kernel modules can use two different methods of automatic loading. The first method (modules.conf) is our preferred method.

- **modules.conf** - This method load the modules before the rest of the services.
- **rc.local** - Using this method loads the modules after all other services are started

Cofiguration steps

```
# modules.conf - configuration file for loading kernel modules
# Create a module alias parport_lowlevel to parport_pc
alias parport_lowlevel parport_pc
# Alias eth0 to my eepro100 (Intel Pro 100)
alias eth0 eepro100
# Execute /sbin/modprobe ip_conntrack_ftp after loading
ip_tables
post-install ip_tables /sbin/modprobe ip_conntrack_ftp
# Execute /sbin/modprobe ip_nat_ftp after loading ip_tables
post-install ip_tables /sbin/modprobe ip_nat_ftp
```

There are a few commands that allow you to manipulate the kernel. Each is quickly described below.

depmod - handle dependency descriptions for loadable kernel modules.

insmod - install loadable kernel module.

lsmod - list loaded modules.

modinfo - display information about a kernel module.

modprobe - high level handling of loadable modules.

rmmod - unload loadable modules.

4 ACKNOWLEDGMENTS

Success is the manifestation of diligence, inspiration, motivation and innovation. I attribute my success in this venture to my seminar guide Prof. M. B. Chandak, (HOD) who showed the guiding light at every stage of my seminar preparation.

I indebted to Dr .N. V Thakur, M-Tech Coordinate of the Department Computer Science & Engineering, who has provided facilities and the infrastructure to work at an extended ends.

Last but not the least I am also thankful to all the faculty members for helping directly or indirectly to accomplish the seminar work. I would like to thank My Mind for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

5 REFERNCES

[1] M. Willekk-Lemair and A.P. Reeves, Strategies for dynamic load-balancing on highly parallel computers, IEEE Transaction on Parallel and Distributed Systems, (4)9, September 1993, Pages 979-993.

[2] M. Wu and W. Sbu, A load balancing algorithm for n-cube, Proceedings of rhe 1996 Inremarwnal Conference on Parallel Processing, IEEE Computer Society, 1996, Pages 148-155.

[3] H. Shan, J.P. Singh, L. Oliker and R. Biswas, "Messge passing and shared address space parallelism on an SMP cluster," Parallel Computing, vol 29, 2003, pp. 167-186.

[4] W. Pan, L. Chan, J. Zhang, Y. Li, L. Wan and F. Xia, "Research on MPI+OpenMP hybrid programming paradigm based on SMP cluster," Application Research of Computers, vol. 26, 2009, pp. 4492–4594.

[5] Calvin Lin, "Priciples of parallel programming," China machine press, Bejing, 2008.

[6] Oren LA'ADAN Amnon BARAK and Amnon SHILOH.Scalable cluster computing with MOSIX for LINUX .InProc.LinuxExpo'99, pages95–100,May1999.

[7] Barak, A., Shiloh, A., " A Distributed Load-Balancing Policy for a Multiwmputer" , Software-Practice and Eqerience, vol. 5, no 9, September 1985, pp 901-913.

[8] Amith R. Mamidala Rahul Kumar Debraj De D. K. Panda Department of Computer Science and Engineering" MPI Collectives on Modern Multicore Clusters: Performance Optimizationsand Communication Characteristics", Eighth IEEE International Symposium on Cluster Computing and the Grid.

[9] Douglis, F., Ousterhout, J, " Transparent Process Migration: Design Alternatives and the Sprite Implementation" , Soj ware-Practice and Experience, vol. 2, no 8, August 1991, pp 757-785.

[10] Walker, B. J., Mathew, R. M., " Process Migration in AIX' s Transparent Computing Facility" , IEEE TCOS Newsletter, Winter 1989, vol. 3(l), pp 5-7.

[11] Theimer, M., Lantz, K., Cheriton, I)., " Preemptable Remote Execution Facilities for the V System" , Proc. of the 10th ACM Symposium on OS Principles, December 1985, pp 2- 12.

[12] Milojicic, D., Zint, W., Dangel, A., Giese, P., " Task Migration on the top of the Mach Microkernel" , Proceedings of the third USENIX Mach Symposium, Santa Fe, New Mexico, April 1993, pp 273-290.

[13] Litzkow, M., Solomon, M., " Supporting Checkpointing and Process Migration outside the UNIX Kernel" , Proceedings of the USENIX Winter Conference, San Francisco, January 1992, pp 283-290.

[14] Alonso, R., Kyrimis, K., " A Process Migration Implementation for a Unix System" , Proceedings of the USENIX Winter Conference, February 1988, pp 365-372.

[15] L. Anand, D. Ghose, and V. Mani, "ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing Systems," Int'l J. Computers and Math. with Applications, vol. 37, no. 8, pp. 57-85, Apr. 1999.

[16] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "On the Design and Evaluation of Job Scheduling Algorithms," Proc. Fifth Workshop Job Scheduling Strategies for Parallel Processing, pp. 17-42, 1999.

[17] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," Proc. ThirdWorkshop Job Scheduling Strategies for Parallel Processing, pp. 1- 34, 1997.

[18] Nikolaos D. Doulamis, Member, IEEE, Anastasios D. Doulamis, Member, IEEE, Emmanouel A. Varvarigos, and Theodora A. Varvarigou, Member, IEEE " Fair Scheduling Algorithms in Grids". IEEE transactions on parallel and distributed systems, vol. 18, no. 11, november 2007