



MACHINE LEARNING BASED PROFILE DRIVEN SCHEDULING ALGORITHM

S.Padmavathi[#], K.Balakrishnan^{*}

[#]Assistant Professor, Department of Computer Science and Engineering,
Thiagarajar College of Engineering, Madurai, Tamilnadu

^{*}UG student, Department of Computer Science and Engineering,
Thiagarajar College of Engineering, Madurai, Tamilnadu
INDIA

Abstract: Clusters of commodity servers are increasingly the platform of choice for running computationally and IO intensive jobs in a variety of industries. It is expected that using clusters will reduce the average job response time. But improper submission of jobs to clusters may lead to two problems, first it leads to blocking of jobs (waiting for results from other jobs) second it leads to disturbing other jobs (i.e. other jobs may be blocked due to submission). Effective utilization of the resources in clusters can help to balance the load and avoid situations like slow run of systems. This paper addresses the principle of effective utilization of cluster resources by Machine Learning based profile driven scheduling. It avoids the above problems by allocating jobs to resources of cluster based on the SVM prediction profiling results. Some of the machines in the cluster run the IO bound jobs effectively with minimum waiting time and with the minimum execution time, while in some other machines run the CPU bound jobs effectively with minimum waiting time and with the minimum execution time. The system statistics like CPU utilization time, Memory space utilized, User and System time utilization are used as the parameters for Profiling. Job dependency analysis is used to prevent dependent jobs to keep blocking and disturbing other jobs. This paper uses the Resource profiling results based on machine learning prediction and Offline Job profiling to perform job allocation onto the resources of cluster.

Keywords: Job Profiling, Machine Profiling, Cluster Scheduling, Support Vector Machine (SVM).

I. Introduction

Scheduling is an important part of any operating system. In a typical production environment like clusters, many different jobs are submitted to clusters. Therefore, the job scheduler software must have interfaces to define workflows and/or job dependencies, execute the submitted jobs automatically by mapping the jobs onto the resources. The existing scheduling systems of cluster like Network RAM supported load sharing, combines job migrations with network RAM, which uses remote execution to initially allocate a job to the most lightly loaded workstation and, if necessary, network RAM to provide a global memory space for the job larger than it would be available otherwise and on other hand memory reservation, incorporated with dynamic load sharing, which adaptively reserves a small set of workstations to provide special services to the jobs demanding large memory allocations [4]. The CPU and memory resource sharing scheduling system analyses the available memory and CPU resources in the cluster for scheduling. If a node is identified for lack of sufficient memory space to serve jobs, then one or more jobs of the node will be migrated to remote nodes with low memory allocations [5]. The Load Balanced Distributed Computing Systems (LBDCS) defines load imbalances, by employing the load balancing algorithm which makes use of predicted load indices computed as weighted averages of the past system and application loads the cluster is balanced [6]. The utility-based scheduling policy based upon real workload data from both an auction-based resource infrastructure, and a supercomputing cluster that removes the head-of-line blocking problem. But all the scheduling systems and policies listed above lacks the resource features knowledge which eventually causes the scheduler to allocate the job to an overloaded resource. This allocation of job to overloaded resources makes the jobs to wait for a long time which minimizes the productivity, throughput and utilization of resources. This paper proposes an idea of allocating and executing the jobs on the cluster resources based on the result of the job profiling and machine learning based resource profiling to eradicate the lack of resource feature knowledge.

II. Existing Approaches

A. Handling Data Intensive Jobs:

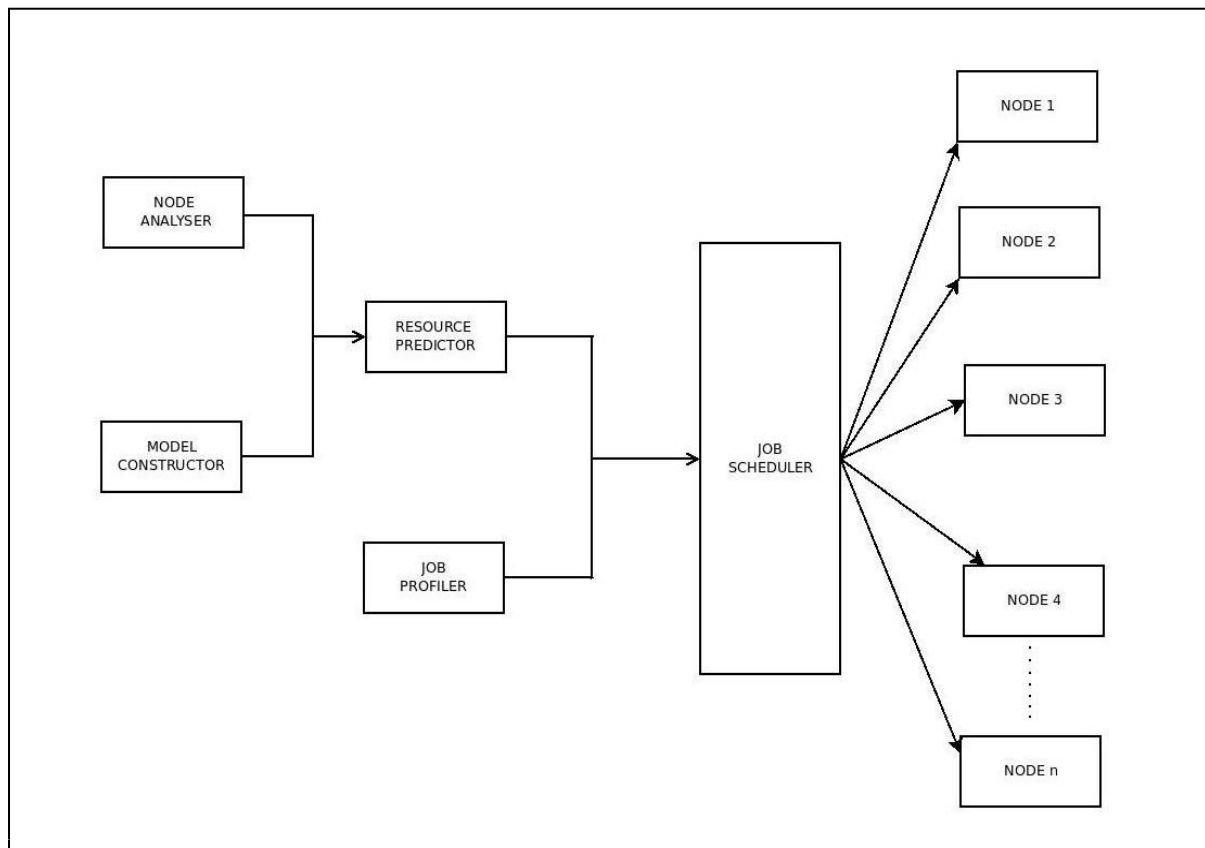
In a cluster system with dynamic load sharing support, a job submission or migration to a workstation is determined by the availability of CPU and memory resources of the workstation at the time. In such a system, a small number of running jobs with unexpectedly large memory allocation requirements may significantly increase the waiting time of the rest of jobs with less or normal memory requirements, slowing down execution of each individual job and decreasing the system throughput. This phenomenon is known as the job blocking problem because the big jobs block the running queue of majority jobs in the cluster. Since the memory demand of jobs may not be known in advance and may change dynamically, the possibility of unsuitable job submissions/migrations to cause the blocking problem is high, and existing load sharing schemes are unable to effectively handle this problem. They proposed two schemes to address this problem. The first scheme, Network RAM [4] supported load sharing, combines job migrations with network RAM, which uses remote execution to initially allocate a job to the most lightly loaded workstation and, if necessary, network RAM to provide a global memory space for the job larger than it would be available otherwise. This scheme has the merits of both job migrations and network RAM. Our experiments show its effectiveness and scalability. However, this scheme requires a network RAM facility in the cluster, which may cause additional overhead and increase cluster network traffic. In order to address this limit, this paper proposes a second scheme, memory reservation [9], incorporated with dynamic load sharing, which adaptively reserves a small set of workstations to provide special services to the jobs demanding large memory allocations. As soon as the blocking problem is resolved by the memory reservation scheme, the system will adaptively switch back to the normal load sharing state. Both schemes target on handling large data-intensive jobs in clusters, and are mutually complementary. The network RAM supported load sharing scheme can fully utilize the cluster global memory space, while the memory reservation scheme has the advantage of simple implementations and low overhead. Thus, both the techniques can be effective alternatives, and practically deployed in cluster computing under different system conditions.

B. Improving performance by sharing CPU and Memory Resources:

This work develop and examine job migration policies by considering effective usage of global memory in addition to CPU load sharing [5] in distributed systems. When a node is identified for lacking sufficient memory space to serve jobs, one or more jobs of the node will be migrated to remote nodes with low memory allocations. If the memory space is sufficiently large, a CPU-based load sharing policy will schedule the jobs. Following the principle of sharing, several load sharing alternatives are presented. The objective is to reduce the number of page faults caused by unbalanced memory allocations for jobs among distributed nodes, so that overall performance of a distributed system can be significantly improved. Trace-driven simulations are conducted to compare CPU-based load sharing policies with our policies. They showed that our load sharing policies not only improve performance of memory-bound jobs, but also maintain the same load sharing quality as the CPU-based policies for CPU-bound jobs. Regarding remote execution and preemptive migration strategies, their experiments indicate that a strategy selection in load sharing is dependent on the amount of memory demand of jobs - remote execution is more effective for memory-bound jobs, and preemptive migration is more effective for CPU-bound jobs. Our CPU-Memory-based policy uses either high performance or high throughput approach and uses the remote execution strategy and performs the best for both CPU-bound and memory-bound jobs.

C. Load Balancing in Distributed Computing System:

Load balanced distributed computing system is to improve the performance of generally underutilized timeshared workstations and highly CPU intensive independent or parallel applications [6]. It affects the initial placement of the tasks and task migrations later during their executions. One of the important implementation features of Load Balanced Distributed Computing Systems (LBDCS) is that it does not use any intermediary such as PVM (Parallel Virtual Machine) or MPI (Message Passing Interface) for intertask communication. It defines various metrics to characterize the level of load and dynamically monitors the system and applications to detect the load imbalances. The employed load balancing algorithm makes use of predicted load indices which are computed as weighted averages of the past system and application loads. Performance analysis of the system has been conducted using a number of hypothetical applications and two simple real life applications (in this case matrix multiplication and merge-sort). Hypothetical applications provide flexibility for testing the system under tunable application conditions. Using load balancing, an average speed up and efficiency close to 70 percent of their theoretical upper bounds are observed for different applications.

Figure 1 Architecture of Profile Driven Scheduler

D. Novel Based Load Balancing for I/O intensive Load :

The impact of disk I/O on overall system performance is becoming increasingly significant as more and more data intensive or I/O-intensive applications are running on heterogeneous cluster. This makes storage device a likely performance bottleneck under I/O-intensive workload. A novel load balancing algorithm [8] is used for all jobs coming to cluster. And it will balance the load in such a way that I/O, CPU and memory resource at each node can be simultaneously well utilized. The Dynamic Clustering-Based Round Robin Scheduling Algorithm [9] depends on basic round robin strategy of load sharing in dynamic clusters.

III. Cluster Scheduling based on Learned Model

The implementation of the proposed scheduling system's graphically represented in Fig. 1 Scheduling system consists of a Machine Profiler (Node Analyser, Model Constructor and Resource Predictor), Job Profiler and Job scheduler which allocate the jobs onto the resources.

A. Machine Profiling

Before the machine in cluster is profile, it is need to establish the connection between the Master node which acts as the Job Scheduler for our system and machines available in cluster for allocation and execution of jobs. This connection establishment done by using *JSch* pure java implementation of *ssh*. *JSch* helps in making *ssh* to remote nodes, executing the commands on those nodes obtaining and storing the command results. Master node fetches the client's hostname, username and password as input from Table Cluster Resource Information as in Table I and using them in establishing the connection with clients via *JSch*.

Table I Cluster Resource Information

Host	User	Password	NodeStatus	CPUIOSStatus
10.2.1.64	Root	Sagar		
10.2.1.65	Root	Sagar		
10.2.1.66	Root	Sagar		

Machine profiling involves the collection of resource features like CPU utilization time, CPU idle time, IO wait time of the CPU, User and System execution time of the CPU. All these resource features are been collected by executing the command ‘sar’ – system utility (sysstat) command on all the resources of cluster and this command results are stored in the MySQL database. There is a file which gives the entire features of the system at /var/log/sysstat/sadddd – signifies the date of the system. These resource features are stored in Resource Utilization table shown in Table II which is used latter for SVM model construction.

B. Node Analyser

The Node analyser of master node collects the present characteristics of all the resources by establishing the connection via ssh using JSch and stores them in a MySQL database. At the database, the class variables are set to determine the nature of resource. The values of the class variables are defined by following:

i. Machine Profiling for Overloaded and Non-Overloaded node

In this profiling, the resource profiled as good and bad resource. The criterion for profiling the resource is based on CPU utilization percentage and the Idle time of the CPU. If CPU utilization is less than 80% and Idle time is greater than 20%, then the system is Good. Else the system considered as Bad resource. This information is then stored in the MySQL Database with the corresponding resource information.

ii. Machine Profiling for CPU bound and IO bound nodes

In this profiling, the resources are profiled as CPU bound and IO bound resource. The criteria for profiling the resource are by memory committed and memory utilization percentage of resource. If memory committed is greater than 80% and memory utilization is greater than 80% then system is CPU Intensive, Else the system is IO Intensive. This information is then stored in the MySQL Database with the corresponding resource information. The Machine profiler results are stored in the MySQL database.

C. Model Constructor

The model constructor of the master node constructs the model from SVM data file with proper SVM data format which is created by fetching all the prerecorded resource information from the MySQL database. The SVM models are constructed based on above node analyser’s results.

D. Resource Predictor

Resource predictor of master node predicts the current resource features information against the SVM model constructed previously and the result is obtained.

Table II Resource Utilization

Node No	Node Status	CPUIO Status	User Time	System Time	Idle Time	No.of process per sec	Memory Used	Committed Memory	Swap Space Used	CPU Utilization	Power Utilized (in Watts)

Table III Jobs Info

JobName	JobNature	AssignedNode
Happy_number	CPUIntensive	
WordCount	IOIntensive	
CpuUtilization	IOIntensive	

E. Job Profiling

Profiling the job involves classifying it as CPU or I/O intensive. The clear identification of whether the job is CPU or I/O intensive cannot be found if the job is executed on loaded machine. Also machine gets loaded, if it receives data from other machines in the network. So, a machine needs to be purely dedicated for the job

profiling alone, so that there won't be interferences from any other load and further CPU cycles can be completely utilized.

Job's nature (CPU or I/O Intensive) cannot be found unless the job is made to run. If we run the job entirely, this profiling itself take more amount of time and it may result in running each job twice. So, the input file of the job is identified, by looking the *inbuilt* File related classes in job. Then, the single unit of input file is taken and is made as input file to job and then executed. To fetch the single unit of file, the shell command '*head*' has been used. The '*head*' command reads the first few lines of any text given to it as an input and writes them to standard output (which, by default, is the display screen). Then the reference of the old file in the job is changed to new file. Then the shell command '*time*' is used to run the job. The time command runs the specified program command with the given arguments. When command finishes, '*time*' command writes a message to standard output giving timing statistics about the program run. These statistics consist of

1. The elapsed real time between invocation and termination,
2. The user CPU time (the sum of the *tmsutime* and *tmscutime* values in a *structtms* as returned by *times(2)*),
3. The system CPU time (the sum of the *tmsstime* and *tmscstime* values in a *structtms* as returned by *times(2)*).

It also prints the percentage of time the CPU spends in processing. If the percentage is greater than 50%, then job is classified as CPU intensive else the job is classified as I/O intensive. As already mentioned, finding the input files needed by the job helps to prevent the bandwidth by sending the input file alone to the target machine instead of sending all the files. Therefore the total transfer time will also be minimized. Since clusters will be heavily loaded with traffic, this work can help to prevent extra traffic from being injected into the network and the already running jobs will not be disturbed. So the result of job profiling will be a list of input files used by the job and the nature of job. The nature of jobs is then stored in the MySQL table JobsInfo shown in Table III for future reference.

F. Job Scheduler

Job scheduler takes in the Machine profiler results and Job profiler results as input and allocates the job accordingly. Job scheduler initially defines which of the resource is non-overloaded and then defines which of them where CPU intensive and IO intensive, as a next step it takes in the job profiling information if a job is IO intensive then that job is allocated to the CPU intensive node and if a job is CPU intensive then that job is allocated to the IO intensive node. The allocation of jobs on to the resource is been stored in the MySQL table Jobs Info shown in Table IV.

Table IV Jobs Info

JobName	JobNature	AssignedNode
Happy_number	CPUIntensive	Node 1
WordCount	IOIntensive	Node 2
CpuUtilization	IOIntensive	Node 3

IV. Algorithm

The System of machine learning based scheduling starts with the Master node which takes the jobs and nodes as input. Algorithm proposed in the paper is as follows:

1. User submits the jobs to the Master node (acts as the Job Scheduler).
2. The Job Profiler takes the job as input and finds the input files that are needed for execution of the job. Once the input files are obtained the Job Profiler runs the job with a small part of input file and define the nature of job.
3. The Node Analyser of the Machine profiler takes in the Node's basic information required for login and executes and fetches result of sysstat command '*sar*' in each of the nodes and store the results.
4. Model Constructor fetches all the Node characteristics convert them into needed SVM data format and constructs the SVM model from the SVM data file which is used for prediction.

Figure 2 Algorithm of Profile Driven Scheduler

```

begin
while nextJob != no
    get the job details
    profiledInformation = nature(job)
endwhile

svmmodel = SVMmodelconstruction()

currentFeature = currentresourcefetcher()

predictionResult = predictResults(svmmodel,currentFeature)

if predictionResult != overloaded
then
    resource is bad
else
    resource is good

StoreDatabase(resourceInformation)

svmCPUIOModel = SVMmodelconstruction(cpuio)

predictionResult = predictResults(svmCPUIOModel,currentFeature)

if predictionResult != CPUINTENSIVE
then
    resource is IO BOUND
else
    resource is CPUINTENSIVE

StoreDatabase(resourceInformation)

nature = nature(job)

if nature == cpubound
then
    assign to IOBOUND resource
else
    assign to CPUINTENSIVE resource

Schedule the job.
    
```

5. Resource predictor takes in the current features of Nodes in cluster and predicts them against the model previously constructed.
6. The Job Scheduler takes in the Job Profiler and Machine Profiler results, makes the decision of allocating the jobs on to the resources.
7. Now the Master node sends the job's source code, input files to the allocated resource for execution.

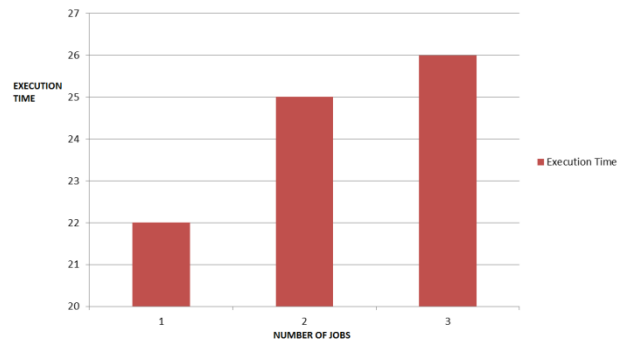
V. Results

The scheduling of jobs in the cluster is done by following the algorithm given above and the results are obtained. Based on the results the jobs and corresponding input files for the jobs are allocated to the resources of the clusters for execution. The scheduler results is been evaluated by varying the number of nodes in cluster with the fixed number of jobs and by varying the number of jobs with the fixed number of nodes in the cluster.

A. Varying the Number of jobs with fixed number of resources:

While varying the size of jobs with the fixed number of resources the execution time of the jobs increased, but it is observed that proposed scheduler minimizes the execution time when compared with the conventional scheduling algorithms and policies. The result of execution of jobs with the proposed scheduler is shown below in Fig. 3.

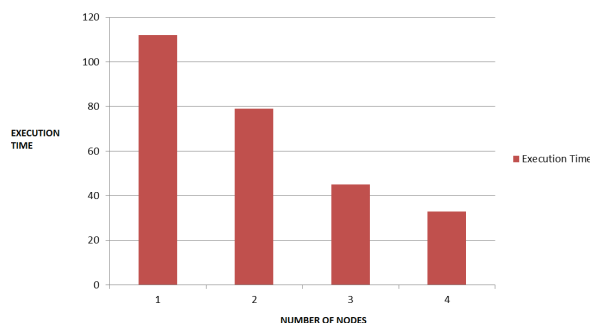
Figure 3 Profile Driven Scheduling with varying number of jobs



B. Varying the number of resources in cluster with fixed number of jobs:

When the number of nodes in the cluster is increased for fixed number of jobs it is observed that the execution time of the jobs is minimized greatly. The reduction in execution time of the jobs is directly proportional with increase in the number of resources in the cluster as shown below in Fig. 4.

Figure 4 Profile Driven Scheduling with varying number of nodes



C. Validation against Random Scheduler

The Proposed system of scheduling is shown in Fig. 6 is validated against the normal random scheduling system of linux is shown in Fig. 5. The evaluation is done based on execution time, memory utilization, CPU Utilization and on soft real time execution. The execution time of the jobs are evaluated first with 4 nodes in cluster with fixed size of job, it is observed that execution time of the jobs is very less in proposed scheduler when compared with the random scheduling algorithm of linux which is represented in Fig. 7.

The systems are evaluated with the job completion with soft deadline, it is observed that percentage of jobs completed within the soft deadline with proposed profile driven scheduling system is more when compared with the random scheduling algorithm is shown in Fig. 8. This is obtained since the overloaded processors are not overloaded again with the jobs and the CPU / IO intensive machine profiling helps in acquiring the balance in the execution of the jobs in every nodes of the cluster.

Now the CPU utilization and Memory Utilization of the nodes in cluster is evaluated. This evaluation is mainly focused on representing the load balance in the profile driven scheduling. It is observed that the random scheduling algorithm leads to lot of load imbalances with respect to both the CPU and memory because of

random allocation of resources. In the proposed profile driven scheduling avoided the load imbalances by profiling both machine as well as the job before the allocation of resources. The CPU Utilization of jobs in Random scheduler is shown in Fig. 9 and Proposed Scheduler is shown in Fig. 10 and the Memory Utilization of jobs in Random scheduler is shown in Fig. 11 and for Proposed Scheduler is shown in Fig. 12.

The experiments are done in a cluster consisting of 4 nodes connected via the Local Area Network (LAN). The execution time of the jobs in the nodes is recorded using the linux utility command 'time'. The soft deadline for jobs is been set by the user who present the job to the cluster for scheduling. The CPU and memory utilization of the machines is obtained using the sysstat command 'sar', 'mpstat', 'iostat' commands.

Figure 5 Profile Driven Scheduling with varying number of nodes in Random Scheduling

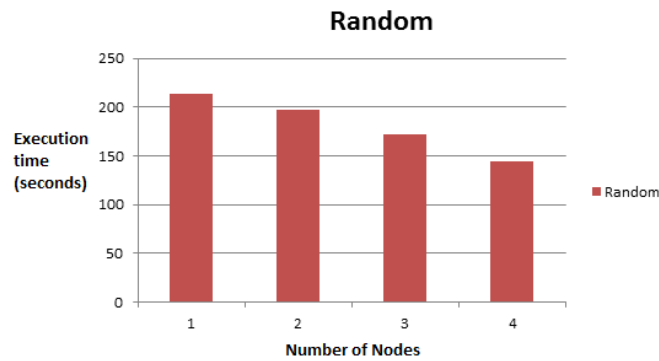


Figure 6 Profile Driven Scheduling with varying number of nodes using Profile Driven Scheduling

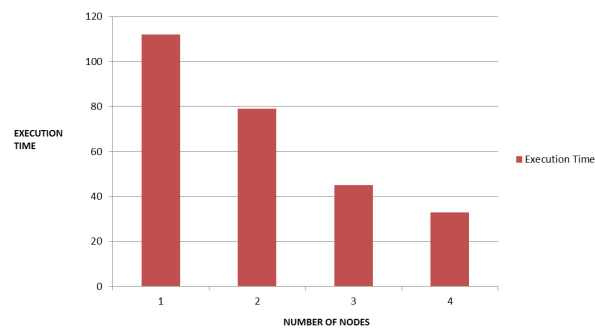


Figure 7 Random vs Profile Driven Scheduling (Execution time)

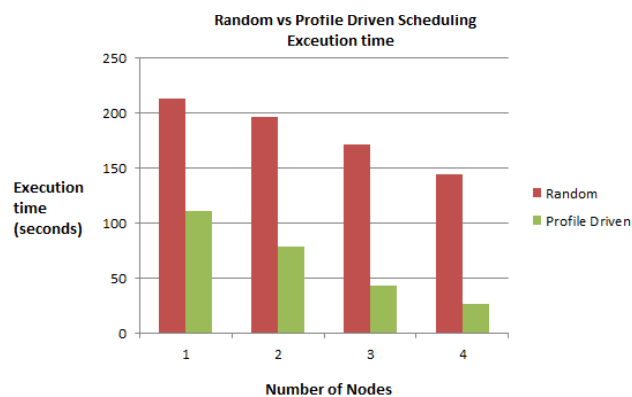


Figure 8 Random vs Profile Driven Scheduling (Percentage of Job Completion within soft deadline)

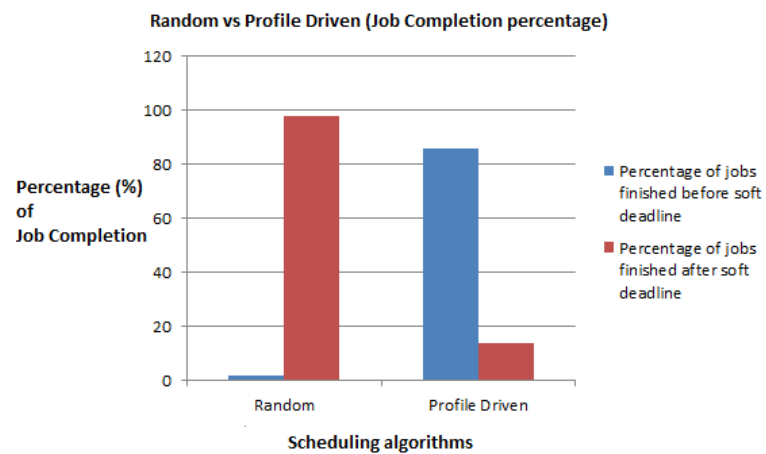


Figure 9 CPU Utilization (%) in Random Scheduling

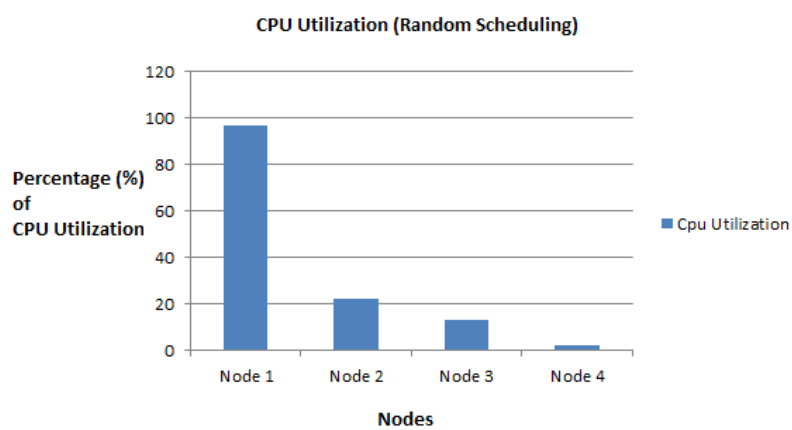


Figure 10 CPU Utilization (%) in Profile Driven Scheduling

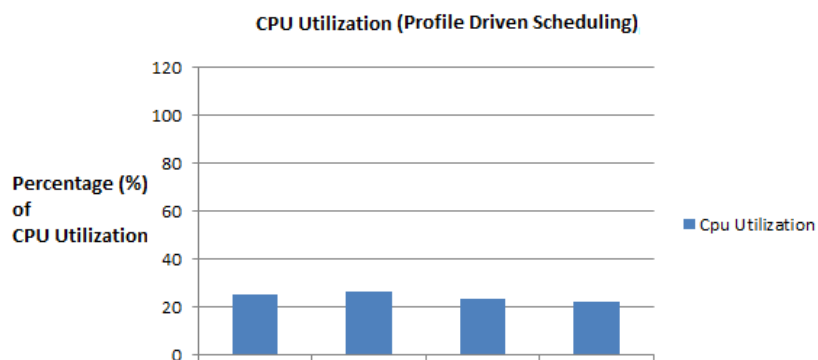


Figure 11 Memory Utilization (%) in Random Scheduling

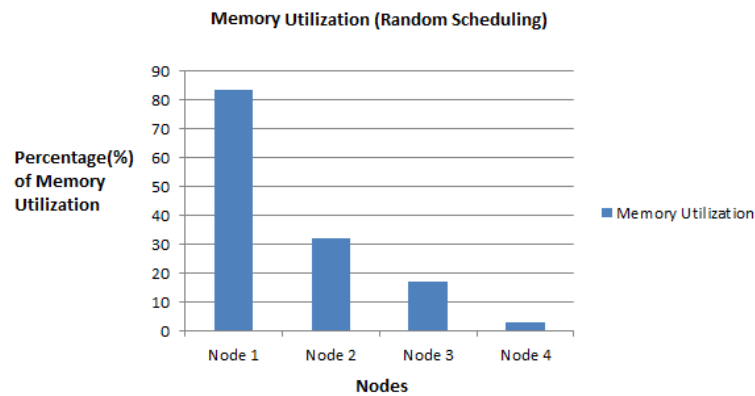
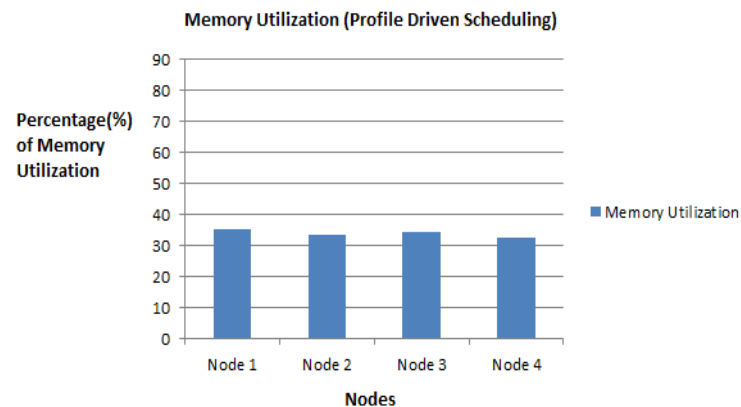


Figure 12 Memory Utilization (%) in Profile Driven Scheduling



VI. Conclusion

This paper proposed, designed and evaluated the Machine Learning based scheduler. The Machine Learned scheduler collects all the characteristics of the available resources in the cluster, based on the present and history of resource characteristics obtained the resources of the cluster predicted and the resources are profiled as overloaded & non overloaded resource and as non-overloaded CPU intensive and non-overloaded IO intensive resources and mapping the jobs accordingly onto the resources based on job profiling results. Overall the combination of the job profiling and machine profiling based on machine learning techniques, the execution time of the jobs in cluster is been minimized compared with other conventional scheduling algorithms. As a future scope the work can be extended by preventing the offline job scheduling and the profiling can be enhanced as online job profiling without executing the jobs with a part of input file.

References

- [1] Leonidas kontothanassis, David Goddeau, "A case study of profile driven scheduling for a Heterogeneous cluster of workstations", Cambridge Research Laboratory, HP Laboratories, cambridge, HPL-2004-192, october27,204.
- [2] ClementinaMarinoni, Fondazinepolitecnico di Milano, "Competence and job profiling frameworks".
- [3] JuJiubin, XuGaochao, Yang Kun, "Online predicting behaviors of jobs in Dynamic Load Scheduling" Department of Computer Science, Jilin University, Changchun.
- [4] Li Xiao, SongqingChen,Xiaodong Zhang, "Adaptive Memory Allocations in Clusters to Handle Unexpectedly Large Data-intensive Jobs", *IEEE Transactions on Parallel and Distributed Systems*, Volume 15, July 2004.
- [5] XiaodongZhang,Li Xiao, YanxiaQu, "Improving Distributed Workload Performance by Sharing Both CPU and Memory Resources", *20th IEEE International Conference on Distributed Computing Systems*, April 2000.
- [6] M. Bozyigit, J. Al-Ghamdi, M. Ghouseuddin, H. Barada, "A load balanced distributed computing system", Information Computer Science Department, King Fahd University of Petroleum Minerals, D++hahran 31261,Saudi Arabia, Volume 11 Issue 12, Pages 753 - 771, 18 Oct 1999.

- [7] Sivarama P. Dandamudi, Hamid Hadavi, "*Performance Impact of I/O on Sender-Initiated and Receiver-Initiated Load Sharing Policies in Distributed Systems*", Natural Sciences and Engineering Research Council of Canada and Carleton University, 1996.
- [8] Bibhudatta Sahoo, Pushpendra Kumar Chandra, "A Novel Load Balancing Algorithm for I/O-intensive Load in Heterogeneous Clusters", *1st International Conference on Advances in Computing*, Chikhlí, India, 21-22 February 2008.
- [9] Sanpawat Kantabutra, Parinya Kornpitak, and Chengchai Naramittakapong, "*Dynamic Clustering-Based Round-Robin Scheduling Algorithm*", The Theory Of Computation Group, Department of Computer Science, Chiang Mai University, Chiang Mai, 50200, Thailand