PERFOMANCE EVALUATION OF RESOURCE SCHEDULING TECHNIQUES IN CLUSTER COMPUTING

Admire Mudzagada, Benard Mapako and Benny M Nyambo

ABSTRACT: Resource management and job scheduling are critical tasks in cluster computing. The heterogeneity of resources causes the scheduling of an application to be significantly complicated and a challenging task in a cluster system. The main focus of this paper is to come up with a dynamic way to calculate the time-slice that each process gets based on the service or execution time for each process. The algorithm is based on the traditional round robin scheduling algorithm. The algorithm will provide fairness on scheduling since the calculation of the time-slice is based on the service time of processes in the ready queue. The total service time of all processes in the ready queue is summed and averaged to get the time-slice. As the process arrives or leaves the ready queue the time-slice will be changing dynamically. We want to come up with an algorithm that maximizes the resource utilization and minimize processing time of jobs. The criterion for performance evaluation is based on response time, waiting time and turnaround time. We want to minimize the response time and the waiting time of all process that requests for a certain service whilst increasing throughput.

Key words: Cluster computing, time-slice, service time, response time, waiting time, turnaround time.

1.0 INTRODUCTION

A cluster consists of independent computers combined into a unified system through software and networking. When two or more computers are used together to solve a problem, it is considered to be a cluster. Clusters are typically used for High Availability (HA) for greater reliability or High Performance Computing (HPC) to provide greater computational power than a single computer can provide.

A cluster system comprises of independent machines that are connected by high-speed networks and uses middlewares to create an illusion of a single system [1] and hide the complexities of the underlying cluster architecture from the users. The *cluster Resource Management System (RMS)* provides a similar interface for user-level sequential and parallel applications to be executed on the cluster system

As high-performance computing (HPC) clusters grow in size, they become increasingly complex and time-consuming to manage. Tasks such as deployment, maintenance, scheduling and monitoring of these clusters can be effectively managed using an automated cluster computing solution [1].

The RMS of clusters provides support of four main functionalities: management of resources; job queuing; job scheduling and execution. The RMS manages controls and maintains the status information of the resources such as processors and disk storage in the cluster system. Jobs submitted by the users into the cluster system are initially placed into queues until there are available resources to execute the jobs. The cluster RMS then invokes the cluster scheduler to determine how resources are assigned to various jobs. After that, the cluster RMS dispatches the jobs to the assigned nodes and manages the job execution processes before returning the results to the users upon job completion.

In cluster computing, the *producer* is the owner of the cluster system that provides and manages resources to accomplish users' service requests [2]. The *consumer* is the user of the resources provided by the cluster system and can be either a physical human user or a software agent that represents a human user and acts on his behalf. A cluster system has multiple consumers submitting job requests that need to be executed.

Mapping and scheduling of Meta-tasks in Cluster computing systems are complex computational problems. They are known to be NP-Complete except under a few special situations [3]. Solving the mapping problem is basically deciding on which task should be moved to where and when, to improve the overall performance. There is a wide variety of approaches to the problem of mapping and scheduling in Cluster Computing systems. In our approach, we revise the decision taken by the Round Robin heuristic and adjust its allocation strategy by introducing a dynamic way to calculate the time-slice in order to improve machine (processor) utilization and hence achieve better mapping performance.

2.0 CLUSTER ARCHITECTURE

A cluster is a collection of inter-connected and loosely coupled stand-alone computers working together as a single, integrated computing resource. Clusters are commonly, but not always, connected through fast local area networks. Clusters are usually deployed to improve speed and/or reliability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or reliability [4].

The architecture of interaction of entities is shown in Fig.1 below.

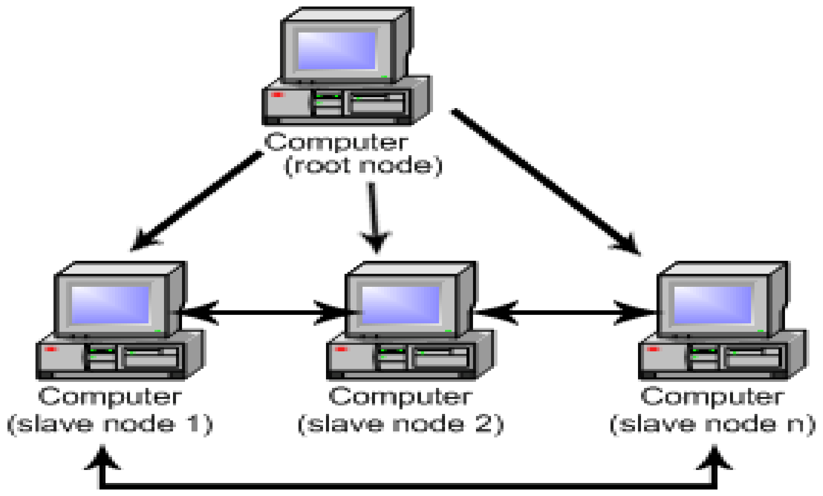


Figure 1: Cluster System Architecture

2.1 Resource Management and Scheduling (RMS)

Resource Management System: A resource management system manages the processing load by preventing jobs from competing with each other for limited computer resources and enables effective and efficient utilization of resources available.

Resource Manager: Resource Managers do basic node state monitoring, receive job submission requests and execute the requests on the slave node.

Job Scheduler: Job scheduler tells the resource manager what to do, when to run jobs, and where.

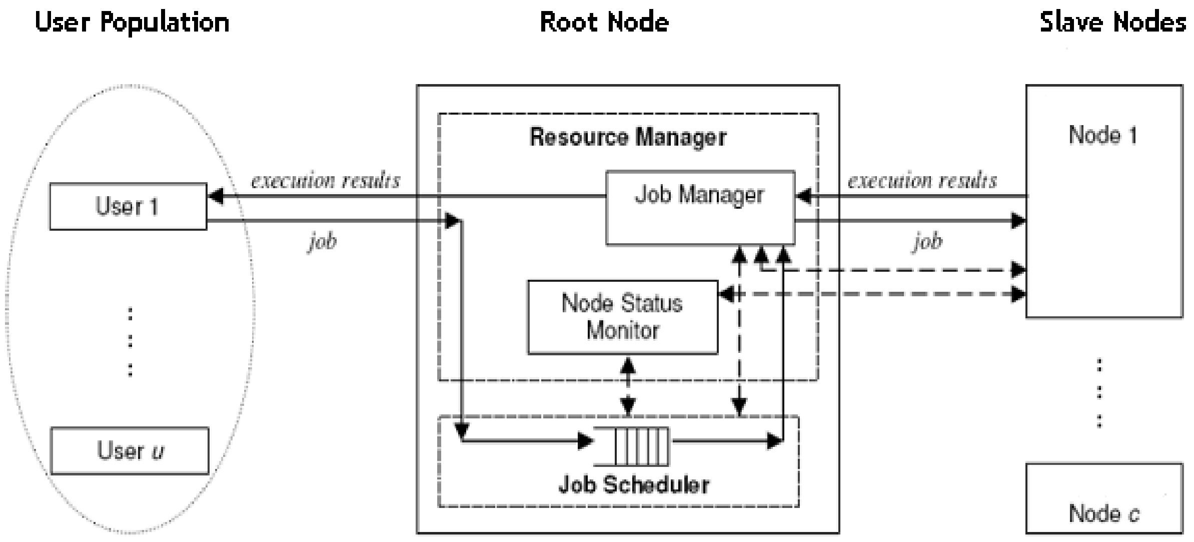


Figure 2: Architecture of Resource Management System

3.0 RELATED WORK

Scheduling and mapping are complex computational problems. They are known to be NP-Complete except under a few special situations [6]. Therefore, there is a wide variety of approaches to the problem in scheduling in Cluster systems. This variety makes it difficult to compare between different techniques since there is no uniform means for qualitatively or quantitatively evaluating them [7].
Mapping techniques can be classified in two into two major categories: **static mapping and dynamic mapping**. The difference is based on the time at which the mapping decisions are made. Static mapping means assigning processes to nodes at the compile time, while in dynamic mapping the tasks are assigned to the nodes at run time and they may be reassigned while they are running.

Although the principal advantage of the static mapping is its simplicity, it fails to adjust to changes in the system state. However, the dynamic mapper takes into consideration the state of the system (workload, queue lengths, etc.) and make use of real-time information. In general, the heuristics for dynamic mapping can be grouped into two categories: on-line (immediate) mode and batch-mode heuristics. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. While in the batch mode, tasks are collected into a set that is examined for mapping at prescheduled times called mapping events [8].

3.0 OVERVIEW OF ROUND ROBIN

Round Robin (RR) is one of the simplest scheduling algorithms for processes in an operating system. It handles all processes without any priority and the major issue in RR is the time slice. Robin scheduling is both simple and easy to implement, and starvation-free.

RR (Round Robin) is one of the oldest, simplest, fairest and most widely used algorithms. The *round-robin* (RR) scheduling algorithm is designed especially for time-sharing systems. A small unit of time, called a *time quantum* or time-slice is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating resources to each process. Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in time-sharing environments in which the system needs to guarantee reasonable response times for interactive users.

To implement RR scheduling, we keep the ready queue as a first-in, first-out (FIFO) queue of processes. New processes are added to the tail of the ready queue. The resource scheduler picks the first process, sets a timer to interrupt after 1 time quantum, and dispatches the process.

One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is greater than one time quantum, the timer will go off and will cause an interrupt to the system scheduler. A context switch will be executed, and the process will be put at the *tail* of the ready queue. The scheduler will then select the next process from the ready queue.

In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row. If a process' CPU burst exceeds 1 time quantum, that process is *preempted* and is put back in the ready queue. The RR scheduling algorithm is inherently preemptive.

Performance:-The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, if the time quantum is very large (infinite), the RR policy is the same as the FCFS policy. The other extreme is that if the quantum is very large the response time will be also high .Thus, we want the time quantum or time-slice to be a dynamic value with respect to the number of processes in the ready queue so as to minimise the number of context switches that may occur.

In Round Robin Scheduling, the time quantum is fixed and then processes are scheduled such that no process get CPU time more than one time quantum in one go. If time quantum is too large, the response time of the processes is too long such that it may not be tolerated in interactive environment. If the time quantum is too small, it causes unnecessarily frequent context switches leading to more overheads resulting in fewer throughputs [9].
In this paper, a dynamic method of calculating the time-slice has been proposed that decides a value that is neither too large nor too small such that every process has got reasonable response time and the throughput of the system is not decreased due to unnecessarily context switches. This method depends on changing time time-slice in each time when a process is executed over the queue; we call this method as ***dyn*RR** technique. In the following sections, we will define the ***dyn*RR** terminology and explain its implementation.

4.0 THE PROPOSED ALGORITHM

Before the scheduling strategy for the proposed algorithm is introduced, we first define several terms and introduce some notation used in the algorithm:

- **P*i*** – processes in the ready queue requesting resources
- **R*j*** – the resources available for scheduling the processes

Suppose we have n resources **R*j* *(j = 1... n)*** have to process ***m*** processes **P*i* *(i = 1... m)***. A schedule for each process **P*i*** is an allocation of one or more time intervals to one or more resources [10]. This can be viewed as matrix with rows representing the number of processes and the columns representing the available resources (cluster nodes).
This can be viewed as matrix with rows representing the number of processes and the columns representing the available resources (cluster nodes).

$$\begin{array}{llc} & & \textbf{Resources} \\ & & R_1 \ \ldots\ldots\ldots\ldots\ldots\ldots \ R_n \\ & P_1 & \left(\begin{array}{c} P_1R_1 \ldots\ldots\ldots\ldots\ldots\ldots P_1R_n \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \\ P_mR_1 \ldots\ldots\ldots\ldots\ldots\ldots P_mR_n \end{array} \right) \\ \textbf{Processes} & & \\ & P_m & \end{array}$$

Figure 3: Matrix Representation of Processes and Resources

The ***dyn*RR** algorithm begins with the set of all unassigned jobs/processes. Initially, the execution time is found for each process in the ready queue. The first process in the ready queue is chosen and assigned to the corresponding resource based on the computed time-slice. If the process completes the process is then removed from queue else the processes is added to the back of the ready queue and the process is repeated until all processes in the queue are mapped.
Suppose we have **m** processes in the ready queue requesting for resources.

$$\mathbf{P_1 + P_2 + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + P_m}$$

The total execution time can be computed as

$$T_{exec}(Pi) = \sum_{i=1}^{m} exec(Pi)$$

(**Note:** The total execution time is computed based on the number of processes in the ready queue)

To obtain the time-slice each process can get, equation above can be multiplied by a dynamic figure of

- $1/_{dyn}(m)$

Where *dyn* **(m)** is the number or processes in the ready queue

T$_{\text{exec}}$ (**P*i***) is the total execution/service time of all processes

The dynamic time-slice will then be computed as

$$Ts = \sum_{i=1}^{m} {}_{exec}(Pi) * (1/{}_{dyn}(m))$$

Figure 4 below shows the flow chart of the proposed algorithm.

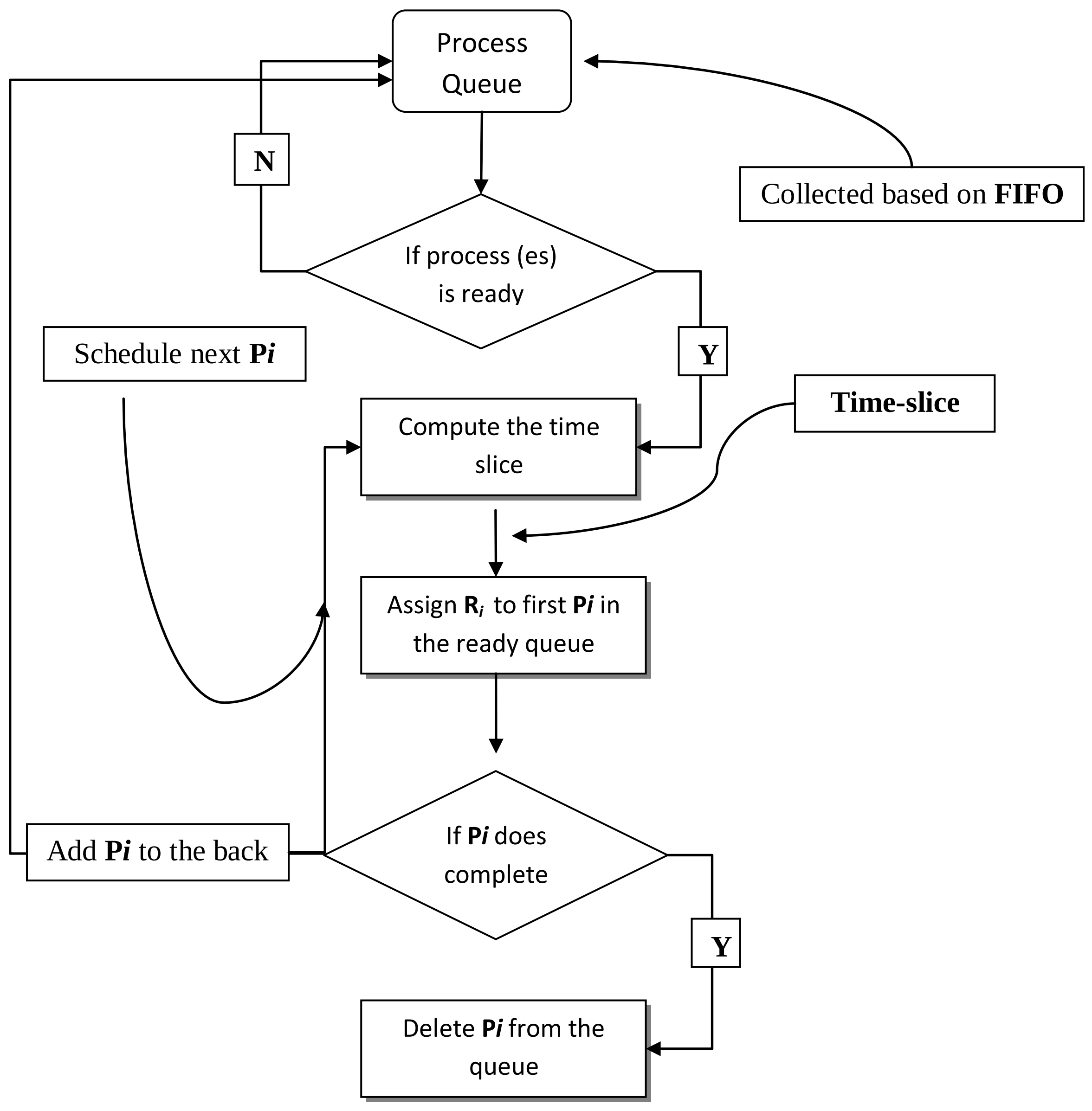

Process Queue
Collected based on **FIFO**
N
If process (es) is ready
Y
Schedule next **P*i***
Compute the time slice
Time-slice
Assign **R***ᵢ* to first **P*i*** in the ready queue
Add **P*i*** to the back
If **P*i*** does complete
Y
Delete **P*i*** from the queue

Figure 4: The general operation of the Proposed Algorithm

5.0 SIMULATION

Graphical Environment

The graphical environment was implemented using NetBeans 3.4.1 compiler. The compiler is composed of a configuration and simulation execution interface. Using JAVA NetBeans tools, it is possible to model, execute, save and modify simulation environments and experiments. Several resource scheduling algorithms can be implemented and their results can be collected and analyzed.

The Data

This simulator simulates processes arriving in a job queue. A process is some abstract job performed by a user [11].

Simulation Inputs

- Initial burst time: This is the amount of CPU time the process will require. In real life this is not really known, but can be predicted with some degree of accuracy.
- Delay: The time separating the arrival of processes.

Tracking Data

- PID: A Process ID is a number used to identify a particular process. Each PID is unique to a process.
- Arrival, start and finish time: Record the time a job arrives, when it is started, and when it is finished. These times allow us to derive three quantifiers: response time, turnaround time and waiting time.

Statistical and Performance Module

- For each simulation executed, the statistical analysis and performance module of the simulation creates a log with the calculation of several metrics based on the jobs executed. The main calculated metrics are: tasks or jobs response time; wait, submission, start and end time of each task; mean jobs reaction time and throughput.

5.1 PERFORMANCE EVALUATION

Scheduling Criteria [12]: scheduling criteria are the basis on which the performance evaluation of Cluster scheduling algorithms is evaluated. There are many possible criteria

- **Waiting Time:** This is the amount of time spent by a process in the ready state. It is the difference in start time and ready time. Usually, the goal is to minimize the waiting time.

  Waiting Time = turnaround Time – Burst Time

- **Response Time:** This is the amount of time between submission of requests and first response to the request. Usually, the goal is to minimize the response time.

  Response Time = Start Time - Arrival Time

- **Turnaround Time:** Turnaround time is the total amount of time that a process is either running or waiting. Its lifetime.

  Turnaround Time=Execution Time+ Waiting Time

Table 1 and Table 2 below shows sample simulation results obtained after a simulation of 10 processes. Simulations results are calculated in Table 4 below.

PID	Burst	Priority	Arrival	Start	Finish	Wait	Response	Turnaround
1	32	9	5	5	36	0	0	32
2	15	7	15	48	62	33	33	48
3	3	5	32	37	39	5	5	8
4	8	1	34	40	47	6	6	14
5	83	3	49	63	145	14	14	97
6	12	3	64	146	157	82	82	94
7	57	4	72	204	260	132	132	189
8	13	5	81	158	170	77	77	90
9	33	1	98	171	203	73	73	106
10	76	3	109	261	336	152	152	228

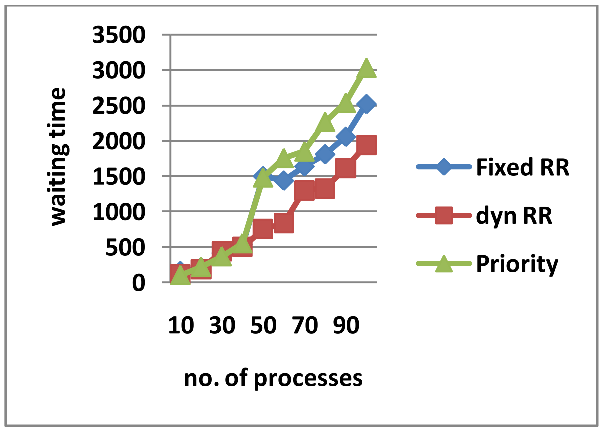
Table 1 : Sample Simulations Results

	Wait	Response	Turnaround
Min	0	0	8
Mean	57.4	57.4	90.6
Max	152	152	288
StdDev	54.68	54.68	72.1

Table 2 : Sample summary of Simulation Calculations

6.0 SIMULATION RESULTS

The ***dyn*RR** algorithm combines the benefit of low overhead round-robin scheduling with low average response time and low average waiting time. The following figures explain our improvements of our proposed algorithm over the RR algorithm and other scheduling algorithms.



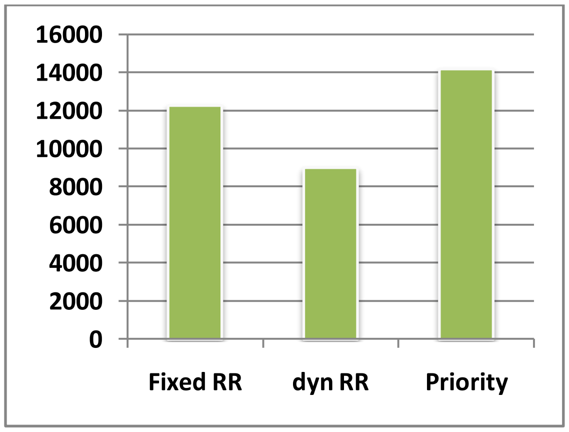
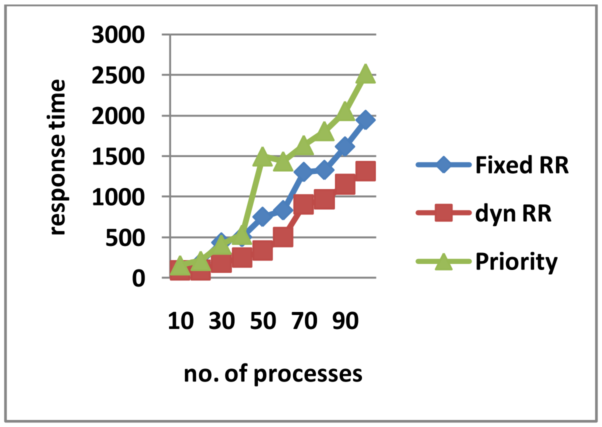


Fig 6.1 : Waiting Time **Fig 6.2 : Cumulative Waiting Time Graph**



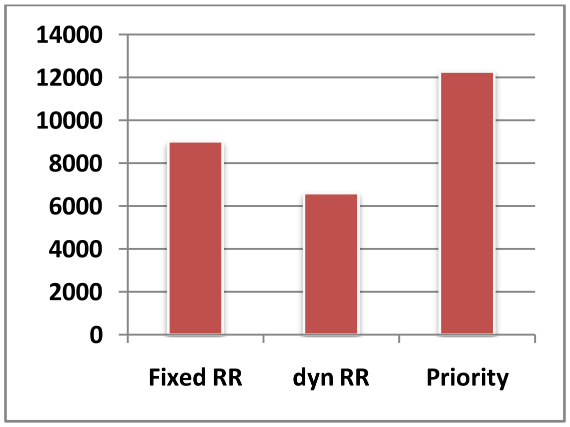
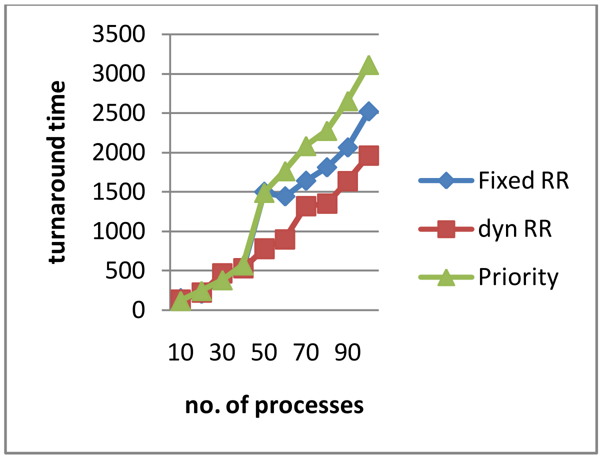


Fig 7.1 : Response Time **Fig7.2 : Cumulative Response Time Graph**



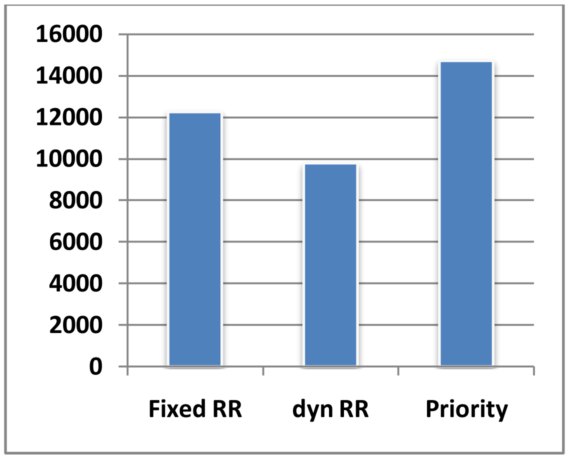


Fig 8.1 : Turnaround Time **Fig 8.2: Cumulative Turnaround Time Graph**

7.0 CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this paper, the problem of resource scheduling of jobs/processes in Cluster computing has been discussed. Three main goals of scheduling that are often desired were discussed. In light of the effectiveness and the efficiency of the RR algorithm, we developed a new cluster resource scheduling based on RR named Dynamic Round Robin (***dyn*RR**), and we described this technique and its improvement over the RR. We have simulated the ***dyn*RR** algorithm together with other scheduling algorithm for efficient comparison of performance. The algorithms simulated are the fixed Round Robin and the priority based algorithm. The algorithms were compared using different performance evaluation criteria. From the simulation study, we get an important conclusion; that the performance of ***dyn*RR** policy is higher than that of RR in general cluster computing system. Both analysis and simulation show that the ***dyn*RR** performs better than the fixed quantum Round Robin.

7.2 Future Work

The work presented in this paper can be expanded in many directions. The proposed algorithm can also be integrated with the priority based algorithm. This can be done by adding the time-slice that each process gets with the priority unit value of each process. The process will submit its priority value (0 to 9). A process with highest priority is identified by a value of 0 and a process with the lowest priority is identified by a value of 9.The process with a priority value 0 is given 10 units which will be added to the calculated time-slice and one unit is assigned to a process with lowest priority.

8.0 REFERENCES

[1] Buyya R, Cortes T, Jin H. Single system image. *The International Journal of High Performance Computing Applications* 2001; **15**(2):124–135.

[2] Chee Shin Yeo and Rajkumar Buyya. *A taxonomy of market-based resource management systems for utility-driven cluster computing* 2006; **36:** 2

[3] Amal S. Khalifa, Reda A. Ammar , Tahany A. Fegrany, Mohamed E. Khalifa. *A preemptive version of the Min-Min heuristics for dynamically mapping meta-tasks on a distributed heterogeneous environment;* 1

[4] Abhishek Gupta. *Cluster Schedulers,* B.Tech, 6th Semester, 04010102 Department of Computer Science and Engineering Indian Institute of Technology Guwahati; 5-9

[5] Mark Baker. *Cluster Computing Whitepaper* CoRR cs.DC/0004014 (2000)

[6] J. D. Ullman, *Polynomial complete scheduling problems*, Proceedings of the fourth ACM symposium on Operating system principles, Pages: 96 - 101, 1973.

[7] R. van Dantzig, W. Heubers, R. Boontje, I. Herschberg, D. Epema, and J. de Jongh, *A Literature was tested against the Min-min algorithm for different Study on Scheduling in Distributed Systems,* Computer System group of NIKHEF, October 1992.

[8] Howard Jay Siegel and Shoukat Ali, *Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems*, Invited Keynote Paper for the Journal of Systems Architecture Special Issue on Heterogeneous Distributed and Parallel Architectures: Hardware, Software and Design Tools ,June 1999

[9] Samih M. Mostafa, S. Z. Rida & Safwat H. Hamad. *Finding Time Quantum of Round Robin CPU Scheduling* 2010; 5(1); 1

[10] Saeed Parsa and Reza Entezari-Maleki, RASA*: A New Task Scheduling Algorithm in Grid Environment* 152-160, 2009; 3

[11][Mullen 1985] B. Mullen, *The Multics Scheduler*, http://www.multicians.org/mult-sched.html (1995)

[12] E.O. Oyetunji and A. E. Oluleye, *Performance Assessment of Some CPU Scheduling Algorithms,* Research Journal of Information Technology 1(1): 22-26, 2009;23

[13] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen and Richard F. Freund, "*Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems*," Journal of Parallel and Distributed Computing, volume 59, Pages: 107-131, 1999.