Enhancing the Performance of Feedback Scheduling

Ayan Bhunia
Student, M. Tech.
CSED, MNNIT
Allahabad- 211004 (India)

ABSTRACT

Feedback scheduling is a kind of process scheduling mechanism where process doesn’t come with any priority. According to the CPU burst needed by the process and the CPU burst remaining the processes are shifted between queues of the feedback scheduler to get completed. In multilevel feedback queue the total architecture is divided into multiple prioritised queues. In this paper, we give an approach for jobs which starve in the lower priority queue for long time to get CPU cycle. As a result response time of those starved processes decreases eight to ten percent and over all turn around time of the whole scheduling process decreases around eight to ten percents. In comparison to other types of MLFQs the performance of the proposed scheduling technique is better and practical according to the consequence.

Keywords

CPU burst, feedback analysis, starvation, response time, shifting to upper queues.

1. INTRODUCTION

In Multilevel Feedback Queue [1],[3] processes are scheduled according to their remaining CPU burst and they are shifted down from queue to queue as they have some remaining CPU burst. Every queue has unique time slice that gradually increases from upper level queue to lower level queue. So the CPU intensive jobs go down from upper queues to lower queues gradually for getting completed. Thus, lower priority queues are filled with CPU intensive jobs and as a result these processes start to starve for getting CPU attention. So then it will follow first come first serve scheduling among these jobs. It can deliver excellent overall performance similar to shortest job first or shortest time remaining come first scheduling for turnaround time, while it can also provide a responsive system for interactive jobs just like Round Robin scheduling. Here interactive job means the jobs which go for input and output operations frequently compare to the jobs which are more focused on getting CPU cycles which are considered as CPU intensive jobs. For this reason, many systems, including BSD Unix derivatives, Solaris, and Windows NT and subsequent versions use a form of MLFQ as their base scheduler [5]. In this paper the issue is resolved that the architecture mentioned in the previous paper [1] of MLFQ. The drawback is found that after processes reaches in the lowest queue, they remain in that queue until they get finished. As a result a severe slowdown in the scheduling and increase the response time and turn around time of the remaining starved processes. Here severe slowdown means that waiting time of the processes are getting increased while residing in the lowest queue. The architecture of MLFQ in this paper we propose, that dynamically reduce the response time of the processes that go down to the lowest queue and as a whole decrease the turn around time of whole scheduling.

2. RELATED WORK

In past few years different approaches are used to increase the performance of MLFQ scheduling in different ways. In paper [6], Recurrent Neural Network has been utilized to optimize the number of queues and quantum of each queue of MLFQ scheduler to decrease response time of processes and increase the performance of scheduling. Here this proposed neural network takes inputs of quantum of queues and average response time and getting the required inputs it takes the responsibility of finding relation between the specified quantum changes with average response time. It can find the quantum of a specified queue with the help of optimized quantum of lower queues. Thus, this network fixed changes and specify new quantum which overall optimize the scheduling time. In this paper [8] smoothed competitive analysis is applied to multilevel feedback algorithm. Smoothed analysis is basically mixture of average case and worst case analysis to explain the success of algorithms. This paper analyses the performance of multilevel feedback scheduling in terms of time complexity. Any performance enhancing approach can use this approach for performance analysis in terms of time complexity. In another paper [10], multilevel feedback queue scheduling algorithm is implemented in Linux 2.6 kernel and new Linux2.6 scheduler performance compared with the proposed approach. It describes two algorithms elaborately and then for different load of job, which are running in background, this scheduler is applied for calculating the average response rime. And to maintain simplicity inverse relationship is maintained between priority of processes and time slice length. This paper is a real guideline for designer of cognitive scheduling systems. Now to analyse the performance enhancement, the proposed approach is implemented and simulated in Condor [2] which provides high throughput computing environment , that handles job queue mechanism, scheduling policy, priority scheme, resource monitoring and resource management, based upon the policy fixed for execution Condor executes the job when user submits the job. For compute intensive jobs Condor is very useful. Basically, Condor pool is composed of more than one machine those are under one main machine called central manager [11]. Pool is a collection of machine and jobs. Job submitted in Condor is basically executed depending upon the Class Ad [11] of machine where it is being submitted and when Class Ad of machine matches then that job is executed in that machine. This is called matchmaking [11]. Now for simulation based approach Condor is used for simulation of the comparison based analysis of two scheduling policies which are Round Robin Opportunistic pol9icy and Multi-level Queue Opportunistic policy in the paper [7]. By effective study of this paper reveals that, for simulation based performance analysis of different type of scheduling policies, Condor is one of the effective way that provides high throughput computing environment. Condor not only executes job on standalone machine it also migrates jobs when more than one computer system is joined in the cluster and it finds idle machine to share the workload for better system throughput, basically in this paper [9] it is elaborated that in Condor when migrates job it checkpoints the current job and sent it to the idle remote machine which belong to it's cluster, for execution. Thus, for better computing Condor is one of the effective approach. In context to this, a leverage quantity is calculated which is the ratio of remote resources utilized and the sending machines utilized resources to support job migration, check pointing and supporting system calls. This leverage quantity is really a matter of issue of any high throughput computing environment for checking environment efficiency. Apart from that check pointing and job migration for data intensive application can create heavy traffic in the total grid, now Condor can handle this network traffic and manages network resources efficiently which is elaborated in the paper [8]. So, Condor not only handles jobs efficiently but also manages network for speedup networking essentials. Before going further review, computing environment of Condor is of several types. Some of useful environments supported by Condor are Vanilla, Standard, and Grid etc. [11]. For check pointing and job migration Standard or Grid universe can be used. In context to the Vanilla universe, the paper [4] described the feature of this universe through Weibull and Log-normal distribution. The paper signifies the Vanilla universe's efficiency of creating a reliable high throughput computing environment. Thus, Vanilla environment's efficiency is utilized in our approach for simulation.

3. PROPOSED APPROACH

In this approach, all the processes are inserted to the first queue and given quantum depending on which queue it is residing. Now before shifting to the next lower queue its CPU burst is modified and waiting time is calculated. Then remaining CPU bursts of each process is calculated according to the time slice of that particular queue. In this way the processes are gradually shifted from one queue to next queue depending on their remaining CPU burst. Now when remaining processes reaches the lowest queue to reduce starvation after scheduling for the first time in that queue, processes are sorted according to their remaining CPU burst. Then they are classified according to some range of values. Here range means what if scheduling function of lowest queue is observed it can be analysed how it is done. Then according to that processes are sending to that queue for total completion of those processes. The algorithm, control flow diagram and result elaborate it fully.

3.1 Proposed Architecture

In proposed architecture there are five queues .The proposed architecture is drawn as figure 1. The processes, which will be scheduled, will come to the queue 1 and it will go downwards to the lower priority queues till get finished. Input file will supply the processes information while the proposed architecture schedules the processes and the results of scheduling will be stored in the output file. Now one major common issue of this architecture is that the number queues are not constant but the time slice of each queue is increased from upper to lower. The architecture supports the scheduling policy of feedback analysis where processes do not have any fixed priority in the beginning according to the CPU burst and the time slice of each queue they are scheduled the proposed algorithm elaborates the scheduling mechanism in details.

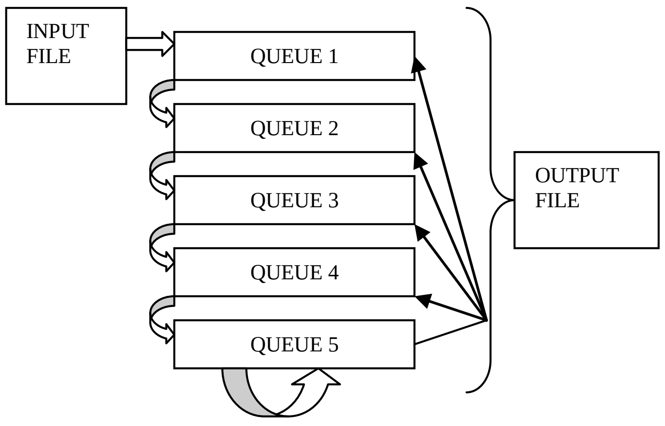


Fig 1: Proposed Architecture

From the above diagram it is clearly shown that this architecture shows that after process are traversed down to lowest queue the processes are again rescheduled to the upper queues according to the remaining time of the their CPU burst which really makes the scheduling more faster and if the processes are not eligible for sending to the upper queues they are recursively scheduled in the lowest queue and every next round some processes according to their remaining CPU burst sent to the upper queues.

3.2 PROPOSED ALGORITHM

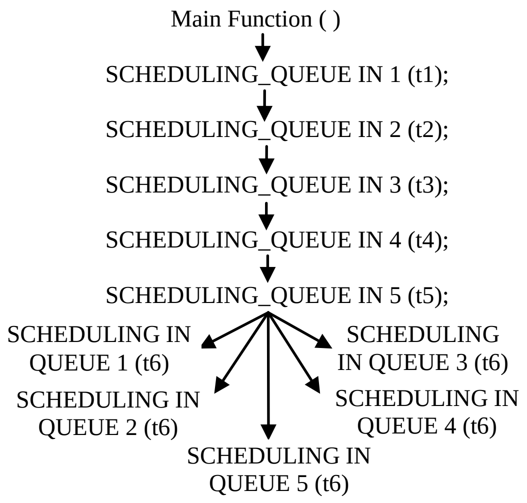
In this proposed algorithm three major functions are implemented in each layer of the multilevel feedback queue. Those functions are inserting processes one by one in each queue, feedback analysis of the processes and waiting time calculation of each process after scheduled in particular queue. The scheduling function calls are implemented recursively in lowest queue to get the scheduling faster and to scheduling from upper queue to lower queue are implemented by calling from one queue's scheduling algorithm to another queue's scheduling algorithm to make it more reliable and dynamic. The three major functions are 1) INSERT IN QUEUE N ( ); where n represents any nth queue of MLFQ.2) FEEDBACK ANALYSIS; [1] 3) WAITING TIME CALCULATION; this section describes the proposed algorithm.

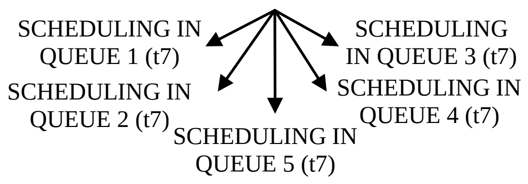
```
Main program Body
{
 Take the information of the each process from a file.
For all P_i, where 1<= i <= n
   {
 Call INSERT IN QUEUE 1 ( ) with its arrival time in q1; //this is
the function to insert the process in the First queue.
    }
Call SCHEDULING IN QUEUE 1 (n, q1); // n is the no of
process and q1 is the quantum of the first queue.
}
SCHEDULING IN QUEUE1 ( )
{
      For all P_i, where 1<=i<=n
     {
  FEEDBACK ANALYSIS ( ); // this function calculate that how
much CPU burst remaining of a process so that it can be shifted to
lower queue.
  WAITING TIME CALCULATION ( ); // this function
calculates the waiting time of a process when it get scheduled in a
particular queue of the MLFQ.
  Either it gets completed.
       Or,
  Call INSERT IN QUEUE 2 ( ) with its turn around time in q1; // this is the function to insert the process in the Second Queue.
      }
Count the number incomplete process,
Call SCHEDULING IN QUEUE 2 ( n1 , q2 , t1);// n1 is the number processes shifted to lower queue, q2 is the quantum of the second queue and t1 is the current time from where scheduling in second queue will start.
}
SCHEDULING IN QUEUE 2 ( )
{
  For all $P_i$, where 1<=p<=n1
       {
 FEEDBACK ANALYSIS ( );
WAITING TIME CLCULATION ( );
Either it gets completed.
 Or.
Call INSERT IN QUEUE 3( ) with its turn around time in q2; // this is the function to insert the process in the Third Queue.
  }
Count the number of incomplete processes.
Call SCHEDULING IN QUEUE 3( n2 , q3 , t2 ) // n2 is the number processes shifted to next lower queue, q3 is the quantum of the third queue and t2 is the current time from where scheduling in third queue will start.
}
SCHEDULING IN QUEUE 3 ( )
{
  For all $P_i$, where 1<=p<=n2
{
 FEEDBACK ANALYSIS ( );
WAITING TIME CLCULATION ( );
Either it gets completed.
 Or.
Call INSERT IN QUEUE 4 ( ) with its turn around time in q3; // this is the function to insert the process in the Fourth Queue.
  }
Count the number of incomplete processes.
Call SCHEDULING IN QUEUE 4( n3 , q4 , t3 ) // n3 is the number processes shifted to next lower queue, q4 is the quantum of the fourth queue and t3 is the current time from where scheduling in fourth queue will start.
}
SCHEDULING IN QUEUE 4 ( )
{
  For all $P_i$, where 1<=p<=n3
{
 FEEDBACK ANALYSIS ( );
WAITING TIME CALCULATION ( );
Either it gets completed.
 Or.
Call INSERT IN QUEUE 5 ( ) with its turn around time in q4; // this is the function to insert the process in the Fifth Queue.
 }
Count the number of incomplete processes.
Call SCHEDULING IN QUEUE 5( n4 , q5 , t4 ) // n4 is the number processes shifted to next lower queue, q5 is the quantum of the third queue and t4 is the current time from where scheduling in fifth queue will start.
}
SCHEDULING IN QUEUE 5 ( )
{
  For all $P_i$, where 1<=p<=n4
 {
FEEDBACK ANALYSIS ( );
WAITING TIME CALCULATION ( );
Either it gets completed.
Or
Check the remaining Burst
If (burst<=q1)
{
Call INSERT IN QUEUE 1 () with its turn around time in queue 5;
Increase the counter that will hold the number of processes that are shifted to first queue.
}
Else if (q1<burst<=q2)
{
Call INSERT IN QUEUE 2 ( ) with its turn around time in queue5;
Increase the counter that will hold the number of processes that are shifted to second queue.
}
Else if (q2<burst<=q3)
{
Call INSERT IN QUEUE 3 ( ) with its turn around time in queue5;
Increase the counter that will hold the number of processes that are shifted to third queue.
}
Else if (q3<burst<=q4)
{
Call INSERT IN QUEUE 4 ( ) with its turn around time in queue5;
Increase the counter that will hold the number of processes that are shifted to fourth queue.
}
Else
{
Call INSERT IN QUEUE 5 ( ) with its turn around time in queue5;
Increase the counter that will hold the number of processes that are shifted to fifth queue.
}
Call SCHEDULING IN QUEUE 1 (n1', q1, t); //n1' is the count of the processes send to queue 1 for completion, q1 is the quantum of the queue 1 and t is the time when first round of scheduling in queue 5 completes

Call SCHEDULING IN QUEUE 2 (n2', q2, t); n2' is the count of the processes send to queue 2 for completion, q2 is the quantum of the queue 2 and t is the time when first round of scheduling in queue 5 completes
Call SCHEDULING IN QUEUE 3 (n3', q3, t); n3' is the count of the processes send to queue 3 for completion, q3 is the quantum of the queue 3 and t is the time when first round of scheduling in queue 5 completes
Call SCHEDULING IN QUEUE 4 (n4', q4, t); n4' is the count of the processes send to queue 4 for completion, q4 is the quantum of the queue 4 and t is the time when first round of scheduling in queue 5 completes
Call SCHEDULING IN QUEUE 5 (n5', q5, t); n5' is the count of the processes send to queue 5 for completion, q5 is the quantum of the queue 5 and t is the time when first round of scheduling in queue 5 completes
}
```

```
FEEDBACK ANALYSIS ( )
{
Suppose, Time slice is d and CPU burst of the process $P_i$ (where i=0, 1, 2…., .n) is t.
if ( t<= d)
{
Allocate the time slice to the process
        {
        If process completes its execution
        Exit () ;}
Else
{
t'=t-d;
Update the process information;
Insert the process in the next lower queue;
}
}
WAITING TIME CALCULATION ( )
{
Declare a variable Varo and initially assign value one to it.
As process comes arriving from time stamp one.
Therefore, Varo =1;
For each process $P_i$ for all processes
Do {
Waiting time= Varo – process arrival time;
        If (CPU burst of the process <= time slice)
         {
         Varo = Varo + burst time of the process;
         }
        Else
        {
       Varo =Varo+ time slice of that particular queue;
        }
```

3.3 CONTROL FLOW DIAGRAM

This control flow diagram will describe the mechanism that how processes will be scheduled from first queue to the last queues according the proposed architecture and how the algorithm works when it get implemented. The following diagram really describes the proposed mechanism which is understandable after consulting the results and Gantt chart. Here SCHEDULING_QUEUE IN n is the generalized form of all scheduling function where n represents a particular queue of a MLFQ.


Main Function ( )
SCHEDULING_QUEUE IN 1 (t1);
SCHEDULING_QUEUE IN 2 (t2);
SCHEDULING_QUEUE IN 3 (t3);
SCHEDULING_QUEUE IN 4 (t4);
SCHEDULING_QUEUE IN 5 (t5);
SCHEDULING IN QUEUE 1 (t6)
SCHEDULING IN QUEUE 2 (t6)
SCHEDULING IN QUEUE 5 (t6)
SCHEDULING IN QUEUE 3 (t6)
SCHEDULING IN QUEUE 4 (t6)


SCHEDULING IN QUEUE 1 (t7)
SCHEDULING IN QUEUE 2 (t7)
SCHEDULING IN QUEUE 5 (t7)
SCHEDULING IN QUEUE 3 (t7)
SCHEDULING IN QUEUE 4 (t7)

Time line: t1<t2<t3<t4<t5<t6<t7

Fig 2: Scheduling Sequence

Repeat the steps of scheduling in the lowest queue are synchronized until all the processes get selected and completed by calling it recursively. In the earlier section of proposed algorithm one thing is clearly mentioned that in each layer basically three functions are implemented. Now the total scheduling mechanism is described by the above diagram that how the processes are traversed from upper queues to lower queues. The above diagram depicts the sequence of scheduling flow and details of control flow. In the lowest queue to reduce starvation processes are boost to upper queues according to the remaining time of their CPU burst. The time line describes that as the times goes on scheduling is get confined in the lowest queue and this scheduling sequence follow the proposed algorithm and describes how process are get managed to get finished.

3.4 GANTT CHART

Following example elaborates the performance enhancing scenario with suitable Gantt chart and discussions. Suppose, MLFQ scheduler has five queues having time slice suppose 16, 32, 64, 128 and 256. This time slice assumption is taken to show the increasing order of time slice and power MLFQ property.

Table 1: Test Case Input

Process identifier	CPU burst	Arrival time
1	621	1
2	751	4
3	499	6
4	526	9
5	546	10

Scheduling is shown by the Gantt chart to see when, where and how the process is getting executed and also to calculate the turn around time.

t1= 1

FIRST SCHEDULING IN QUEUE

P1	P2	P3	P4	P5

1 17 33 49 65 81

t2= 81

FIRST SCHEDULING IN QUEUE 2

P1	P2	P3	P4	P5

81 113 145 177 209 241

t3=241

FIRST SCHEDULING IN QUEUE 3

P1	P2	P3	P4	P5

241 305 369 433 497 561

t4=561

FIRST SCHEDULING IN QUEUE 4

P1	P2	P3	P4	P5

561 689 817 945 1073 1201

t5=1201

FIRST SCHEDULING IN QUEUE 5

P1	P2	P3	P4	P5

1201 1457 1713 1969 2225 2481

Remaining burst of P1 is 125, P2 is 255, P3 is 3, P4 is 30, P5 is 50.

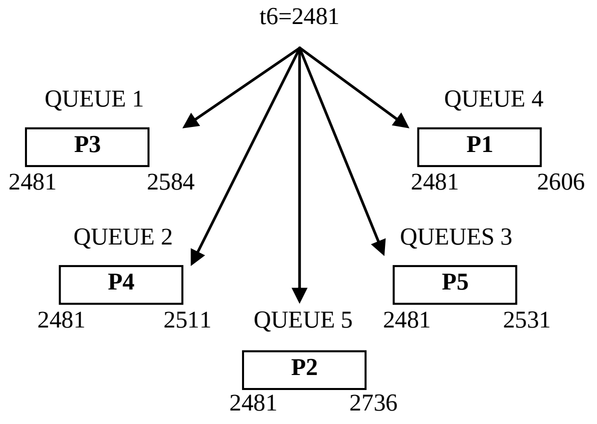


Fig 3: Sending Jobs to Required Queues

So, Turnaround time is 2736 in proposed algorithm. Now if we use the previous algorithm then the scheduling will be in the following way,

t6=2481

SECOND SCHEDULING IN QUEUE 5

P1	P2	P3	P4	P5

2481 2606 2861 2864 2894 2944

Here, turnaround time is 2944 which is greater than the proposed approach. It is shown that with the proposed approach turnaround time of scheduling is getting decreased and system is getting faster. As the total scheduling time is decreased the turnaround time of each process is also decreased.

4. SIMULTAIONS AND RESULT ANALYSIS

The simulation is done in Condor [2] which provides a high through put computing environment to execute heavy data job. Test case inputs are taken to show that for different type of CPU burst of processes how results are coming for different type of MLFQs. This result analysis really gives a comparison based idea of proposed approach and other existing type of MLFQs.

Table 2: First Test Case Input

Process identifier	CPU burst	Arrival time
1	421	1
2	551	4
3	299	6
4	326	9
5	346	10

Table 3: First Test Case Output

Algorithm	Turn around time
Power MLFQ	2535
Equal MLFQ	1935
Proposed approach	2535

Table 4: Second Test Case Input

Process identifier	CPU burst	Arrival time
1	621	1
2	751	4
3	499	6
4	526	9
5	546	10

Table 5: Second Test Case Output

Algorithm	Turn around time
Power MLFQ	2944
Equal MLFQ	3007
Proposed approach	2736

Table 6: Third Test Case Input

Process identifier	CPU burst	Arrival time
1	821	1
2	951	2
3	699	8
4	726	11
5	746	14

Table 7: Third Test Case Output

Algorithm	Turn around time
Power MLFQ	4068
Equal MLFQ	3943
PROPOSED APPROACH	3969

Combining the all three test case inputs and outputs the comparison based graph is drawn to show how this approach is efficient than existing MLFQs.

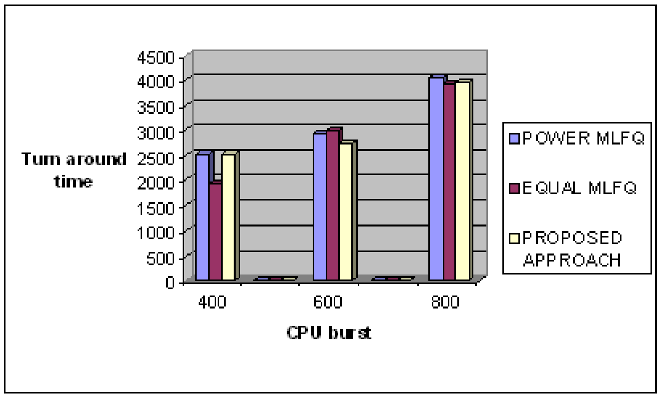


Fig 4: Comparison of Different MLFQ's Performance

The paper presents the improvement of the Multi-level Feedback Queue's CPU time allocation strategy, by adding parallelism. After, that calculating the time of completion of the scheduling in each queues. This new approach increases the system efficiency by decreasing the scheduling process and increase the response time of the processes which are in the lowest queue and starving. This parallelism can be implemented practically then it really affects the scheduler to schedule in more efficient manner. And file handling of this implantation done with the help of condor [2]. By using condor_submit [11][9] command, we submit the scheduler programme in condor computing environment. Any vanilla [11] or standard [11] computing environment is enough to run job. It needs the input file which stores the information of the processes and the output and waiting time is calculated and stored in he output file. This file handling is done by condor and Condor is very first to calculate[4][5] and here condor is needed because CPU burst of the processes are very high and we have to have calculate the times of completion of the processes of each queue and the calculating the remaining burst and waiting time all this things. So condor computing environment [7] [9] is a very helpful way to solve this computation problem.

```
Universe=vanilla
Executable= job
Input= file. In
Output=result. Out
Log= info. Log
Queue.
```

Fig 5: Submit Description File for Condor

Executable is the exe file of the job and file.in the input file that stores information of the processes and output file is result.out file which stores the result of scheduling. Queue is the word that describes that job is submitted to condor cluster for single time. There is another two files generated after completion of the job execution is info.log and info.error. First one is the log file that get generated which stores the in formations such as time of job submission, time when job get executed , address of the executing machine, date of submission, date of finish etc .

5. CONCLUSION

As starvation in the lower level queue is an important issue in multilevel feedback queue scheduling. Hence our approach gives an optimistic solution regarding reducing starvation of those remaining processes. And it can be seen from the results that overall turnaround time of the processes is getting minimized by eight to ten percents and scheduling becomes much faster.Our approach extends the performance of feedback scheduling algorithm by minimizing the response time and overall turn around time of the system by around ten percentage.

6. REFERENCES

[1] Hoganson, Kenneth (2009), "Reducing MLFQ Scheduling Starvation with Feedback and Exponential Averaging", Consortium for Computing Sciences in Colleges, Southeastern Conference, Georgia.

[2] Liu, Chang, Zhao, Zhiwen and Liu, Fang (2009), "An Insight into the Architecture of Condor –A Distributed Scheduler", IEEE, Beijing, China.

[3] Parvar, Mohammad R.E, Parvar, M. E. and Safari, Saeed (2008), "A Starvation Free IMLFQ Scheduling Algorithm Based on Neural Network", International Journal of Computational Intelligence Research ISSN 0973-1873 Vol.4, No.1 pp. 27–36

[4] Wolski, Rich, Nurmi, Daniel and Brevik, John (2007), "An Analysis of Availability Distributions in Condor", IEEE, University of California, Santa Barbara.

[5] Torrey, L.A., Coleman, J. and Miller, B.P. (2007), "A Comparison of the Interactivity in the Linux 2.6 Scheduler and an MLFQ Scheduler", Software Practice and Experience, Vol. 37, No 4, pg. 347-364, John Wiley & Sons, Ltd.

[6] Becchetti, L., Leonardi, S. and Marchetti S.A. (2006), "Average-Case and Smoothed Competitive Analysis of the Multilevel Feedback Algorithm" Mathematics of Operation Research Vol. 31, No. 1, February, pp. 85–108.

[7] Abawajy, Jemal H. (2002), "Job Scheduling Policy for High Throughput Computing Environments", Ninth IEEE International Conferences on Parallel and Distributed Systems , Ottawa, Ontorio, Canada.

[8] Basney, Jim and Livny, Miron (2000), "Managing Network Resources in Condor", 9th IEEE Proceedings of the International Symposium on High Performance Distributed Computing, Washington, DC, USA.

[9] Litzkow, Micheal J., Linvy, Miron and Mutka, Matt W. (1988), "Condor – A Hunter of Idle Workstations" IEEE, Department of Computer Sciences, University of Wisconsin, Madison.

[10] Scheduling: The Multilevel Feedback Queue. http://pages.cs.wisc.edu/~remzi/Classes/537/Fall2009/Notes/cpu-sched-mlfq.pdf

[11] Condor Team, University of Winconsin-Madision, "Condor Version 7.2.5 Manual".