Improving Efficiency of LoadLeveler

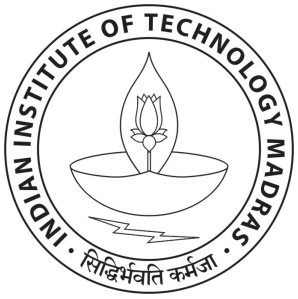
A Project Report

submitted by

D.V.REVANTH

in partial fulfilment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

JUNE 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **Improving Efficiency of Loadleveler** , submitted by **D.V.REVANTH**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Harishankar Ramachandran
Project Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 12 June 2018

ACKNOWLEDGEMENTS

This work would not have been possible without the guidance and the help of several people. I take this opportunity to extend my sincere gratitude to all those who made this thesis possible. First, I would like to thank all my teachers who bestowed me with good academic knowledge. I am indebted to my advisor Prof. Harishankar Ramchadran whose expertise, generous guidance and support made it possible for me to work on this topic. I would like to thank my family for giving support and guidance all through my life. I would also like to thank all my friends and well-wishers for helping me in difficult times and being a good source of support and guidance.

ABSTRACT

KEYWORDS: IBM LoadLeveler, Architecture, Backfilling scheduler, Virgo.

LoadLeveler is a job management system that allows users to run more jobs in less time by matching the jobs' processing needs with the available resources. LoadLeveler schedules jobs, and provides functions for building, submitting, and processing jobs quickly and efficiently in a dynamic environment.This report talks about the basics of LoadLeveler. Backfilling scheduling and it's variants Easy Backfilling and Conservative Backfilling are explained. Working of Virgo cluster is explained. Data is taken from the Virgo cluster and is run in a simulation to find out about efficiency of the cluster.

TABLE OF CONTENTS

CHAPTER 1

LOADLEVELER

1.1 Introduction

LoadLeveler is a job management system that allows users to run more jobs in less time by matching the jobs' processing needs with the available resources. LoadLeveler schedules jobs, and provides functions for building, submitting, and processing jobs quickly and efficiently in a dynamic environment.

LoadLeveler has different types of interface for user submitting jobs and administrators to configure system and control the jobs. Control files define the elements, characteristics, and policies of LoadLeveler and the jobs it manages. The command line interface, which gives you access to basic job and administrative functions. An application programming interface (API), which allows application programs written by users and administrators to interact with the LoadLeveler environment. Users are typically restricted to submit jobs and check their status. Administrator is responsible for configuration , job scheduling and accounting. Administrator has control over the jobs by setting job classes that define how and when a job will run and selecting the specific scheduler

1.2 Architecture

Machine in Loadleveler perform following roles

Job manager:When a job is submitted it is placed in a queue managed by jobmanger.Then job manager contacts central manager to find a machine to run the job .It starts the execution of job on the nodes and the keeps the information about job life cycle.It schedules jobs from the submit only machines.

Executing machine:It is a machine that runs the job.

Submitting machine:It is used to submit jobs to the cluster from the machines that are out of the cluster.It allows the user to query and cancel the job they submitted.

Central manager:It runs on a single machine and it has Resource manager and job scheduler.It finds the machines that are available to run the job.It then notifies the scheduler.

Resource manager: Resource manager consists of the information about the cluster. A daemon on node each pushes the information to resource manager.Resource manager has the status of every nodes including dynamic and static attributes.Static attributes includes memory,hardware attributes etc.Dynamic attributes include job state on the node and CPU utilization. Scheduler allocates the jobs by matching the requirements of the jobs and the machines available using the data from resource manager.

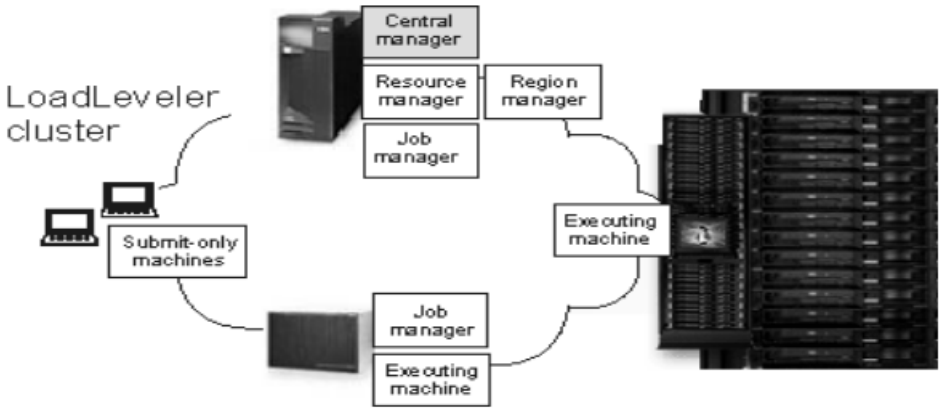


Figure 1.1: LoadLeveler cluster

1.3 Loadleveler Daemons

Daemons that process the jobs LoadL_master : Master daemon runs on all machines in cluster and manages other daemons.

LoadL_schedd:Schedd daemon receives jobs and manages them on machines selected by the negotiator daemon

LoadL_startd:Startd daemon Monitors job and machine resources on local machines, Generates local adapter configuration,Forwards information to the negotiator, resource manager, and region manager daemons,spawns the starter process.

LoadL_region_mgr:The region manager daemon detects and monitors node and adapter status from all of the startd machines it manages and sends the status information to the central and resource manager daemons.

LoadL_resource_mgr: Collects all machine updates from the executing machine. Generates events to the scheduler for changes in the machine status. Maintains a list of all jobs managed by the resource manager. Responds to query requests for machine, job, and cluster information.

LoadL_negotiator:Negotiator daemon Monitors the status of each job and machine in the cluster,responds to llstatus and llqcommands.Runs on central manager.

1.4 Job Cycle

When a job is submitted to loadleveler.Schedd daemon on scheduler stores the information about the job on local disk and sends job information to negotiator daemon in central manager.Negotiator daemon authorizes schedd daemon to start and the job.At this point job is in pending or starting state. Schedd daemon contacts the startd daemon on executing machine to start the job.Startd daemon starts a starter process and schedd daemon send the information about the job to it.Schedd daemon notifies the negotiator that the job has started and negotiator marks the job as started. After job is completed starter process notifies the startd daemon and startd notifies schedd which in turn and negotiator daemon and negotiator marks it as completed.

1.5 Work Load Manager

WLM monitors system resources and regulates their allocation to processes. These actions prevent jobs from interfering with each other when they have conflicting resource requirements. A single WLM class is created for each job step and the process id of that job step is assigned to that class. This is done for each node that a job step is assigned to run on. LoadLeveler then defines resource shares or limits for that class depending on the LoadLeveler enforcement policy defined. These resource shares or limits represent the job's requested resource usage in relation to the amount of resources available on

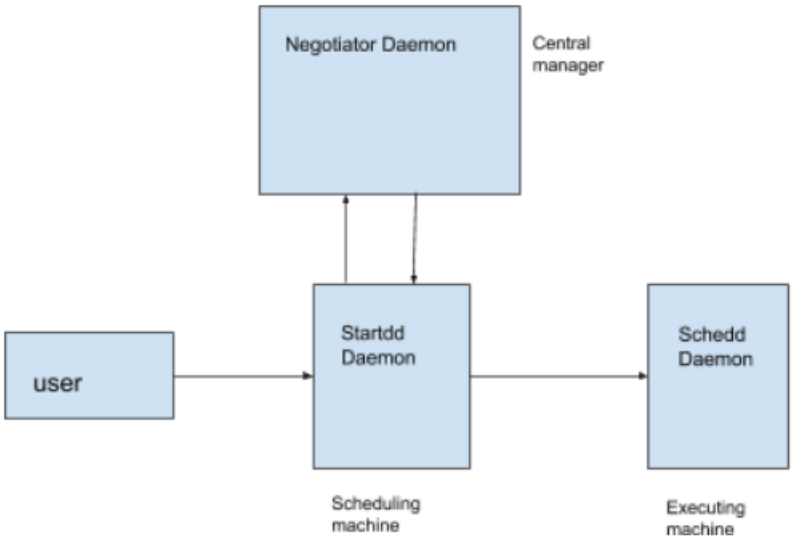


Figure 1.2: Jobcycle of Loadleveler

the machine.When the enforcement policy is shares, LoadLeveler assigns a share value to the class based on the resources requested for the job step (one unit of resource equals one share).

CHAPTER 2

Backfilling

2.1 Introduction

Backfill is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order. When Loadleveler schedules, it prioritizes the jobs in the queue according to a number of factors and then orders the jobs into a 'highest priority first' sorted list. It starts the jobs one by one stepping through the priority list until it reaches a job which it cannot start. Because all jobs and reservations possess a start time and a wallclock limit, Loadleveler can determine the completion time of all jobs in the queue. Consequently, Loadleveler can also determine the earliest the needed resources will become available for the highest priority job to start.

Backfill operates based on this 'earliest job start' information.Because Loadleveler knows the earliest the highest priority job can start, and which resources it will need at that time, it can also determine which jobs can be started without delaying this job. Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job. If Backfill is enabled, Loadlveler, 'protects' the highest priority job's start time by creating a job reservation to reserve the needed resources at the appropriate time. Then any job which do not interfere with this reservation can be started

2.2 Backfilling in LoadLeveler

Backfill is applied for the job queue after jobs are sorted according to priority in Loadleveler. This priority is based on SYSPRIO values.SYSPRIO used in our system is

$$SYSPRIO = 10000000 * \$(UserTotalShares - UserUsedShares) - QDate. \quad (2.1)$$

Backfill tries to allocate resources to each job sequentially in the queue.When it encounters a job which it cannot allocate resource immediately the job becomes TOP_DOG.In loadleveler backfill classified jobs into 3 types: REGULAR,TOP_DOG and BACKFILL.

REGULAR:When there are enough resources for a job to run and no TOP_DOGS are allotted then that job is called a regular job.

TOP_DOG:When a job doesn't have enough resources currently but will be available in a future time and it is not expected to block another top_dog.

BACKFILL:When a job have enough resources available and and it can be using the resources reserved by any top_dog only if it is able to complete its job before start time of top_dog.

To use backfill scheduling we must need an estimate of job run time(Wall clock limit) which can be provided by user or predefined for a job class by admin. If run time is more accurately predicted we can improve the efficiency of backfill scheduler. As a job wouldn't run for the entire duration of wall clock limit and top dog can be scheduled before the wall clock time or some other jobs can be included before the top dog starts. Backfilling improves the efficiency of large system by 20% and it greatly improves the quality of service for short jobs and less for large jobs.

2.3 Types of Backfilling

There are 2 main variations in Backfill algorithm EASY aggressive backfill and Conservative backfill. Conservative Backfilling: In conservative backfilling every job that enters the system has a reservation.Jobs can move forward in the queue as long as they don't delay any previous job in the queue. EASY or Aggressive backfilling: In EASY backfilling jobs can move forward as long as they don't delay the first job that has reservation. In conservative backfilling ,new job is given a reservation that it does not delay previous jobs.Therefore existing jobs have fixed start time and it is difficult for jobs coming later to backfill easily.So when a long job comes mostly it wont get backfilled. In Easy backfilling reservation is made only for 1 job at any point of time.So many jobs can come and backfill.But it would cause some delays for the jobs that are already

present in the queue.So jobs which require large nodes will get delayed by a significant amount. Conservative backfill allows such jobs to have a start time fixed whereas in Easy they are mostly not given reservation and they have to wait until they are at the front of the queue. So jobs which require more time but less processors find it easy for them in Easy backfilling.But coming to job which require more processors less time find it beneficial in conservative backfilling.Coming to Jobs which requires less processors and less time mostly tends backfill in both variants.Jobs which require more processors and more time won't get advantage from both strategies. ‘

2.4 Example

Let us consider Jobs J1,J2,J3,J4,J5,J6,J7 are jobs in queue with priority in same order.Maximum nodes in cluster are 24.

Table 2.1: Example dataset of jobs

$Jobid$	$NoofCPU$	$time$
J1	3	2
J2	4	2
J3	18	24
J4	8	2
J5	20	24
J6	6	14
J7	6	14

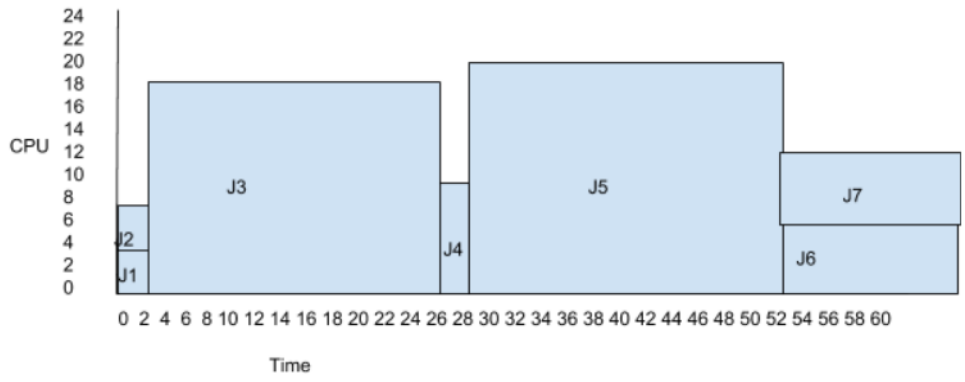


Figure 2.1: First Come First Serve

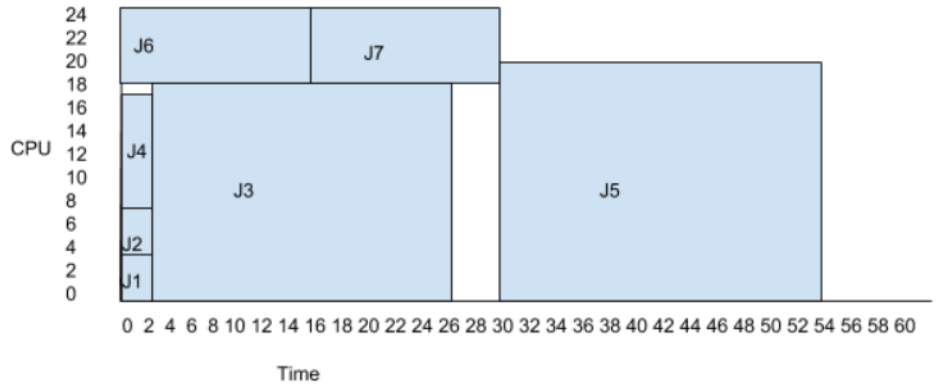


Figure 2.2: Easy Backfilling

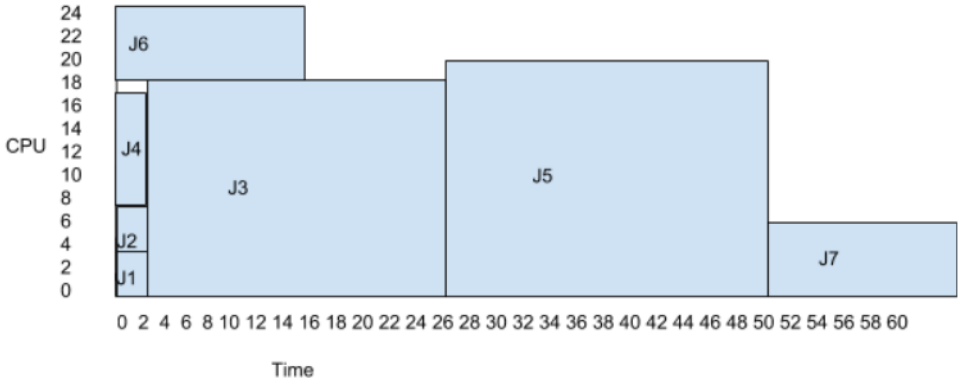


Figure 2.3: Conservative Backfilling

CHAPTER 3

Working on our sysem

3.1 Virgo

Virgo cluster consists of 298 nodes.Out of which 2 are master nodes and 4 are storage nodes.Each node had 16 cores.These nodes are divided into groups accordingly to Loadleveler admin file. They are parallel,LParallel(Large parallel),Serial, SNCCRD, PNCCRD, CSE, QMANAGER, IBM.Each group has been allocated some nodes of the cluster. There are several classes into which each job can be classified.Each class

Table 3.1: No of nodes allocated to each group

$Group$	$Noofnodes$
Parallel	127
LParallel	60
Serial	63
SNCCRD	16
PNCCRD	16
CSE	4
QMANAGER	3
IBM	4

has certain wall cock limit, Job cpu limit and cpu limit.A job is classified into the classes based on no of processors it requires and software it requires.The classes defined on our system are Small, Medium, Long, Verylong,Small128 ,Long128 ,Verylong128, Small256, Medium256, Long256, Verylong256, Small512, Medium512, Long512, Verylong512, Fluent, Fluent16, Abaqus, Ansys, Mathmat, Matlab, Star, Star32, Gaussian, Lgaussian, Linda32, Aesmall, Aemedium, Aelong, Aeverylong, Aeverylong128, Cssmall, Csmedium, Cslong, Csverylong, AE,IBMQueue.These jobs are scheduled using backfill algorithm.The job queue is determined by the fairshare value and order of arrival of jobs.

3.2 Simulator for backfilling

I wrote an algorithm for backfilling which takes no of jobs , no of nodes and maximum time for each job

```
#include <stdio.h>
int processor=292;
int runtime=15000;
int matrix[292][15000];
int limit=292;
```

This contains the variables to run our code.It has number of processors available,run time of the simulation and number of processors our system can use out of all the processors.

```
int main()
{
    int i=0;int j=0;int k=0;int njobs=0;
    scanf("%d",&njobs);
    for(i=0;i<processor;i++)
    {
        for(j=0;j<runtime;j++)
        {
            matrix[i][j]=0;
        }
    }
    int input[njobs][2];
    for(i=0;i<njobs;i++)
    {
        scanf("%d",&input[i][0]);
        scanf("%d",&input[i][1]);
    }
    for(i=0;i<njobs;i++)
    {
        int g=i+1;
```

```
        check(input[i][0],input[i][1],g);
    }

    int endtime=0;
    endtime=etime();
    float count=0;
    printf("%d\n",endtime);
    for(i=0;i<processor;i++)
    {
        for(j=0;j<(endtime);j++)
        {
            if(matrix[i][j]!=0)
            {
                count=count+1;
            }
           // printf("%d\t",matrix[i][j]);
        }
        //printf("\n");
    }
    float efficiency=0;
    float total=processor*endtime;
    efficiency=100*count/total;
    printf("%f %f %f",count,efficiency,total);
    return 0;
}
```

Data is read and stored in input matrix and it is given to the check function to check if the job can be backfilled.

```
int fill(int a,int b,int start,int tstart,int jobid)
{
    int l=0;int m=0;
    for(l=0;l<a;l++)
    {
```

```
            for(m=0;m<b;m++)
            {
                matrix[start+l][tstart+m]=jobid;
            }
        }
        return 0;
}
int checkblock(int c,int d,int p,int q)
{
        if((q+c)>limit)
        {
            return 1;
        }
        int i=0;int j=0;int f=0;
        for(i=q;i<(q+c);i++)
        {
            for(j=p;j<(p+d);j++)
            {
                if(matrix[i][j]!=0)
                {
                    f=1;
                    return f;
                }
            }
        }
        return f;
}
int check(int c,int d,int jobid)
{
        int x=0;int y=0;int flag=0;
        for(x=0;x<runtime;x++)
        {
            for(y=0;y<processor;y++)
```

```
            {
                flag=checkblock(c,d,x,y);
                if(flag==0)
                {
                    fill(c,d,y,x,jobid);
                    return 0;
                }
            }
        }
        return 0;
}
```

Check is used to find where a job starts and checkblock finds if it is possible to back fill.Fill functions stores the data in the output matrix.

```
int etime()
{
    int i=0;int j=0;int coloumn=0;
    for(i=0;i<runtime;i++)
    {
        coloumn=0;
        for(j=0;j<processor;j++)
        {
            if(matrix[j][i]!=0)
            {
                coloumn=1;
            }
        }
        if(coloumn==0)
        {
            return i;
        }

    }
```

}

This function is used to find the end time of our simulation.

3.3 Input Data

Input data is taken from llq of virgo cluster for 24 hours.It consists of JobId, User, Date and time of submission,Class, Status of job, Running node.Another file is given by llsummary which includes job id and start time of job on the machine.By comparing the jobid I obtained the job starting time of each running job.Then based on class of each job I took the maximum running time of job and found the ending time of job.At the start(t=0) of simulation found when running jobs will end. Each node will have more than 1 job running at a time. So I calculated the job which will end at the last on each node and made the entire node run for that duration.Since I could not know what resource each job in waiting queue required I took the maximum resource the job can take based on the class it came into queue.The resource and waiting time for each class is taken from Load admin file and hpce data.The were considered as consuming whole nodes at the maximum limit of class so for each job I considered they are consuming all the processors in the nodes.There were fair share values to consider but I could not find the fair share values of users queued.So I took their order of arrival as priority list.Some of the jobs were running beyond their cpu limit time as they may have been not using the cpu time but using IO time.So I have considered Such jobs will run until just the start of our simulation.

3.4 Results

While checking the data I observed that there are around 44 nodes which were not being used .Jobs were available to run but the nodes were not being used.While observing the jobs which were running on the nodes some job classes which are not in the groups class list are also running on those nodes.So the algorithm may consider nodes which are not running while scheduling. I have run the input data on the backfill simulation code by considering that 445 jobs are to be scheduled on only 248 nodes are working and i

CPUs	CPU Time	classname
>=1 <16	2 Hrs	Small
>=1 <16	24 Hrs	Medium
>=1 <16	240 Hrs	Long
>=1 <16	900 Hrs	Verylong
>=16 <=128	2 Hrs	Small128
>=16 <=128	240 Hrs	Medium128
>=16 <=128	480 Hrs	Long128
>=16 <=128	1200 Hrs	Verylong128
>128 <=256	480 Hrs	Small256
>128 <=256	900 Hrs	Medium256
>128 <=256	1200 Hrs	Long256
>128 <=256	1800 Hrs	Verylong256
>=1 <=16	120 Hrs	Fluent
>16 <=32	240 Hrs	Fluent32
>=1 <=16	1 to 240 Hrs	Abaqus
>=1 <=16	1 to 240 Hrs	Ansys
>=1 <=16	1 to 240 Hrs	Mathmat
>=1 <=16	1 to 240 Hrs	Matlab
>=1 <=16	120 Hrs	Star
>16 <=32	240 Hrs	Star32

Figure 3.1: HPCE Data

got the following results.3877 is the time at which every given job gets completed.The table gives the efficiency of the machine from t=0 to t= given time. Now considering

Table 3.2: Efficiency over time for 248 nodes

$time$	1	24	250	500	1000	2000	3000	3877
efficiency	100	99.915	98.8	98.6	98.7	98.3	72.12	60.8

the same for 292 Nodes I got the following results. If more nodes are being used we

Table 3.3: Efficiency over time for 292 nodes

$time$	1	24	250	500	1000	2000	3000	3877
efficiency	100	99.92	99	98.8	99	93.3	68.5	51.69

can get more computational power in less time. The original efficiency of the system was around 80%.

3.5 conclusion

There are a total of 292 compute nodes in Virgo.But out of those 292 some of the nodes were not working.Even if jobs are available they are not starting. In scheduing algorithms Loadleveler has 2 algorithms Fisrt come first serve and Backfilling.Slurm also uses Backfiling scheduling and FCFS. Out of the present available scheduling algorithms available Backfilling is one of the most used and most efficient algorithm compared to other space sharing algorithms.Slurm has a time sharing algorithm called Gang scheduling and it was not found on Loadeveler. Slurm is also having the same scheduling algorithm as loadleveler. The actual scheduling is around 80% it is less than our results because we are considering our jobs use complete nodes while doing the jobs and they will run for entire job duration.Some jobs do not require whole nodes so some of the processors on the node are not running and jobs which require whole node can't run on them and every job do not run for entire job time defined in class. If a certain job is not found to fill that vacant part the processors will remain idle.Due to these factors the original efficiency would be less than what we have got in result. Efficiency is calculated from start of the simulation to the time at which we are finding.. The efficiency of our system is very less at the last because as we are not giving the jobs continuously at the last only few jobs will be running and remaining nodes would be idle by completing the jobs which were assigned to them.So the efficiency drops at end of the running time.

REFERENCES

[1] Loadleveler -*IBM Loadleveler knowledge center. Retrieved Jun 17, 2018*

[2] Backfilling -*Backfill Overview. Retrieved Jun 17, 2018*

[3] Types of scheduling algorithms -*Workshop on Job scheduling Strategies. Retrieved Jun 17, 2018*

[4] Slurm -*About Slurm and schedulers available. Retrieved Jun 17, 2018*

[5] Virgo details -*Details of Virgo cluster. Retreved Jun 17,2018*