MEASURING THE QUALITY OF SERVICE ON NODES IN A CLUSTER

Rohitashva Sharma*, P. S. Dhekne*, R. S. Mundada*, Sonika Sachdeva*,
Computer Division, Bhabha Atomic Research Centre, Mumbai, India

Helge Mainhard#, Olof Barring#, Tony Cass#, CERN, Geneva, Switzerland

Abstract

It is important to know the Quality of Service offered by nodes in a cluster both for users as well as local resource management software like LSF, PBS and CONDOR for submitting a job on to a given node. This will help in achieving optimal utilization of nodes in a cluster. Simple metrics like load average, memory utilization etc do not adequately describe load on the nodes or Quality of Service (QoS) experienced by user jobs on the nodes. We had undertaken a project to predict Quality of Service seen by user job on a cluster node by correlating simple metrics like Load Average, CPU Utilization and IO on the node. This paper presents our efforts and methodology we have followed for predicting QoS of nodes in a cluster.

We have derived QoS metric in 3 different ways. I) by using Unix Load average metric, II) by using VmStatR metric III) By using CPU utilization and load on the node. We will discuss variations between measured execution time for sample probe programs and execution time predicted by QoS metric derived in above-mentioned manner. We have also studied behavior of CMSIM (simulation) and ORCA (reconstruction) programs under various load conditions and tried to find correlation metric to predict QoS for these jobs. Finally, we will present difficulties experienced in predicting Quality of Service on nodes in a cluster

INTRODUCTION

Computer farms or clusters have become common in any organization to fulfill the computing needs. A system administrator should have to manage cluster in such a way that the resources should be used optimally to get maximum throughput. To achieve this one must know the capability or resource provided by each node in the cluster. Linux system parameters like load average do not adequately describe the load on a given node, which intern results in less optimal utilization of resource.

We Computer Division, BARC in collaboration with IT Division, CERN took up a project to predict Quality of Service (QoS) offered by the node for a given task. The value of QoS varies between 0 and 1. This metric defines the goodness of the node for further tasks. If we know the execution time of task in no load condition on a given node we can predict the execution time in given conditions using QoS metric. The relation is defined as

$$T_{execution} = \frac{T_{noload}}{QoS}$$

$T_{execution}$ = Wall clock execution time for any task
T_{noload} = Wall clock execution time of the task on a given node without load
QoS = Quality of Service

In further sections we will discuss the methodology used and observations in detail.

APPROACH / METHODOLOGY

Depending on the type of resource used by a task, we classified the tasks in three categories- CPU bound, IO bound and network bound tasks. CPU bound task mainly perform computations during the course of its execution and an IO bound task spends most of its execution time in doing IO operations on disk. When a task is submitted its resource requirement depends on its task category so QoS will be different for different type of tasks submitted to the node

We selected the probe applications, which are representative of each task category. The execution time of probe applications in no-load condition and in different load conditions will be used to predict the Quality of Service of a given node. We also developed the programs to generate load in each task category on the node. The idea is to generate different kind of load combinations using combinations of loading programs and measure the execution time of representative probe.

System metrics are monitored to correlate the load on the node with the execution time of probe. We use EDG-Fabric Monitoring System to monitor the system metrics. Metrics are measured at the start of the probe as well as during the execution of the probe. Then the wall clock execution time of the probe is correlated with metrics monitored before submission of probe. Datasets are generated for all the probes in different load conditions.

We selected LINPACK CPU benchmark program as CPU bound probe as it represents scientific programs and performs mostly CPU bound operations. We developed a program for generating CPU bound load on the node. We could run multiple instances of loading program to vary CPU load. Disk io can be done differently on a given

*{rsharma, dhekne, rsm, sonikam}@barc.ernet.in
#{Helge.Mainhard, Olof.Barring, Tony.Cass}@cern.ch

node and system behaves differently in different cases. We considered two kind of disk IOs one where disk io is done in big blocks of data (referred as block IO in rest of the paper) and the other where IO is done in small chunks (referred as character IO in rest of the paper). For disk IO bound probe we used a simple disk io benchmark program called Bonnie. The code of bonnie was modified to make it run as per our needs. We also developed programs to generate both kinds of Io loads. We left the network bound exercise as the test was being done for all sequential programs and there is very less network communication involved. This was also to keep things simple as execution time of network bound task depends on the load on communicating partner as well.

We developed scripts to run probes in different load conditions. The scripts are responsible for generating a given load condition, monitor system metrics and launching of probe. The output of the script is a formatted text file containing metric values and probe execution time. This file can be imported to Microsoft Excel for further analysis.

SETUP

To carry out these tests we setup a Linux cluster with 32 nodes where each node comprised of Pentium 4 processor @1.6 GHz with 640 MB memory and 40GB hard disk. Each node runs a copy of RedHat Linux 7.3 operating system. EDG Fabric Monitoring System is also installed on the cluster to monitor system parameters.

OBSERVATIONS

We generated several datasets of probe execution time and system metrics. The monitored system metrics are load average, memory utilization, CPU utilization, swap space utilization etc. We correlated the metric values with the execution time of the probe. We had to eliminate a few metrics from correlation, as their effect on execution time of probe could not be justified. In this section we will discuss about the variation of execution time with relevant metrics and need of other metrics to correlate.

CPU Probe

We correlated the execution time of CPU probe in different load conditions with load average metric. We have used 1 minute load average because it is the one that better represents dynamic load changes. Using load average QoS is defined as

$$QoS = \frac{1}{1 + LoadAverage} \qquad \text{(Equation 1)}$$

Figure 1 shows the variation of wall clock execution time of the CPU bound probe with respect to Unix Load Average in CPU, CPU + character IO and CPU + block IO load. You can see that in CPU and CPU + character IO load execution time increases with increase in load average but in case of CPU + block IO load execution time remains approximately constant with increasing load average.

Figure 1: Variation of Execution Time of CPU probe with Load Average

Unix load average includes the average number of processes ready to run and average number of processes waiting for IO. This makes load average unfit for predicting QoS for CPU bound tasks.

This forces us to have other means to predict QoS for CPU bound tasks. We propose two methods to predict the QoS. One is to measure a metric (later called VmStatR) that represents running average of number of processes in run queue. The QoS can be defined as

$$QoS = \frac{1}{1 + VmStatR} \qquad \text{(Equation 2)}$$

We added a sensor in monitoring system to measure VmStatR metric and executed the CPU probes in different load conditions and observed the variation of execution time of probe with VmStatR metric. These variations are shown in figure 2. It is evident from figure 2 that VmStatR represents true load on CPU and can indicate about the CPU available to newly submitted task. The execution time of probe varies linearly with VmStatR in all three load conditions.

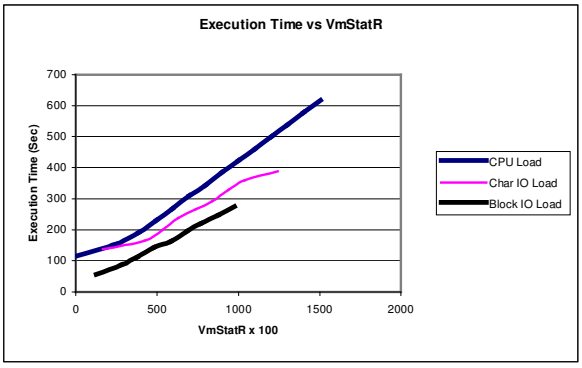

Figure 2: Variation of Execution Time of CPU probe with VmStatR

Available CPU to a new process can be calculated using CPU utilization metric. CPU utilization metric includes CPU time spent in user mode, system mode and the time it was idle. These times are represented in percentage. When a new process is submitted, it will be entitled for available idle time and a share of user and system time. The QoS can be defined as

$$QoS = \frac{\left(IdleTime + \frac{UserTime}{1+VmStatR} + \frac{SystemTime}{1+VmStatR} \right)}{100}$$

(Equation 3)

We calculated the QoS for CPU probe in block IO load condition in which we were not getting good results using load average. The table below shows the observations for this case. We can see here that the predicted QoS using equation 1 (load average) reduces with increasing block IO load but the predicted QoS using equation 3 is not varying hence the execution time of probe.

Table 1: Variation of predicted QoS using equation 1 and 3 in block IO load condition

QoS using eq. 1	QoS using eq. 3	Execution Time (Sec)
0.2433	0.4300487	32
0.1605	0.4375441	31
0.1329	0.4624468	32
0.1136	0.415	30
0.1042	0.4536079	31
0.0952	0.4290476	30
0.0869	0.4430435	31

For the final comparison we executed CPU probe under different load combinations and measured the execution time. The loading combinations are

CPU Load (LC)
CPU Load + Block IO Load (LC + LB)
CPU Load + Character IO Load (LC + LCh)
Block IO Load + Character IO Load (LB + LCh)

Figure 3 shows the comparison of measured execution time with the execution time calculated using QoS obtained from all three equations.

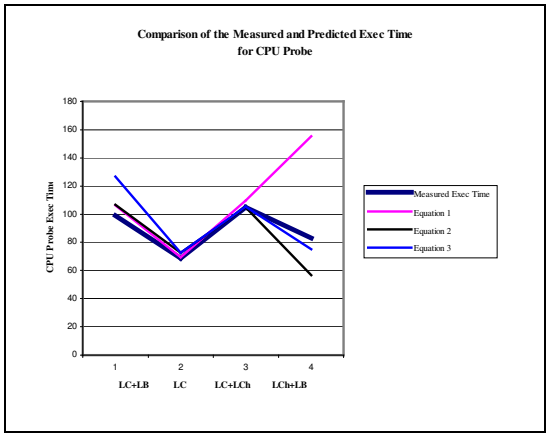

Figure 3: Comparison of measured execution time of CPU probe with predicted execution time

IO Probe

For IO probe we concentrated mostly on block IO operations. We executed IO probe in different loading conditions as was done for CPU probe and calculated the QoS using the three equations. Figure 4 shows the comparison of measured execution time with the predicted execution time.

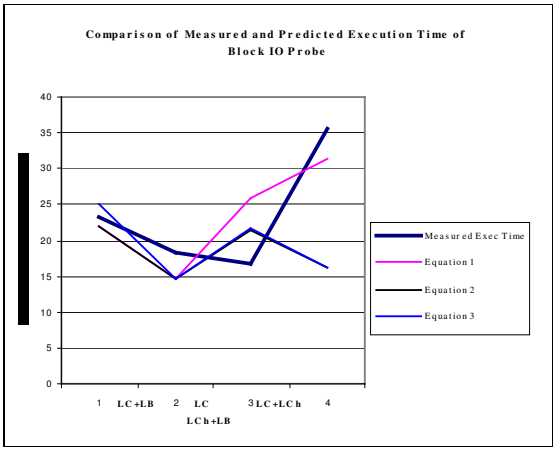

Figure 4: Comparison of measured execution time of IO probe with predicted execution time

CMSIM

We predicted the execution time of CMSIM job using the QoS metric and table 2 shows the measured execution time, predicted execution time using equation 2 and percentage error for CMSIM task in CPU + IO load condition.

Table 2: Comparison of measured and predicted execution time for CMSIM task

Measured Execution Time (Sec)	Predicted Execution Time (Sec)	% Error
585	610.8687	4.422
739	744.3209	0.720017
929	934.466	0.588377
1082	1080.702	-0.11999
1230	1216.43	-1.10328
1413	1381.166	-2.25294
1687	1707.317	1.204332

PROBLEM AREAS

In this section we will discuss the problems we faced and possible causes of increase in error in prediction. When the size of task is more than available memory, swapping starts is the system. At this time Linux kernel dynamically modifies the priority of the tasks and allocate resources. Figure 5 below shows the variation of execution time probe, % memory used and % swap space used. Here we use a memory loading program, which occupies the given amount of memory in the machine. After every data sample we increase the memory occupied by the loading program. As we can observe in figure 5 that when there is enough memory to accommodate probe there is not much variation in the execution time of the probe but as the used memory approaches 100% swapping starts and it becomes difficult to explain the variation of execution time.

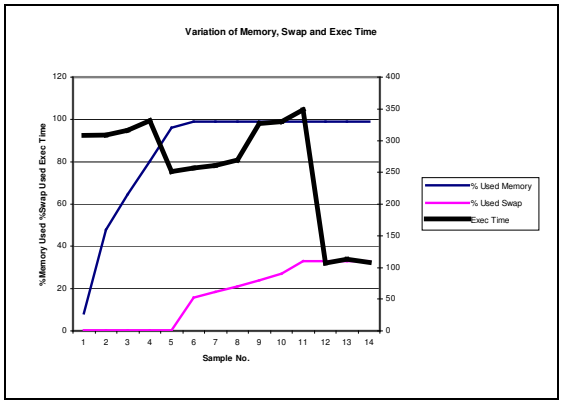

Figure 5: Variation of Execution time when swapping is there in node

The other limitation comes because of the monitoring system. Latest value of a parameter is important to predict better quality of service but it is very difficult to monitor metrics at very high frequency, as at higher sampling frequency monitoring will itself load the node.

In our calculations and observations we only consider the metrics before the submission of task. It may so happen that some task(s) exit during the course of execution of our task and increase the QoS for the rest of execution or some task(s) are submitted after the submission of our task and deteriorate the QoS predicted for our task.

CONCLUSION

After analyzing different datasets we have found that the execution time of probe depends on no load execution time of the probe and the availability of resources to it. Equation 2 and 3 can be used to predict QoS for CPU bound tasks. We successfully predicted QoS for CMSIM jobs. Equation 1 gives the nearest prediction for IO bound tasks. When there is swapping in the system it is difficult to predict QoS and chances of error in prediction increases with the time interval between metric samples.

REFERENCES

[1] Understanding the Linux Kernel, Second Edition by Daniel P. Bovet, Marco Cesati

[2] Linux Kernel Internals (2nd Edition) by Michael Beck, Harald Bohme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus, Dirk Verworner

[3] "Predicting the CPU availability of Time-shared Unix Systems" UCSD Technical Report, Rich Wolski, Neil Spring, Jim Hayes