Load Distribution on a Linux Cluster using Load Balancing

Aravind Elango *M. Mohammed Safiq*

Undergraduate Students of Engg.
Dept. of Computer Science and Engg.
PSG College of Technology
India

Abstract:

In a cluster of computers, the system load that exists in each of the computers if nearly equal is an indicator of good resource utilization. Instead of balancing the load in the cluster by process migration, or by moving an entire process to a less loaded computer, we made an attempt to balance load by splitting up large incoming problems by our novel method and then distributing them to computers in the cluster. The results on implementation of our ideas, algorithms and test values have also been put forth. In this paper we present our Load Balancing technique along with experimental results as a proof of technique.

Keywords:

Cluster computing, Linux clusters, Load Balancing.

Problem Definition:

The distribution algorithms that we propose is for problems that are inherently parallel or problems that can be worked with reasonable amount of synchronization. The problem should be divisible into a number of smaller, similar tasks. Examples of such problems include mathematically intensive operations like all matrix operations, Fast Fourier Transformation and prime numbers computation.

Proposed Distribution Algorithm:

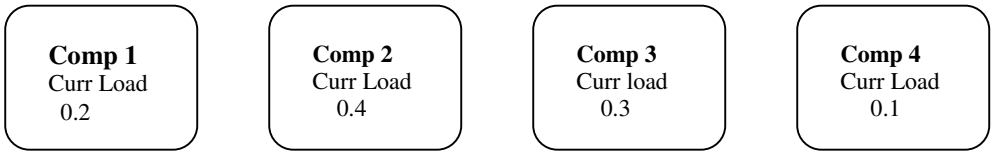
Our distribution algorithm considers the number of nodes present, their current load and the amount of computation required and calculates the load distribution among the nodes. The inputs to the algorithm are number of nodes online, the expected load

increase each task is to cause, the number of tasks and load in each node. The tasks are to be distributed in such a way that each node gets a proportionate number of tasks to process and at the same time all the nodes get equally loaded. Finding the total increase in load for all nodes taken together, and then dividing it by the number of systems can achieve this. This provides the expected load per node. Now this becomes a theoretical upper limit for the load on each node. Hence we subtract the current load from the expected load to get the increase in load for each node. If the increase is zero or negative, it suggests that the load in that particular node is higher or equal to the theoretical upper limit and hence it would not merit any consideration for the allocation of tasks. Next the percentage of tasks each node is to calculate is found out by dividing the increase in load for each machine by the total increase in load (considering only the nodes which are going to do processing). This gives a measure of how to divide the tasks amongst the nodes. Multiplying this percentage by the total number of tasks, the numbers of tasks per node for all the nodes, which are going to take part in the processing, are found.

Finding The Load Per Task (lpt):

Before the distribution algorithm is executed, the procedure should be scanned once to estimate the number of arithmetic operators, relational operators and system calls that each task requires and the load per task is calculated.

Let us suppose that there are four computers in the cluster with loads as shown below:



The process to be executed:

```
cr = cc = 1

er = ec = c1 = c2 = r1 = 400

lseek(fr , cr*c1*4 ,SEEK_SET);
lseek(fc , cc*c1*4 ,SEEK_SET);
for(;cr<r1;cr++)
{
      for(;cc<c2;cc++)
      {
            if(cr==er&&cc==ec)
                  exit(0);
```

```
                el3=0;
                for(i=0;i<c1;i++)
                {
                        read(fr,&el1,4);
                        read(fc,&el2,4);
                        el3+=el1*el2;
                }
                send(new_fd,&el3,4,0);
                lseek(fr,-c1*4,SEEK_CUR);
        }
        cc = 0;
        lseek(fc,0,SEEK_SET);
        lseek(fr,c1*4,SEEK_CUR);
}
```

If the above code were to be executed, the load per task is to be calculated as follows:

Analyzing the code we can find

No. of Arithmetic and: 400*400*400*2 + 400 = 128000400

No. of Relational operations : 400*400*2 = 320000

No. of System Calls : 400*400*400*2 + 400*400*2+400*2 = 128320800

Total Increase in Load = Load increase for 128000400 arithmetic operators +
Load increase for 320000 relational operators +
Load increase for 128320800 System Calls

The Load increase for the arithmetic and relational operators and system calls is mentioned in table 1.1

Substituting those values,

Total Increase in Load = (128000400 * 3.025 e –9) + (320000 * 3.025 e –9)
+ (128320800 * 3.210 e –9)

Increase in Load for the execution of the entire process is = 0.8

The entire process consists of 1,60,000 tasks and hence

Increase in Load per task = 0.8 / 1,60,000 = 0.000005

Expected Load per machine = (1.0 + 0.8)/4 = 0.45

Increase in Load of Comp 1 = 0.25

Increase in Load of Comp 2 = 0.05

Increase in Load of Comp 3 = 0.15

Increase in Load of Comp 4 = 0.35

Percentage of Tasks allotted to Comp 1 = 0.25 /0.8 = 0.3125

Percentage of Tasks allotted to Comp 2 = 0.05 /0.8 = 0.0625

Percentage of Tasks allotted to Comp 3 = 0.15 /0.8 = 0.1875

Percentage of Tasks allotted to Comp 4 = 0.35/0.8 = 0.4375

Number of Tasks Allotted To Comp 1 : 0.3125 * 160000 = 50,000

Number of Tasks Allotted To Comp 2 : 0.0625 * 160000 = 10,000

Number of Tasks Allotted To Comp 3 : 0.1875 * 160000 = 30,000

Number of Tasks Allotted To Comp 4 : 0.4375 * 160000 = 70,000

This Procedure is detailed in the algorithm 1.1

Operators	**Symbol**	**Increase in Load/Operation**
Arithmetic Operators	Addition(+) Subtraction(-) Multiplication (*) Mod(%) Division(/)	**3.025 e –9**
Relational Operators	Equalto(=) Less Than(<) Less Than or Equalto (<=) Greater Than or equalto(>=) Not Equalto(!=) And (&&) Or (|) .	**3.025 e –9**
System Calls	Resource Allocating system call such as fopen(), brk() etc…. Utilizing system Calls such as read() and Write()	**3.210 e -9**

Table 1.1

These values were found on an experimental basis by substituting various values and finally those which provided the best results are presented above. The increases in load computed based on these values, ideally reflected the increase in load for other matrix operations, factorials and prime numbers.

Algorithm 1.1

Inputs: n, lpt, nt, l_i **Outputs:** ntm_i

Where

n = Number of Terminals in the cluster

lpt = Load Per Task

li = Current Load in Machine i

nt = Number of Tasks

ntmi = Number of Tasks for Machine i

elm = Expected load per machine

Expected load per machine

$$elm = \frac{\sum l_i + nt * lpt}{n}$$

Increase in load for node i

$$ilm_i = elm - l_i$$

Percentage of tasks allocated to node i

$$P_i = \frac{ilm_i}{\sum ilm_i} \quad (ilm_i > 0)$$

$$P_i = 0 \quad (\text{if } ilm_i <= 0)$$

Number of tasks allocated for node I : $ntm_i = P_i * nt$

IMPLEMENTATION AND EXPERIMENTAL RESULTS

We noticed that most of the tasks which could be split into well defined Independent modules and processed, had matrix operations as a major part in them. Hence we chose ***matrix multiplication*** to experiment our ideas with.

Implementation Procedure :

- The machines that are connected in the network and in working condition are checked and their IP addresses are entered in a data file.
- As the next step, the present load in each of the machines is retrieved.
- The approximate increase in load for the computation of a single task is calculated by scanning the procedure once and estimating the number of arithmetic operators, logical operators and system calls.
- The increase in system load for various operations as we have found, are as mentioned in the table 1.1.
- Once the load per task has been calculated, inputs are provided for the distribution algorithm and the number of tasks that are to be calculated in each machine is determined.
- These tasks are allotted to the appropriate computers and the results are sent back to the computer in which the request had initially been made and these results are combined to achieve the desired output.

The design was tested by multiplying 2 matrixes each having 400 rows and 400 columns. These inputs have been generated by a random function. The procedure was tested with the following configuration. Socket programs were used to communicate between computers.

Operating System	Linux Red Hat 7.0
Processor	Pentium-III 650Mhz
Hard disk Capacity	8 GB
RAM	128 MB
Communication between Computers	TCP/IP Sockets
File System	Network File System (NFS)

Results:

Normal Environment				Distributed Environment					
No. Of Computers	Initial load	Final Load	Time for Computation (Seconds)	No. Of Computers	Initial load Comp - 1	Initial load Comp – 2	Final load Comp – 1	Final load Comp -2	Total Time for Computation (Seconds)
1	0.0	0.92	127.127	2	0.0	0.0	0.48	0.47	63.728
1				2	0.20	0.0	0.58	0.60	84.210
1				2	0.20	0.20	0.68	0.69	66.937
1				2	1.0	0.0	1.0	0.95	129.245

Further work:

Currently we are working on the experimentation of an improved algorithm that consider the network bandwidth available, the congestion situation and node unavailability during runtime, to provide a fault tolerant, high availability cluster computation environment. The experimental results are currently unavailable. As future work, we plan to extend the algorithm for heterogeneous environments.

Conclusion:

In this paper we have detailed our distribution algorithm that aims to reduce processing time based on assigning load values to operators and system calls and using balanced load as an indicator of efficiency.

Reference:

1. "*Empirical Analysis of Overheads in Cluster Environment*", Brian K. Schmidt, V. Sunderam.
2. "*Massive Parallelism with Workstation Clusters – Challenge or Nonsense?*", Clemens H. Cap, 1993.
3. "*Optimizing Parallel Applications for Wide-Area Clusters*", Henri E. Bal, Aske Plaat, Mirjam E. Bakker, Peter Dozy, Rutger F. H. Hofman, 1997.