ANALYSIS OF THRESHOLD BASED CENTRALIZED LOAD BALANCING POLICY FOR HETEROGENEOUS MACHINES

Archana B.Saxena[1] and Deepti Sharma[2]

[1]Department of Computer Science, Jagan Institute of Management Studies, Affiliated to GGSIPU, Delhi
archanabsaxena@gmail.com
[2]Department of Computer Science, Jagan Institute of Management Studies, Affiliated to GGSIPU, Delhi
deepti.jims@gmail.com

Abstract

Heterogeneous machines can be significantly better than homogeneous machines but for that an effective workload distribution policy is required. Maximum realization of the performance can be achieved when system designer will overcome load imbalance condition within the system. Load distribution and load balancing policy together can reduce total execution time and increase system throughput.

In this paper; we provide algorithm analysis of a threshold based job allocation and load balancing policy for heterogeneous system where all incoming jobs are judiciously and transparently distributed among sharing nodes on the basis of jobs' requirement and processor capability for the maximization of performance and decline in execution time. A brief discussion of job allocation, transfer and location policy is given with explanation of how load imbalance condition is solved within the system. A flow of scheme is given with essential code and analysis of present algorithm is given to show how this algorithm is better.

Keywords

Heterogeneous Systems, Central Server, Threshold, Load Balancing, Load Distribution, Centralized Load Balancing, Sender Initiative, Receiver Initiative.

Abbreviations

S: System, **CS:** Central Server, **LB:** Load Balancing, **CM:** Capability Matrix, **LM:** Load Matrix, **Ji:** Job, **n:** Node, ***T***:Threshold, ***L:***Load, ***No:*** Node Overloaded, ***Nu:*** Node Under Loaded, ***SN:*** Suitable Node, **Nxj** : Non Executing Jobs, **Job_P**: Pending Job, **Job_E**: Executing Job, **M:** No of machines

DOI : 10.5121/ijait.2011.1505

1. INTRODUCTION

Load balancing is a mechanism that enables jobs to move from one computer to another within the distributed system. This creates faster job service e.g., minimize job response time and enhances resource utilization. Various studies, e.g., (Barak., Shiloh, 1985) (Khan, 2010), have shown that load balancing among nodes of a distributed system highly improves system performance and increases resource utilization. The successful load balancing algorithm depends on the following factors: job info, job requirement info, machine details, and current status of machines. Load balancing algorithms can be static, dynamic and adaptive. System state is used in dynamic load balancing algorithm to make load distribution decision where no such things are required in static algorithm. Here, load distribution technique is hardwired on the basis of system state. Where adaptive load balancing algorithm adapts their changes dynamically on the basis of system state (Niranja, Krueger and Singhal, 1992), load balancing can be distributed and non-distributed. In general, non-distributed based dynamic load balancing approach can be implemented by two ways: centralized and semi distributed. Centralized load balancing require fewer message to achieve load balancing condition with in system (Ali M. 2010) as nodes have to interact with only CS not with any other neighbor node.

We have adopted centralized dynamic approach for load balancing. In this approach we consider a system where ***M*** heterogeneous machines are registered with CS. These entire machines are connected through high speed network with each other. Centralized Policy by considering system state is used for load distribution and load balancing in this approach. Load distribution will depend on processor capability, job requirement and threshold of machine. Load balancing is accomplished by transferring load from heavily loaded system to lightly loaded system. This movement can be initiated by heavily (Sender initiative policy) loaded machines or by lightly (Receiver initiated policy) loaded machines. Load balancing policy will include: information, transfer and location strategy. This proposed strategy is best suited for small size network as it is proved through a study that centralized load balancing are best suited for network where node strength < 100.

The proposed load balancing algorithm has following objectives:

- Greater overall improvement in the system by reducing job response time and increasing system throughput.
- To have performance endurance in system in case of partial failure with system.
- To maintain system stability so that system performance does not decline and system spends more time in executing jobs rather than passing jobs to other nodes.

Parameters used in Algorithm:

Load Measurement: In the current scheme CPU queue length is used as load descriptor because it is simple to obtain and can give good idea about time required to finish currently assigned work as we have assumed (execution time =job processing requirements / processing capabilities).

Performance Measurement: Proposed algorithm is both system performance oriented and user performance oriented. System performance oriented is measured through job wait time and job response time is indicator of user performance.

System Stability: In current scheme system stability depends on the time CS server is spending in load balancing among nodes, it should not be more than certain percentage (p) of its total working time.

2. RELATED WORK

In cluster computing era, load balancing has gained interest among researchers and lots of work had been done in this regard to provide better systems. **Mohammed A. M. Ibrahim [2010]** used a mobile agent based approach to improve load balancing of parallel task in heterogeneous computing environment in which master process send mobile agents to every workstation to measure the load level of local machine and send it to master server. In second phase master server selects low loaded machine on the basis of threshold and form a cluster of such machines. Finally this adaptive threshold policy adjusts the threshold according to the changes in system. **Ali M. Alakeel [2010]** presented and analyzed factors viz. load estimation, information strategy, transfer strategy, job resource requirement, performance indices and other issues which need to be considered in building load balancing policy for the system. **Leping Wang and Ying Lu [2010]** proposed a power management system for heterogeneous soft real time clusters. A threshold based approach makes three important design decisions ordered server list, server activation threshold and work load distribution. Server activation threshold and work load distribution are designed to achieve optimal power consumption. **Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma [2008]** have studied various Static(Round Robin and Randomized Algorithms, Central Manager Algorithm, Threshold Algorithm*)* and Dynamic (Central Queue Algorithm, Local Queue Algorithm) Load Balancing Algorithms in distributed system and their results shows static load balancing algorithms are more stable with such parameters. **Neeraj Nehra, R.B. Patel, V.K. Bhat [2007]** have proposed a DDLB(Dynamic Distributed Load Balancing) scheme for minimizing the average completion time of application running in parallel and improve the utilization of nodes. In order to achieve their target they will make use of MA (Mobile agent) to distribute load among nodes in a cluster. **Satoshi Azuchi, Sojeong Hong, Jinoh Kim [2006]** provided a new distributed priority-based computing architecture to utilize the computing resource more efficiently and be tolerant against single point of failure. Each computing server's load balancing is achieved by dynamically changing its priority and control task acceptance rate based on a computing server's priority. **Helen D. Karatza and Ralph C. Hilzer [2002]** have worked together to find a load sharing policy in heterogeneous distributed environment where half of the total processor have double of the speed of others and excepts only two types of jobs: first class and Generic. **Kalim Qureshi and Masahiko Hatanaka [2000]** presented a short survey on HDC systems and identified load balancing problems in parallel raytracing in such systems. Mutual authentication and capability based access control model is used for security. **Niranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal Ohio State Universit [1992]. Anna Hac aud Theodore J. Johnson [1986]** has addressed a Load balancing problem in LOCUS distributed file system and proposed a CSS (centralized synchronization sites) policy for optimal process and read site placement. **Mohammed Javeed Zaki, Wei Li, and Srinivasan Parthasarathy [1997]** have worked on the concept of Dynamic Customized Load Balancing for heterogeneous network. Their experiment result shows that different load balancing schemes(Local, global, centralized and distributed) are best for different application under varying processor, program and system parameter and therefore application-driven customized dynamic load balancing becomes essential for good performance.

3. FUNCTIONAL DESCRIPTIONS

In centralized load balancing policy, one centralized node CS (Central Server) acknowledges the incoming jobs. Each job *ji* has its arrival time and processing requirements associated with it. CS searches the Capability Matrix (CM) to assign the job to suitable node. A suitable node *N* is eligible to process current job if **N.capability > *Ji.capability***.

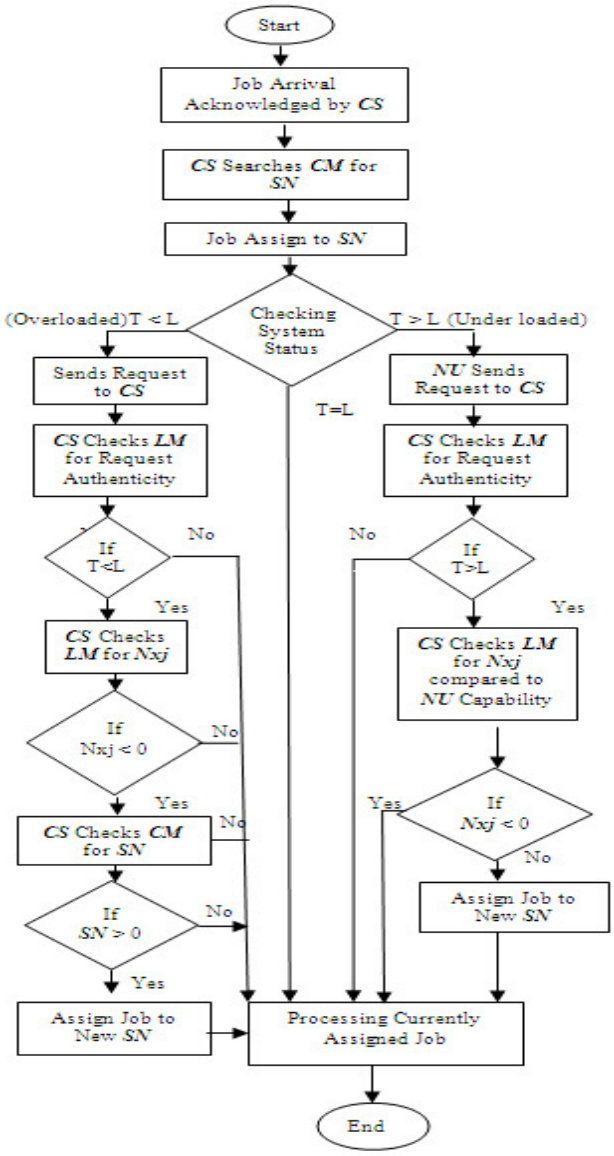

Fig.1. Flow Chart showing Functional Description of the Proposed Model

Once the job is assigned to suitable node, the system can be in any of the three states:

- **Load balanced**: where Load = Threshold
- **Over Loaded** : where Load > Threshold
- **Under Loaded** : where Load < Threshold

If the system is in load balanced condition no further step is required and all processors will execute assigned jobs. Load imbalance condition can be treated by two ways:
Sender Initiative Load Balancing (Overloaded Node: Load > Threshold): when a Node ***'No'*** would like to share its workload, it sends a request through communication link to ***'CS'*** for load sharing. In order, to respond the requested node ***'No', 'CS'*** proceeds with following actions:

1. To confirm job status of ***'No'***, ***'CS'*** checks ***'LM'*** and request rejected if ***'No'*** is not overloaded, otherwise move to step 2.
2. ***'CS'*** checks '***LM'*** for ***'Nxj'*** with ***'No'*** (Only non-executing jobs are considered for migration). If there is no such job then procedure ends otherwise move to step 3.
3. After selecting the job for migration, ***'CS'*** looks for (Through ***'CM'***) node that can process selected job. Move to step 4 if one of them is found to have enough processing capabilities to execute selected job.
4. The last step involved job migration from Overloaded (***'No'***) to newly selected node and updating '***LM'.***

Receiver Initiative Load Balancing (Under loaded Node: Load < Threshold): when an under loaded Node ***'Nu'*** would like to share its workload, it sends a request through communication link to ***'CS'*** for load sharing. In order to respond the requested node 'Nu'***, 'CS'*** proceeds with following actions:

1. To confirm job status of ***'Nu'***, ***'CS'*** checks '***LM'*** and request rejected if ***'Nu'*** is not under loaded, otherwise move to step 2.
2. ***'CS'*** checks the processing capabilities of required node ***'Nu',*** in order to decide what sort/type of job can be assign to his node ***'Nu'.***
3. ***'CS'*** checks ***'LM'*** for non executing processes/jobs pending with any node where job requirements matches with processing capabilities of node ***'Nu'***. (In present system only non-executing jobs are considered for migration). If there is no such job then procedure ends otherwise move to step 4.
4. The last step involved job migration from under loaded (***'Nu'***) to newly selected node and updating ***'LM'***.

4. COMPONENTS OF PROPOSED MODEL

A model for providing load balancing in heterogeneous systems consists of information, transfer and location strategy. The components in each strategy are shown below:

4.1 Information strategy

It is the information hub of the algorithm which is responsible for collecting information about the nodes and supplying required information to the transfer and location strategy. Proposed algorithm maintains required information in the form of three tables:

4.1.1 Job Queue (Arrival time, Processing requirements):

A job queue is a collection of pending jobs within the system at different clock timing.

Table 1. Job Queue

Arrival Time	Processing Requirements
0	1590
2	716
3	2076
10	1581
11	2412
18	1263

4.1.2 Capability Matrix (Mac No, Model No, Clock speed, Capability (15 * Speed), Execution Threshold (50 * Capability)): It saves details of registered nodes.

Table 2. Capability Matrix

Mac No	Model No.	Clock Speed	Capability	Execution threshold [CaP. * 50]
Mac I	Core 2 Duo 5250	1.5 GHz	45	2250
Mac II	Core2 DuoT5200	1.60 GHz	48	2400
Mac III	Pentium 4-M 2.2	2.2 GHz	33	1650
Mac IV	Pentium 4 HT 2.4C	2.4 GHz	36	1800
Mac V	Pentium 4 HT 2.6C	2.6 GHz	39	1950
Mac VI	Pentium 4 HT 2.8C	2.8 GHz	42	2100
Mac VII	Pentium 4 HT 3.2	3.2 GHz	48	2400
Mac VIII	Pentium 4 HT 3.4	3.4 GHz	51	2550
Mac IX	Pentium 4 HT 651	3.4 GHz	51	2550
Mac X	Pentium 4 HT 661	3.6 GHz	54	2700
Mac XI	Pentium 4 Extreme Edition 3.73	3.73 GHz	56	2800

4.1.3 Load Matrix (Arrival Clock, Job Requirement, Start Time, Machine No., Execution Time (Job requirement/processor capability), End Time (Start time+ Time Required for Execution) and status):

It saves load of a machine at a particular point of time.

Table 3. Load Matrix

Clock	Job Requirement	Start Time	Machine No.	Execution Time	End Time	Execution Status
0	1590	0	Mac-III	1590/33=48	48	E
2	716	2	Mac-IV	716/36=20	22	E
3	2076	3	Mac-VI	2076/42=49	52	E
10	1581	10	Mac-V	1581/39=41	51	E

4.2 Transfer Strategy

This component of the proposed model is responsible for accepting incoming jobs and making a decision where it should be transferred for execution. If there is load imbalance in the system then this strategy decides which jobs to be moved from one location to another and thus balancing load within system.

4.2.1 Job Allocation Policy

Jobs are allocated among eligible processors according to job allocation policy and related information is saved in Load Matrix. A processor can be selected for job execution if any one or all of the following conditions are met.

- Job requirements are within processing capabilities of the processor.
- If two or more processor can execute the same job then fastest processor will execute the job.
- If load is more than execution threshold then next fastest processor or the one who is ready to work will execute the job.
- If all the processor that are capable enough to process the job are busy then compare the pending workload of these processor and then allot the job to the one who is least loaded.

4.2.2 Threshold Policy

It is an integer value that represents load status of machine. A job is transferred if queue length of a local node exceeds a certain threshold (***T***) otherwise executed locally.

4.2.3 Load Balancing Request

Proposed algorithm offers load balancing as sender initiated (Overloaded Node) and receiver initiated (Under Loaded Node).

- Sender-Initiated (T<L): When a new job is assigned to the machine, it checks its status and if overloaded can launch balancing request to CS and ask to provide node to share some load.
- Receiver-Initiated (T>L): When a finish job departs from machine it checks its status and sends a request to CS and shows its willingness to accept more load.

4.2.4 Job Transfer Policy

Algorithm adopts consider-new-only approach where only newly arrived and non executing jobs are considered for transfer.

4.3 Location Strategy

The main job of Location strategy is to decide destination node for Load balancing. In current scheme this decision is taken by CS by measuring current load (Job queue length which includes executing and pending jobs) and processing capabilities of the node. A node is selected under location strategy if any of the following conditions are met:

- If processing capabilities of current job is more than processing requirements of transferring job.
- If transfer of current job to selected processor will not cause its load to go above threshold (T).

- Among all the selected processor fastest processor, shortest distance from old node and with the smallest pending queue length will be chosen.

Once the job is transferred to the required location, information is updated in load matrix.
Required information for various decisions by transfer strategy and location strategy will be provided by information strategy.

5. ALGORITHM

Based on the above components of proposed model, an algorithm is designed. This algorithm consists of two procedures: assignment and load balancing.

5.1 Assignment

Assignment algorithm is used to assign incoming jobs to nodes for execution with the help of Job_Queue and Capability_Matrix.
Assignment is executed through two main functions: Random_jobs ( ) and Assignment( ).

Random_jobs ( ):

1. Generates job arrival time, saves it in an array and sort array in ascending order.
2. Generates job requirement time in an array.
3. Creates a 2D array for job arrival clock and processing requirement and finally sort the array.

Assignment ( )

1. FCFS (First Come First Serve) assignment strategy considers capability matrix for assignment.
2. Writes information in Load_Matrix.

```
Random_Job(No_of_jobs)
{
// generates job arrival time and stores it in an array

 Random ran_job = new Random();
        for (int i =0; i<no_of_jobs;i++)
        {
                arr_time[i] = ran_job.nextInt(100);
        }
// sorting job arrival time in ascending order
        for (int i=0; i < no_of_jobs ; i++)
        {
                for(int j=i+1; j < no_of_jobs ; j++)
                {
                        if(arr_time[i]>arr_time[j])
                        {
                        mid1=arr_time[i];
                        arr_time[i] = arr_time[j];
```

```
                    arr_time[j]=mid1;
                    }
            }
    }
```

// generates job processing requirements and store in an array

```
Random ran = new Random ();
    for (int i=0; i < no_of_jobs ; i++)
    {
        job[i] = ran.nextInt(5000);

    }
```

// creating a 2d array

```
int [ ][ ] job_arr_time_combo= new int[No_of_jobs][2];
    for (int i=0 ; i < no_of_jobs ; i++)
    {
        for(int j=0 ; j < 2 ; j++)
        {
            if(j= =0)
                job_arr_time_combo[i][j]=arr_time[i];
            else

                job_arr_time_combo[i][j]=job[i];
        }
    }
```

Calculate current Load(Ni)

// calculating present load of node Ni and checking its execution status

```
{
int c_load = "select count(job_requirement) from Load_matrix where mac_no = "Ni" and
Execution status = "E" or Execution status = "P" "
Return c_load  // to assignment ( )
}
```

Assignment ( )

```
int processor[ ][ ]= select MacNo, Processing_capabilities from Capability_Matrix
```

// arrange processor array in descending order

```
for (int i=1; i< no_of_jobs;i++)
    {
    for(int j=i+1; j< no_of_jobs;j++)
    {
```

```
                if(job[i]<job[j])
                {
                        mid1=job[i];
                        job[i] = job[j];
                        job[j]=mid1;

                }
        }
        }
for (int i=0; i< no_of_jobs;i++)
{
        for (int j=0; j< processor.length();j++)
        {
                if (jobi.req < nj.cap)
                {
        // calculating present load of machine
                int c_load= Calculate current Load(Nj)
        //if current load < threshold, assign current job to that node and updates load_matrix
                if (c_load < T)
                {
                        nj.assigned();
                        nj.update_Load_Matrix();
                }
}       }       }
```

5.2 Load Balancing

In the current scheme load balancing operation can be initiated by over loaded or under loaded nodes. In both cases, request has to be routed through central server. Server will listen to request checks its validity, check for availability of non executing nodes and finally calls load balancer accordingly. Load balancing is done through following functions:

Overloaded ( ) / Underloaded ( )

1. Validates request made by node.
2. Checks load and capability matrix for executing jobs.
3. Calls load balancer ( ).

Load_Balancing_Request(Ni)

1. Execution status (E/P) of a node is compared with threshold of machine.
2. Returns an integer value (0 for load balanced and 1 for under loaded or over loaded nodes).
3. Updates the status of requested node in system.

Load_Matrix_Nxj(No)

1. Accepts Mac_No of Overloaded Node
2. Returns status of non executing jobs that can be transferred for balancing load within system.

Check_CM(Job_req)

1. Accepts job_requirement and searches Capability Matrix.
2. Return Mac_no or 0, depending on the availability of node.

Load_Balancer (No,New_n,Job_id)

It accepts Mac_No of overloaded node, Mac_No of new node and Job Id of transferred job for executing transfer procedure.

OverLoaded()

```
{
// validating request made by node

int status = Load_Balancing_Request(No);
if (status = = 1)
{
        // checking load matrix for non executing jobs
        int job_status = Load_Matrix_Nxj (No);
        if  (job_status = =1)
                // checking capability matrix for best suitable node for executing selected jobs
int node_status=Check_CM();
if (node_status = = 1)
{
// calling load balancer to balance load
        Load_Balancer()
}
} } }
```

UnderLoaded()

```
{
// validating request made by node
int status = Load_Balancing_Request(Ni);
if (status = = 0 )
{
// checking for non executable jobs from load matrix
        int job_status = Load_Matrix_Nxj (No);
        if  (job_status = =1)
        {
                // checking suitable node from capability matrix
                int node_status=Check_CM();
                if (node_status = = 1)
                        {
                        // calling load balancer for balancing load
                        Load_Balancer()
                        }

                } } }
```

```
Load_Balancing_Request(Ni)
{

int Job_P =  "Select count(execution status) from Load_Matrix where Mac_No= Ni  and execution status = "P" "

if (Job_P >= 1 )
{
return 1 ;
}  }

Load_Matrix_Nxj(No)
{
int job_count[ ][ ] ;
Statement stmt;
Resultset rs = stmt.executeQuery(select job_requirement and Job_Id from Load_Matrix where Mac_no = "No" and Execution_status="P")

//move till end of result set
while(rs.next())
{
// entering data in 2d array
for (int i=0;i<rs.getRows();i++)
{
        for (int j=0;j<2;j++)
        {
                // In first column add job_id and in second column add job_requirement
                if (j= =0)
                        job_count[i][j]=rs.getString("job_ID");
                else
                        job_count[i][j]=rs.getString("job_requirement");

}        }

// Requirement details and Job_id of all non executing jobs are saved in job_count array

if ( job_count.length( ) = =0)
        return 0; // No job is available for transfer
else

        return 1; // Nxj are available for transfer

} }
Check_CM(job_req)
{
for (int i=0;i<Node.count();i++)
{
if(job_req < ni.capability)
```

```
        {
        int count = select count(Execution status) from LoadMatrix where Mac_no=ni and
        Execution status= "E"
        //Query to check availability of current node through Load Matrix
        If (count = 0)
                {
                return ni; // Node is free to take transfer
                }
        }
        return 0;
} }
```

Load_Balancer(O_node, N_node,job_id)

```
{
Delete from Load_matrix where jobId = job_id and MachineNo = No;
Insert into Load_Matrix values() ;
}
```

6. ANALYSIS OF THE ALGORITHM

To analyze the algorithms, let there be 'k' jobs and 'm' number of servers.
Thus, T(n) = Sorting + Assignment

Sorting Algorithm: Here for every i = 0 to m-1 there are m-1 to 1 comparisons i.e. (m-1) + (m-2) +----------2+1 = m (m-1)/2=$m^2/2$ comparisons.
So, **time complexity = T ($m^2/2$).**

6.1 Analysis for Assignment Algorithm

Best Case: When there is a single server, there can be maximum 1 comparison.
Worst Case: When there are m servers, there can be maximum $1.m_1+1.m_2$ ----------$+1.m_n$.
If there are 'm' servers then the number of comparisons will be **'m'**.
Following are given, some cases with different values of k (number of jobs) and m (number of servers).
Let T (n) = sorting + assignment

Case 1: k=1, m=1
T(n) = T(0)+T(0);
T(n) = 2T(0)
T(n)=1

Case 2: k=1, m>1
T(n)= T($m^2/2$)+T(m)
T(n)= O($m^2/2$)

Case 3: k>1, m=1
T(n) = T(1) + T(k)
T(n) = O(k)

Case 4: k>1, m>1
T(n) = T($m^2/2$) + T(km)
T(n) = O($km^2/2$)

6.2 Analysis for Load Balancing Algorithm

Load Balancing

For each server there will be 'n' comparisons and the **complexity remains T(m).** But if there are unbalanced nodes in the server then for every unbalanced nodes 'j' there will be (m-j) comparisons.Let the number of unbalanced node = j. Following are the cases given with different values of m (number of servers) and j (number of unbalanced jobs). Let the constant time T(1) for inserting and deleting data in the database in loadbalancer( ). Query processing time is considered as ***constant***.

Case 1: When j=0, m>1
$T(n) = T(m^2/2)+T(m)$
$T(n) = 2T(m^2/2)+T(0)$
$T(n)= O(m^2/2)$, c=2
Case 2: When j=1, m>1
$T(n) = T(m^2/2)+T(m)+T(m-p)+T(1)$,
Where p is the position of
Unbalanced node in the cluster
$T(n) = 3T(m^2/2)+T(1)$
$T(n)= O(m^2/2)$, c=3
Case 3: When j>1 and m>1
$T(n)= T(m^2/2)+T(m)+T(j(m-p))+T(j.1)$
,j<=m
$T(n) = 4T(m^2)$
$T(n) = O(m^2)$, c=4

7.Conclusion

In this paper, a threshold based job allocation policy for heterogeneous systems is presented. In this, all incoming jobs are compared with the registered nodes and jobs being allocated according to their capability and threshold value. Load imbalance conditions are treated by Sender Initiative (Overloaded Node: Load > Threshold) or Receiver Initiative (Under loaded Node: Load < Threshold) Load Balancing. Algorithm and its analysis are also given. The main objective is to provide a Load Balancing mechanism so that overall performance can be maximized.

In the future, through simulation we will try to prove that our load balancing scheme improves work throughput.

REFERENCES:

[1] Alakeel Ali M (2010), "A guide to Dynamic Load Balancing in Distributed Computer system", IJCSNS (International Journal of Computer Science and Network Security), VOL.10 No.6, June 2010.

[2] Mohammed Ibrahim A. M. (2010), "Cluster of Heterogeneous computers: Using Mobile Agent for improving Load balance", International Journal of Science and Technology Education Research Vol. 1(7), December 2010, pp. 143 – 146.

[3] Wang Leping, Lu Ying (2010), "An Efficient Threshold Based Power Management Mechanism For Heterogeneous Soft Real Time Clusters", IEEE Transactions on Industrial Informatics, VOL. 6, NO. 3, AUGUST 2010.

[4] Khan Z., et al (2010), "Performance Analysis of Dynamic Load Balancing Techniques for Parallel and Distributed Systems," International Journal of Computer and Network Security, vol. 2, no. 2, February 2010.

[5] Sharma Sandeep, Singh Sarabjit and Sharma Meenakshi (2008), "Performance Analysis of Load Balancing Algorithms", World Academy of Science, Engineering and Technology, 2008.

[6] Nehra Neeraj, Patel R.B. and bhat V.K. (2007), "A Framework for Distributed Dynamic Load Balancing in Heterogeneous Cluster", Journal of Computer Science Volume3, Issue 1 Pages 14-24.

[7] Branco Kalinka R. L. J. Castelo, et al (2007), "PIV and WPIV: Two New Performance Indices for Heterogeneous Systems Evaluation".http://www.dcc.ufla.br/infocomp/artigos/v5.4/art08.pdf
[8] Lan Zhiling, Taylor Valerie E. (2007), "Dynamic Load Balancing of SAMR Applications on Distributed Systems", Proceedings of ACM/IEEE conference on SuperComputing.
[9] Othman Ossama, and Schmidt Douglas C.(2007) "Optimizing Distributed System Performance via Adaptive Middleware Load Balancing". http://www.cs.wustl.edu/~schmidt/PDF/load_balancing_om_01.pdf
[10]Azuchi Satoshi, et al (2006), "Computing Service Architecture: central monitor-absence load balancing", May 2006. http://www-users.cs.umn.edu/~jinohkim/docs/loadbalancing.pdf
[11] Godfrey Brighten, Lakshminarayanan Karthik and et al (2004), "Load Balancing in Dynamic Structured P2P Systems", In Proc. IEEE INFOCOM, HongKong.
[12] Karatza Helen D. and Hilzer Ralph C. (2002), "LOAD SHARING IN HETEROGENEOUS DISTRIBUTED SYSTEMS", Proceedings of the 2002 Winter Simulation Conference: 489-496
[13] Parent Johan, Verbeeck Katja and Lemeire Jan (2002), "Adaptive Load Balancing of Parallel Applications with Reinforcement Learning on Heterogeneous Networks", Published in Proc. of Int. Symposium DCABES 2002, Wuxi, China, Dec 16th – 20th, 2002.
[14] Othman Ossama, O'Ryan Carlos, and Schmidt Douglas C, (2001) "An Efficient Adaptive Load Balancing Service for CORBA", IEEE Distributed Systems Online, Feb 2001.
[15] Kalim Qureshi and Masahiko Hatanaka (2000)," An introduction to load balancing for parallel ray tracing on HDC systems", Special Section: Computational Science Tutorials, Current Science, VOL. 78, No. 7, 10 APRIL 2000
[16] Shahzad Malik (2000) "Dynamic Load Balancing in a Network of Workstations", Research Report for Parallel Processing Course, Carleton University, November 2000.
[17] Zaki Javeed Mohammed, Li Wei and Parthasarathy Srinivasan, (1997), "Customized Dynamic Load Balancing for a Network of Workstations", Journal of Parallel And Distributed Computing 43, 156–162 (1997).
[18] Shivaratri Niranjan G., Krueger Phillip and Singhal Mukesh (1992), "Load distribution for locally distributed systems", Journal: Computer, Vol 25 Issue 12, December 1992, IEEE Computer Society Press Los Alamitos, CA, USA
[19] Hac Anna, Johnson Theodore (1986), "A Study of Dynamic Load Balancing in a Distributed System", SIGCOMM '86, Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols.
[20] Barak A., Shiloh A. (1985), "A Distributed Load-balancing Policy for a Multicomputer", Journal of Software-Practice and Experience, Vol. 15, No. 9, pp. 901-913, September 1985.

Deepti Sharma is an Asst. Professor in Department of Computer Science at Jagan Institute of Management Studies (Affiliated to Guru Gobind Singh Indraprastha University), Rohini, Delhi. She is MPhil, MCA and pursuing her PhD (Computer Science). She has more than seven years of teaching experience in the areas of C, C++, Data Structures, Operating System and DBMS. Her research areas include "Load Balancing in Hetrogenous Web Servers" and "Mobile Banking" on which papers have been published in National and International conferences and journals. Various seminars, workshops and FDP (AICTE) have been attended.

Archana B Saxena is working as Assistant Professor and Web Administrator in Jagan Institute of Management Studies (JIMS) since Oct 2004. Presently she is perusing PhD in Computer Science from GNDU on topic "Load Balancing in Heterogeneous Distributed Systems". She has attained MPhil Degree in CS from MKU, in 2006, MCA from IGNOU in 2003 and Bachelors in Commerce from Delhi University. She has served as guest Lecturer in ARSD, Delhi University and Faculty as NIIT. She has written various papers on Load Balancing, Mobile Computing and Data Mining published in various journals and conferences.