Efficient Dynamic Load Balancing Algorithms for Multiclass Jobs in Peer to Peer Networks

R Manjula, Pavankumar Kolla, Kolla Haripriyanka

***Abstract*— Load balancing is an important concept for the efficient operation of peer to peer networks. We present three new dynamic load balancing algorithms, that performance guarantee optimal. The first, Queue with consistent hashing algorithm, that balances the multiclass jobs are executed on the specified range of peers having its own capability. The second, Rate with random adjustment algorithm balances the multiclass jobs are executed on specified peer. The third approach combines the both approaches to schedule jobs on specified peer. Here, we are taken similar jobs under single class, and the different classes form a multiclass job.**

***Index Terms*— Scheduling, load balancing, peer to peer systems, queuing theory, mean response time.**

I. INTRODUCTION

A core problem in peer to peer systems is the distribution of processes to be stored or computations to be carried out to the nodes that make up the system. These systems might have various processors with different processing capabilities, connected by two-way (either way) communication links, and each has their own resources/buffers. In those systems, if some hosts remain idle while others are too busy with the processes, system performance will be affected significantly. To avoid this, load balancing is regularly used to distribute the jobs and improving its performance measures like mean response time. Several influencing factors are depending upon the designing issues of load balancing algorithms, for example, network bandwidth, network topology, arrival rates of jobs at each processor in the system. Load balancing algorithms can be classified as either dynamic or static.

To scale performance, Dynamic Load Balancing distributes IP traffic across multiple cluster hosts. The client does not need to wait for long time to complete its process. So, the throughput is increased, CPU overhead and response time is reduced. It also ensures high availability by detecting host failures and automatically redistributing traffic to the surviving hosts. Dynamic Load Balancing provides remote controllability and supports rolling upgrades from an operating system.

Manuscript received by March 11, 2011; Manuscript revised by April 8 2011. This work was supported in vit university,vellore,india.

Prof R.manjula working as Associate Professor and also as Ph.d Candidate affiliated with School of Computing Science and Engineering at Vellore Institute of Technology, Vellore, India. (e-mail: rmanjula@vit.ac.in)
Pavankumar Kolla pursing his M.Tech(cse) from School of Computing Science and Engineering at Vellore Institute of Technology, Vellore, India.(phone no:+91-9490636505,e-mail:pavankumar39@gmail.com).
Kolla Haripriyanka pursuing her b.tech in sri mittapalli college of engineering,Guntur,ap,india.(e-mail:kolla.haripriyanka@gmail.com)

A Dynamic algorithm proceeds further based on the status of the system at that instance. Here the status may be related to any class of information [2] at each processor. Where as a static algorithm [2] do its jobs by a fixed policy irrespective of system status. We can reduce the additional computation overheads by introducing the proxy like a coordinator in the distributed systems, which have the information of all the peers and takes the decision according to the situation of processors. By introducing a proxy we will compensate the system performance by reducing the additional over head. To understand the distributed peer to peer system working model, each node (scheduler) independently handles all its overheads, such as computation and communication overheads.

Some solutions were proposed to decrease the communication overheads, by estimating the current condition of the peers in the system. From the literature survey we analyzed how effectively we can use randomization in the load balancing problem. On observing the simulations of Li-Kameda [3] and FD algorithm [4], still the computation overheads are high. As an example, Li-Kameda algorithm takes more than 400 seconds (approximately) and FD algorithm is taking more than 105 seconds for solving a general case [4]. If we use LBVR algorithm [2], which was proved that the convergence rate of LBVR was super-linear. Convergence rate give an idea of computation overheads. As the convergence rate increases computation overheads will be reduced significantly. It is observed that the LBVR algorithm generates optimal solution within 0.1 seconds for distributed systems [2]. Based on LBVR we propose the 3 distributed dynamic load balancing algorithms. And also we are extending our work in this paper for peer to peer networks also. The new algorithms are QARVR, RRAVR and QCHAVR. Now in our paper we show how we implement these algorithms for multi class of jobs for processing. We also consider the peer to peer network nodes also.

These algorithms analyzed in terms of minimization of mean response time, the load balancing ability. Based on these proposed algorithms the performance of the distributed dynamic load balancing will give the optimal solution. The Proxy mainly perform the following five operations:

A. Cluster Identification
B. Load Analyzer

C. Efficiency Tracker
D. Overhead Reducer
E. Performance Reporter

A.This module is for identifying the cluster hosts using the IP address, which is a unique for each cluster host. Proxies use this IP address for sending the requests to one particular server for processing.

B.This module identifies the configurations of each cluster hosts and nodes. So, the workload is fully distributed according to the configurations. It also identifies the server failure. If any server give a failure report, immediately no one requests are send to that server. That failure server's workload is shared by other servers that will do the same process. The network continues its function in good manner.

C.This module which keep track of each cluster hosts and nodes priority, which maintain the cluster will keep maximum possible numbers of task's (higher is better). It will keep the system more effective and efficient.

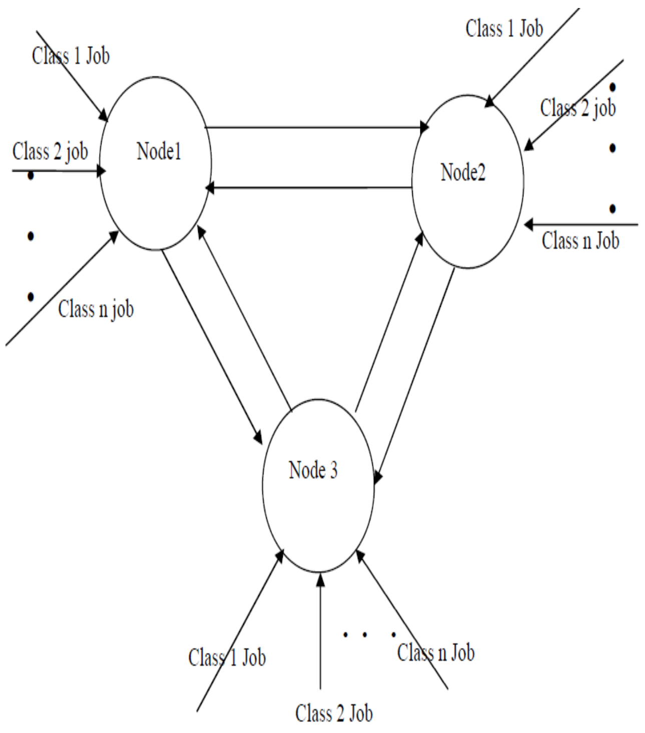
D.This module is for removes the cluster hosts and nodes which have low priority and require more time for perform the workload. Due to this process we can avoid bottlenecks situation by using a centralized dispatcher. It also used to maintain a throughput as high. Response time is inversely proposal to the throughput i.e.) if the throughput is high then the response time is low. So, the clients did not wait longtime.

II. THE PEER TO PEER SYSTEM MODEL AND DYNAMIC LOAD BALANCING ALGORITHMS CLASSIFICATION

We present a general system model peer to peer [1] in the design of the algorithms. We consider a general peer to peer system shown in Fig. 1.Peer to Peer network: The system have N peer to peer heterogeneous nodes having a processor, which represent the node having different computational capabilities, and these nodes are communicated with each other via communication links with full duplex communication. Here there are n different class of jobs for example such as video, audio, data are submitted at each node

A class of job arriving at peer j may either be executed or performed locally or transfer through to another peer k(k Є N) for remote execution. The service time of a particular class of job is a random variable that follows an exponential distribution with mean 1/μj, where μj denotes the average class job service rate of peer j and represents the rate (in jobs executed per unit time) at which peer j operates when busy. The queue having different class of the jobs in each peer is first come first serve policy and the buffer space is infinite. Once a class of a job starts executing in a node, it is allowed to complete execution without interruption and cannot be transferred to another peer at that time.

Figure 1.an example peer to peer network



In this model, we assume that there exist a communication delay occurred when a class of a job is transferred from one peer to another before the class of a job can be executed in the system and denote pjk(t) as the class job flow rate from peer j to peer k (k Є N) at time t. Further, we assume that each link (j,k) can transfer the load at its own transmission capability .We denote k as the set of transmission capacities of all the links and kkj as the transmission capacity of a link (j,k), kjk Є k.

A peer topeer network system for load balancing algorithms, the model for a node is consists of a scheduler, an infinite buffer to hold the different class of jobs, and a CPU. The scheduler is to schedule the jobs arriving at the peer such that the mean response time of the jobs is a minimum.

III. ALGORITHMS

All the peer to peer to peer network systems information such as ip address ,specifications of each processor(speed, memory, the class of problems it can execute) is shared among the network before starting its operation. If any new node wants to be in the network, the node must share the information about is specification. Each peer has a database that stores about the specification of all nodes in that network . Each node has a proxy scheduler in it. We have a database table that stores the utilization corresponding to each node is shared and update the query tables when ever change is occurred.

First, the jobs are submitted to the proxy scheduler in the node. After then the proxy will decide based on class of job which has to execute on a node such that load is balanced. The scheduler in each node follows the three algorithms

(I) when system utilization is low, RRAVR performs much better than QCHAVR and QARVRW with a relatively longer status exchange interval, which means less communication overhead.

(II)When system utilization is very high, QCHAVR performs the best among the three load balancing policies with high communication overheads.

(III)When the system utilization changes rapidly, QARVR is suitable and can achieve good performance with moderate communication overhead.

The parameters are:

N=Number of nodes in the peer to peer networks,

U=percentage of utilization of processor,

= (number of particular class of jobs executing currently/maximum number of jobs processor execute)*100,

Padre=peer to peer network IP address,

Portno=specifying the particular class of job is going to Executed on a processor,

Jobid=class name of the job,

Queue=data structure storing the jobs in first in first order,

QL=number of jobs waiting in the queue, Cjcount=class job count.

(i):Rate with random adjustment algorithm via Virtual Routing (RRAVR):

```
Procedure for proxy (jobid, portno, ipaddr1)

{

do

Step1:

Decide the class of job based on job id with port no and identify the node ipaddr1.

Step2.

Query the database tables that store the information about the utilization corresponding to the jobid (class of job) and corresponding processors ip addr in which minimum utilization is there. Result ip address are sent to step 3

Step3.

If the array of node ips of the result is one of the nodeip1(its own ip)

Then allocate the job to node, if the ip is itself then

step4;

break;

Else(result array.length == 1)

Then allocate the job to resulted node and return step4;

break;

Else(result arry.length>1)/*That is two or more nodes having the minimum value */

Randomly allocate any of the processors available and return step4

break; Step 4:
Cjcount=cjcount+;

Update the utilization factor based on updated cjcount;

Send this updated value to all other nodes;

Step5:

Return the ipaddr;

end;

}
```

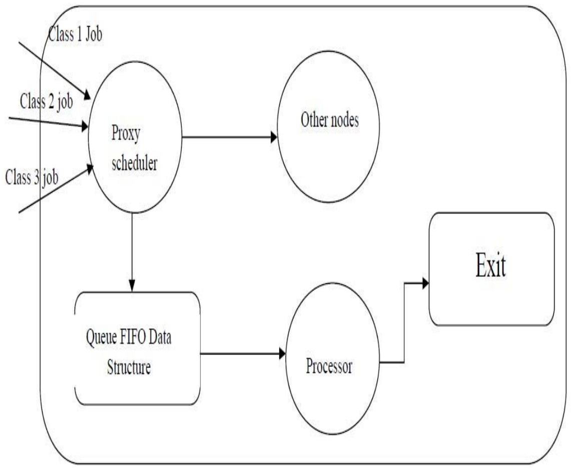


Fig 2 RRAVR algorithm

(ii) Queue with consistent hashing algorithm via Virtual Routing (QCHAVR):

```
Procedure for proxy(jobid,portno,ipaddr1)

{

 begin

Step1:

Decide the class of job based on jobid with portno

And identify the node ipaddr1.

Step2.

Query the database tables that store the information
about the utilization corresponding to the jobid (class of
job)and corresponding processors ipaddr in which
minimum ql(queue length) is there.result(arry) ip address
are sent to step 3

Step3.

If the array of nodeips of the the result is one
of the nodeip1(its own ip)

Then allocate the job to node itself and goto step4;

break;

Else(result array.length == 1)

Then allocate the job to resulted node and retun step4;

break;

Else(result arry.length>1)/*That is two or
more nodes having the minimum value */

allocate any of the processors available with
result from hash function and return step4

break;

Step 4:
Cjcount=cjcount+;

Update the utilization factor based on updated cjcount;

Send this updated value to all other
nodes;

Step5:

Return the ipaddr;

Return
End

}
```

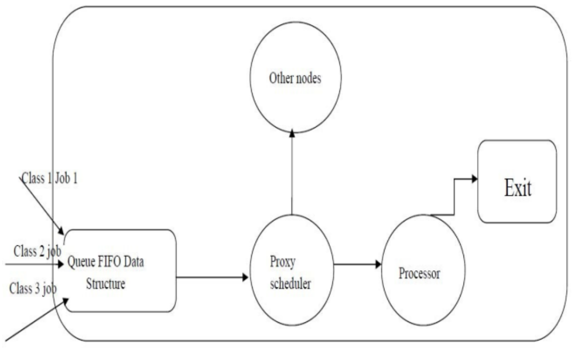


Fig 3 QCHAVR algorithm

(iii) Queue and rate with algorithm via Virtual Routing (QARVR):

```
Procedure for proxy(jobid,portno,ipaddr1)

{

begin
Step1:

Decide the class of job based on jobid with portno

And identify the node ipaddr1.

Step2.

Query the database tables that store the information about
the utilization corresponding to the jobid (class of job)and
corresponding processers ipaddr in which minimum of
(utilization*ql) is there result(arry) ip address are sent to
step 3

Step3.

If the array of nodeips of the the result is one of
the nodeip1(its own ip)

Then allocate the job to node itself and goto step4;

break;

Else(result array.length == 1)

Then allocate the job to resulted node and retun step4;

break;

Else(result arry.length>1)/*That is two or more
nodes having the minimum value */

Randomly allocate any of the processors available
and return step4

break;

Step4:

Cjcount=cjcount+1;
Update the utilization factor based on updated cjcount;

Send this updated value to all other
nodes;

Step5:

Return the ipaddr;

Return;
End

}
```

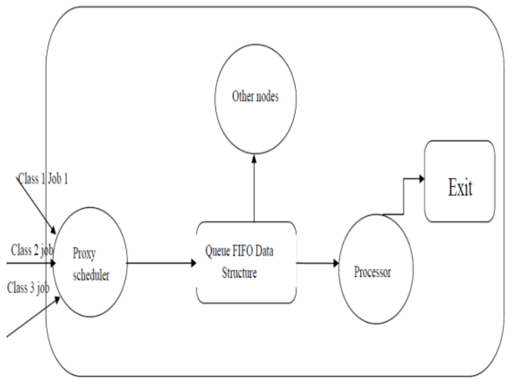


Fig 4 QARVR algorithm

IV. Performance evaluation and Results

We have deployed software which takes ip address as the nodes identity and a scheduler is placed in each node for load balancing the multi class jobs. (Here we have taken three multiclass jobs such as sum, multiplication, division)we analyze the performances of these algorithms with respect to the load on the system. First, in our simulation, we set job arrival rate =0.1 jobs/sec and then increase the value by 0.005 jobs per step. Also, we set time to service each job is 60 seconds and time to execute that job 20 seconds in our experiments .When the average utilization is greater than 0.95, we terminate the simulation. The results of our experiments are shown in Fig5

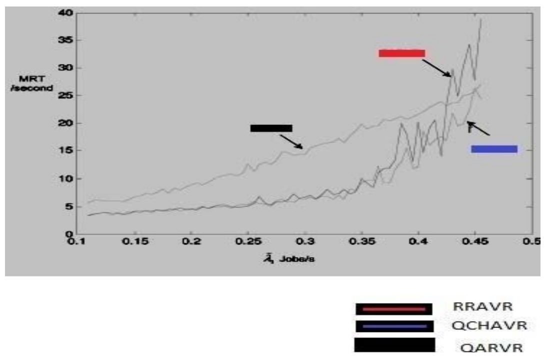


Figure 5 the results of our experiments

From these simulation experiments, we can conclude that, when the system load is light or moderate, the algorithms of RAP policy are preferable to obtain a minimum (or near minimum) mean response time of the jobs. If the load of the system is high, the algorithms of the QAP policy can achieve a better performance. The algorithms of the QRAP policy are suitable in the cases when the system load is fluctuating.

Performance exporter:

This modules generates the graphical visual report for the performances of the each cluster hosts .If there is a variation in the performance of the each cluster host affects and effects will be shown as a graphical visual display of the work load balance. Using this graphical report, the administrator easily knows the overhead of all servers in the network.

Table 1.The mean response times for normal class Jobs for three algorithms (in seconds)

Class of job	rate	queue	Rate and queue
Sum	0.2045	0.2080	0.1789
Multiplication	0.3165	0.3125	0.3224
Division	0.3845	0.3467	0.3120

V. Conclusion and future enhancement

In this paper first we have discussed the problem of load balancing in peer to peer networking systems. Based on the LBVR algorithm we have extended the features of the RRAVR AND QCHAVR algorithms and we have proposed one algorithm for Dynamic Load Balancing among the peers, hence the computation over heads are small [3]. Dynamic Load Balancing scalability determines how its performance improves as hosts are added to the cluster. Scalable performance requires that CPU overhead and latency not grow faster than the number of hosts.

If the number of servers and operations of a scheduler increases, eventually overhead on the proxy to manage the request also increases. So, in future we can also maintain more number of proxies in order to perform more number of tasks. In our paper we are not primarily concerned with the security issues. We are just sending client requests through the Network. But if it is for ATM purpose then it will be prolonged to secure transactions. So, in future we can add the security purposes to this project.

References

[1] Ying Qiao,Gregor V.Bochmann,"A Diffusive load Balancing Scheme for Clustering peer to peer system",2009,15 International Conference on Parallel and Distribute Systems.

[2] Zeng Zeng, and Bharadwaj Veeravalli,"Design and Performance Evaluation of Queue-and-Rate-Adjustment Dynamic Load Balancing Policies for Distributed Networks" IEEE TRANS. ON COMPUTERS, VOL. 55, NO. 11, NOVEMBER 2006.

[3] Z. Zeng and V. Bharadwaj, "A Static Load Balancing Algorithm via Virtual Routing," Proc. Conf. Parallel and Distributed Computing and Systems (PDCS '03), Nov. 2003.

[4] j. li and h. kameda, "load balancing problems for multiclass jobs in distributed/parallel computer systems," ieee trans. computers, vol. 47, no. 3, pp. 322-332.