Dynamic Load Balancing In Distributed Environment Using Rate Adjustment Policies

Jeganathan J.[#1] **Jaiganesh M .**[#2] **Muneeshwari P**[#3]
Department of Information Technology[1, 2, 3]
PSNA College of Engineering and Technology, Dindigul, Tamilnadu, India.
jeganathanjayaram@gmail.com[1]*, jaidevlingam@gmail.com*[2] *radhamunishapcse@gmail.com*[3]

***Abstract* - In today's world of distributed systems has raised a lot concerns about the efficiency. The aim of researchers in this field is to develop new dynamic load balancing techniques that could be applied on the real-time that is more efficient and dynamic than the existing systems. The real time problems of network congestion, centralized point of failure and fault tolerance still remain an open research topic and they are not fully eliminated. My work aims at giving firm solutions for these issues. This work proposes solutions that are more visible because here every implementation is done in real time using several clients and server machines. The clients send the requests to the server based on the number of jobs currently processed by each server and those jobs which are waiting. The decision is taken dynamically to send jobs to the server with least nodes. More dynamism is further incorporated by using parameters like arrival rate and mean service time which are calculated dynamically by the centralized machine using the data received from the server side so that server side over head is avoided. Further, the work portrait's a peer to peer system. The request is sent from the client to the server. Here our main aim is to provide fault tolerance in an intelligent way. Further an efficient scheduler was designed that avoids starvation of low priority jobs.**

Key Words—Distributed systems, Load Balancing, fault tolerance, Scheduler.

.

1. Introduction

The major challenge associated with web technology is the unpredictable and uneven nature of traffic on the Internet[3], [17]. This is particularly difficult for small businesses that have to carefully balance their investment into infrastructure for handling customer requests. On one hand they need a sufficiently powerful infrastructure to handle most of the traffic. At the same time they do not want to over-invest by acquiring hardware that mostly remain idle as both the cost of acquisition and the cost of ownership, in terms of maintenance and management, need to be considered. Thus the main perspective of my project is to show improvements from existing systems and perform optimization in it. The main objective here is to propose a new solution with out any new additional hardware, which increases the cost, poses a great threat to small businesses. The benefit we obtain here is going to be less cost but good optimization. So what improvements can be done with the existing resources is going to be a major issue here.

The clients send the requests to the server based on the number of jobs currently processed by each server and those jobs which are waiting. The decision is taken dynamically to send jobs to the server with least nodes. More dynamism is further incorporated by using parameters like arrival rate and mean service time which are calculated dynamically by the centralized machine using the data received from the server side so that server side over head is avoided. Existing dynamic load balancing techniques have been studied and altered. Initially the real time distributed environment has been created. More parameters are calculated dynamically and the solution emphasizes dynamism. The problem of how often load balancing should be done in dynamic state to avoid performance overhead. Dynamic state load balancing issues are taken in to concern. No server memory is used to queue the requests as the waiting requests are stored on the originating clients in a distributed fashion. **.**

As addressed earlier the existing systems suffer from some serious problems that are still open for research. Existing load balancing methods check the average idle-time of the workstations periodically. But in these methods load balancing cannot be performed until the end of a period even if load imbalance has occurred in the middle of the period. Issues on the shuttling jobs and preemptive versus non preemptive transfer still exist in real time. So, finding solution for these real time issues like time optimization, central point of failure, network congestion due to messages, fault tolerance etc in existing system to achieve optimization will be the major task.

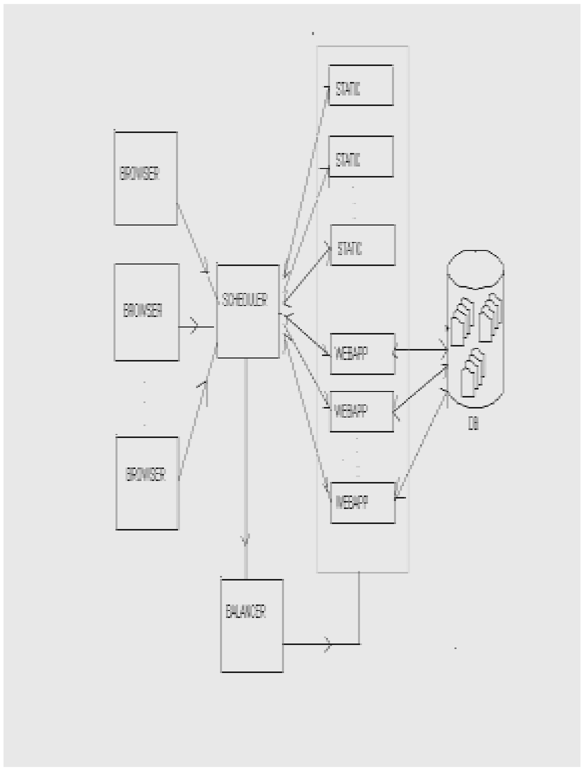


Figure 1. Normal operation of the Algorithm

2. Literature Survey

2.1. Byzantine Fault Tolerance

Dynamic load balancing on the workloads of the clustered workstations has emerged as a powerful solution for overcoming load imbalance[1], [4]. In order to detect such imbalances, some load balancing methods check the average idle-time of the workstations periodically. But in these methods load balancing cannot be performed until the end of a period even if load imbalance has occurred in the middle of the period. More over load balancing when done very often also leads to severe performance overhead. We also try to prevent it by predictions done on collected histories like at which time of day is more requests coming to the server and will that lead to imbalance.

The new method decides a proper time to perform load balancing and does perform the balancing right after the detection of the load imbalance. We also show that a static load balancing method with a long period is suitable if the workstations have to deal with the jobs having unpredictable arrival times and relatively short execution times. Here the load balancing is done randomly and also done based on previous history collected through which we can predict at which time there will be heavy load and how often we should perform load balancing.

Here we check the time the server is idle and then if it is greater than the threshold level LB is performed. Moreover we also randomly perform check on imbalance condition with in the given time period and we adjust the time period dynamically if the load imbalance has been detected and this is also recorded in history. Another major issue that arises here is if the average idle time is a little bit smaller than the threshold value we call this as minor imbalance and we have to ignore this. If we avoid this frequent load balancing can be avoided. For this we can wait for a delta time delay after an imbalance has been detected to see weather the server becomes active again.

The environment visible phenomenon of this project is the client machines, central machine and several server machines. The jobs client send to server can also be seen as only we are going to give those to the servers. So the client side can feel them. The jobs can be high intensity jobs like image files or low ones like text documents. Several of these jobs will be given as input and checked. Moreover since this is going to be implemented in a dynamic environment a lot of testing can be done for various inputs and varying load conditions. The centralized machine we use knows about the server machines their configuration and capacity based on the history it maintains. These details are only known to this and hidden from the client side. The decision the load balancer takes purely depends on the information collected from the server side such as calculation of the idle time, its efficiency, etc…and these are purely system visible and environment hidden. The load balancer, centralized priority schedulers are internal identifiers of the system and they need not be mentioned in the development specification. There is going to be replications among them also. There’s going to be a distributed architecture. The protocols used are mainly TCP, IP, HTTP and FTP.

2.2. ELISA: Estimated Load Information Scheduling Algorithm

From the estimated queue lengths of the nodes in its neighboring nodes and the accurate knowledge of its own queue length, each node computes the average load on itself and its neighboring nodes. Nodes in the neighboring set whose estimated queue length is less than the estimated average queue length by more than a threshold theta form an active set. The node under consideration transfers jobs to the nodes in the active set until its queue length is not greater than theta and more than the estimated average queue length. The value of theta, which is predefined, is a sensitive parameter and it is of importance to the performance of ELISA. Here, the threshold theta is fixed in such a way that the average response time of the system is a minimum.

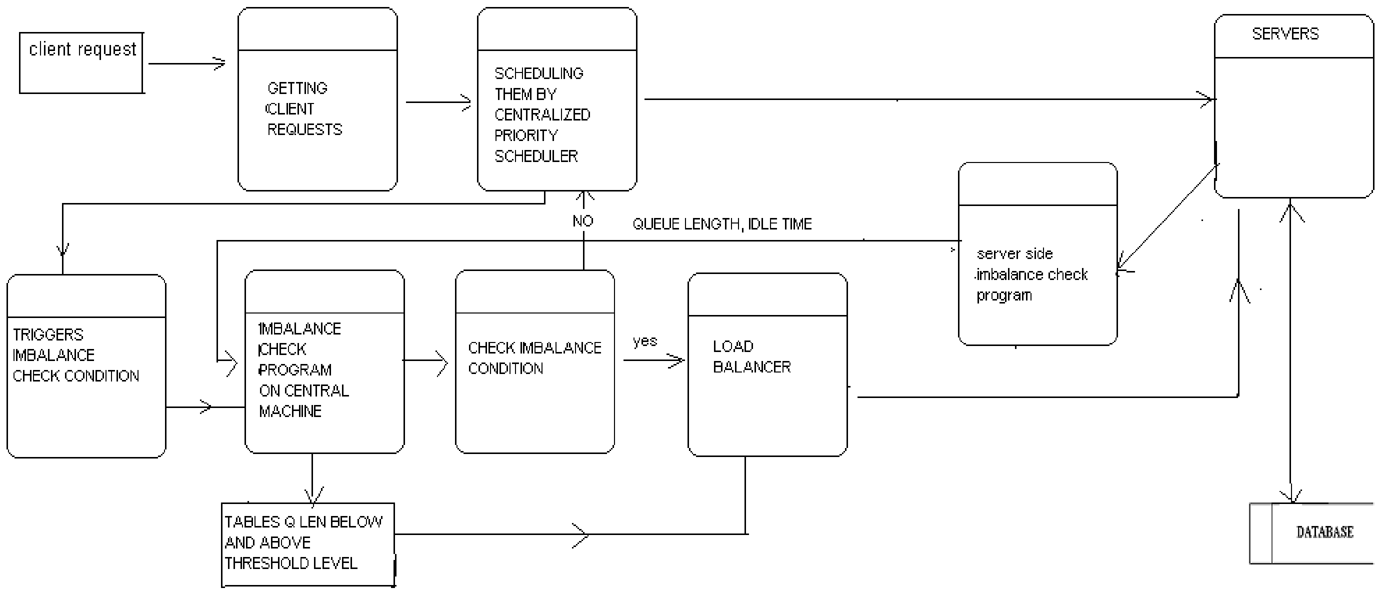
When a job moves from one CPU to another, when it is balanced consider a case where before it reaches the target system it also gets imbalanced and becomes idle. Now, again the job has to be returned back to the priority scheduler and then re assigned. This shuttling time causes a lot of starvation for that process since again it will be put only in back of the queue it has to wait get processed. So to avoid this first we identify such kind of shuttled jobs and then we place it in a separate high priority queue so that it gets processed quickly. Thus we solve discrimination of these shuttled jobs.

The request is sent from the client to the server. Initially a small sample file is downloaded from each sever. The main file download is resumed from the server with the least response time. If the server from which the current down load takes place is cut off then the download process is resumed from the next server with the shortest response time. Thus our main aim is to provide fault tolerance in a intelligent way. This is implemented in a real time environment with variable buffer size. Further, when designing the centralized priority scheduler more emphasis is laid on avoiding the starvation of the low priority jobs. Load balancing is done by rescheduling the jobs to the queues that are free. Synchronization is achieved and the parallel down load of the jobs takes place where periodically chance is given for the low priority jobs avoiding starvation.

3. System Architecture

The scheduler is immediately placed before the queue. When a job arrives at a node, the scheduler first arranges the job based on its priority and then decides where the job should be sent to the ready queue of the server. Here we use a centralized priority scheduler. Once the job has entered the queue, it would be processed by the processor and will not be transferred to other nodes. The concept of load balancing is brought in to picture only when an imbalance condition is detected. Until there is no imbalance the jobs are ordered using the priority scheduler and sent to the corresponding CPU's. When the imbalance is detected rescheduling is done. Using this technique, the static algorithms can attempt to control the job-processing rate on each node in the system and eventually obtain an optimal (or near optimal) solution for load balancing. On rescheduling also we focus only on transferring the non-preemptive jobs for transferring to avoid the overheads. The performance of this system is more simple and efficient because we do load balancing only when an imbalance is detected. Until that balancer is not brought to picture.

If there are many application servers sharing one database server the database server become congested. Usually, write operations are more costly than reading operations as the results of reading operations sometimes are cached and the corresponding disk access is eliminated. The strategy proposed here helps prevent a database server from becoming overloaded as the web-servers control the amount of queries submitted to the database server. The strategy can be used with load balancers, but it falls in the category of techniques that do not require additional hardware. . No server memory is used to queue the requests as the waiting requests are stored on the originating clients in a distributed fashion. Thus the main objective of my project is to show improvements from existing systems and perform optimization in it. The advantage of the proposed method is that it can be used to serve a high number of requests that otherwise might result in server crashes. Here we check the time the server is idle and then if it is greater than the threshold level LB is performed.


client request
GETTING CLIENT REQUESTS
SCHEDULING THEM BY CENTRALIZED PRIORITY SCHEDULER
SERVERS
NO
QUEUE LENGTH, IDLE TIME
server side imbalance check program
TRIGGERS IMBALANCE CHECK CONDITION
IMBALANCE CHECK PROGRAM ON CENTRAL MACHINE
CHECK IMBALANCE CONDITION
yes
LOAD BALANCER
TABLES Q LEN BELOW AND ABOVE THRESHOLD LEVEL
DATABASE

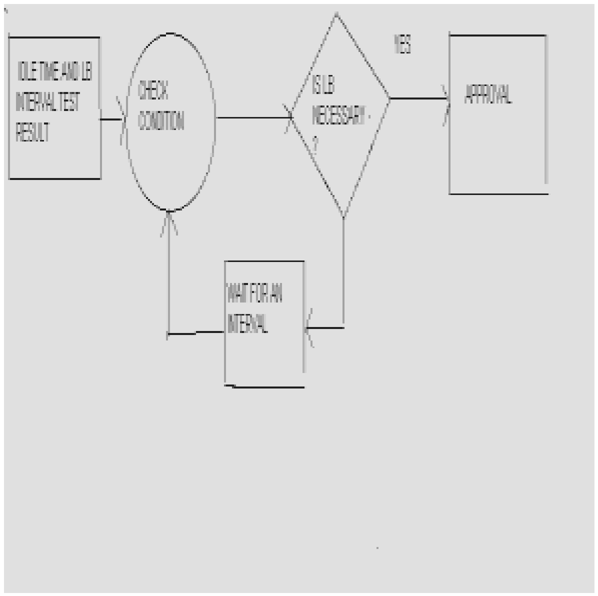
3.2 A Policy for Dynamic Load Balancing

In this section, we modify the centralized one-shot LB strategy to a distributed, adaptive setting and use it to develop a sender-initiated DLB policy. The distributed one-shot LB policy is different from the centralized one-shot LB policy.

Each time an external load arrives at a node, the node seeks an optimal one-shot LB action that minimizes the load-completion time of the entire system, based on its present load, its knowledge of the loads of other nodes, and its knowledge of the system parameters at that time. For clarity, we use the term external load to represent the loads submitted to the system from some external source and not the loads transferred from other nodes due to LB. We will assume external load arrivals of random sizes. Each time an external load is assumed to arrive randomly at any of the nodes, independently of the arrivals of other external loads to it and other nodes.

Consider a system of n distributed nodes with a given initial load and assume that external loads arrive randomly thereafter. We assume that nodes communicate with each other at so-called “sync instants” on a regular basis. Upon the arrival of each batch of external loads, the receiving node and only the receiving node prompts itself to execute an optimal distributed one-shot LB. Namely, it finds the optimal LB instant and gain and executes an LB action accordingly. Since load balancing is performed locally at the external-load-receiving node, say, node j, the policy depends only on its knowledge state vector ij, rather than the system knowledge state I.

Further, considering the periodic sync-exchanges between nodes, each node in the system is continually assumed to be informed of the states of other nodes.



In practice, external loads of different size (possibly corresponding to different applications) arrive at a distributed-computing system randomly in time and node space. Clearly, scheduling has to be done repeatedly to maintain load balance in the system. Centralized LB schemes [10], [11] store global information at one location and a designated processor initiates LB cycles. The drawback of this scheme is that the LB is paralyzed if the particular node that controls LB fails. Such centralized schemes also require synchronization among nodes. In contrast, in a distributed LB scheme, every node executes balancing autonomously.

Moreover, the LB policy can be static or dynamic [2], [12]. In a static LB policy, the scheduling decisions are predetermined, while, in a dynamic load-balancing (DLB) policy, the scheduling decisions are made at runtime. Thus, a DLB policy can be made adaptive to changes in system parameters, such as the traffic in the channel and the unknown characteristics of the incoming loads. Additionally, DLB can be performed based on either local information (pertaining to neighboring nodes) [13], [14] or global information, where complete knowledge of the entire distributed system is needed before an LB action is executed.

Due to the emergence of heterogeneous computing systems over WLAN'S or the Internet, there is presently a need for distributed DLB policies designed by considering the randomness in delays and processing speeds of the nodes. To date, a robust policy suited to delay-infested distributed systems is not available, to the best of our knowledge [3]. In this paper, we propose a sender-initiated distributed DLB policy where each node autonomously executes LB at every external load arrival at that node. The DLB policy utilizes the optimal one-shot LB strategy each time an LB episode is conducted, and it does not require synchronization among nodes. Every time an external load arrives at a node, only the receiver node executes a locally optimal one-shot-LB action, which aims to minimize the average overall completion time. This requires the generalization of the regeneration-theory-based queuing model for the centralized one-shot LB [9]. Furthermore, every LB action utilizes current system information that is updated during runtime. Therefore, the DLB policy adapts to the dynamic environment of the distributed system.
This paper is organized as follows: Section 2 contains the general description of the LB model in a delay-limited environment. In Section 3, we present the regeneration based stochastic analysis of the optimal multi node one-shot LB policy and develop the proposed DLB policy.

4. Scheduling Algorithm

Scheduling techniques have also been used in the multiple-class domain such as Output-Controlled Round Robin (OCRR) [5], Priority Queuing (PQ) [6], Weighted Round Robin (WRR) [12], [24], PQWRR [13], DRR+ [9], and DRR++. DRR++ suffers from head of line blocking when scheduling more than one higher priority stream. PQ is unfair to lower-priority traffic. PQWRR is unfair to AF and BE by using PQ for the EF traffic and WRR for the AF and BE traffic. Finally, OCRR, DRR+, and DRR++ are originally designed for two classes only. The common approach to support Diff Server traffic is to save all same-class packets from different sources in a shared FCFS (First Come First Served) buffer [10], [13], [14]. However, it is difficult to control the service order of packets from different sources because a busty source in a class may cause a higher delay and even loss for well behaved streams within that class.
In view of various deficiencies discussed above (namely, supporting only one or two classes of traffic, unfairness, non smooth scheduling (busty transmission from same stream), higher service time, and higher startup latency and jitter), we extend our OCRR [5] to support multi class traffic and provide extensive performance analysis. Our objective is to fairly schedule IP packets in the Diff Server

We consider a backbone packet switching network with a number of core routers in the Diff Serv domain. Each core router is physically connected to R immediate upstream routers at its input ports (each called a “source router”). We assume R is fixed for a given interconnect topology. An OCGRR scheduler resides at each output port of a core (or an edge) router and schedules traffic into a (output) router that is immediately downstream. Each core router may request edge routers to adjust their arrival rates to satisfy the QoS requirements, in addition to adjusting their scheduling parameters to achieve a desired QoS Q represents a stream, and X bits are the total transmitted bits in a frame. The En queue process adds a new packet of any stream in its relevant buffer. It then appends the stream reference to the relevant Active List provided that the stream not in the Active List becomes backlogged (i.e., had a positive grant but was empty).

The scheduling for each class is divided into two parts:

1) In the De queue Init process, the grant of each stream inside class J is incremented by some quantum computed based on the frame beginning time (see Section 3.3). Then, if a non backlogged stream becomes backlogged, its reference is appended to Active List J; and

2) in the De queue Process, packet scheduling is performed. There are two scheduling processes: one for the classes that use a dedicated buffer per stream in a class, and the other for the classes that use one shared buffer for all streams in a class. The former case has the following steps:

1. Schedule Domain Determination: Define a schedule domain J to include the backlogged streams with in class J before processing this class. This domain is handled with the Round Len parameter. The backlogged streams from the beginning of Active List J and from this domain can only transmit traffic during the current frame period, and the serving of newly backlogged streams during this period will be postponed to the next frame.

2. Traffic Transmission: When scheduling class J traffic within the frame, only the backlogged streams in class J can transmit traffic. Whenever a stream in this class becomes non backlogged (defined earlier), the stream is removed from Active List J, and the schedule domain J becomes smaller. Thus, in the next round, a small number of streams will participate. In OCGRR, when a stream becomes non backlogged, its grant parameters remain unchanged.

The scheduler visits the backlogged streams one by one. In each visit, only one packet is transmitted from a stream and then the next stream is visited. This continues until the last backlogged stream in the schedule domain J is visited.

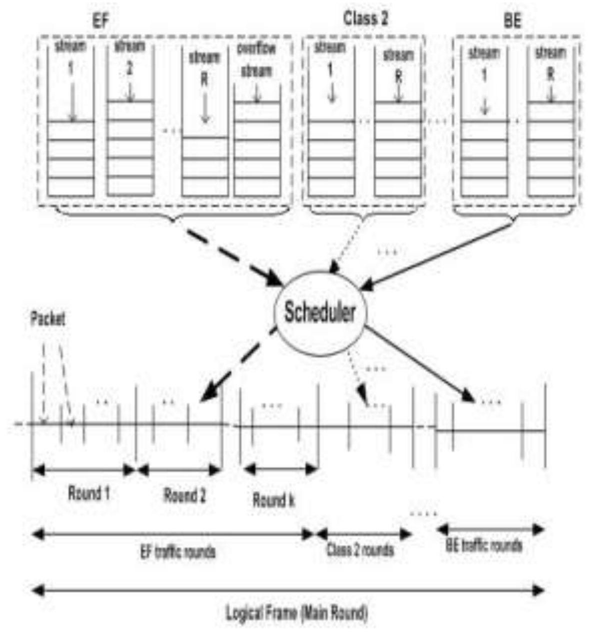
Then, the scheduler starts visiting the backlogged streams from the head of Active List J. We call this scheme Multiple Round Robin (MRR) transmission because each stream may transmit its packets in multiple rounds, but only one packet in each round. Scheduling for class J is stopped whenever there is no backlogged stream left in Active List J or the total transmitted traffic exceeds _. The latter condition is evaluated at the end of each packet transmission.

When scheduling a packet from a stream, OCGRR first schedules the packet with any size and then updates both grant values of the stream with the transmitted packet size. This update may lead to a negative grant if the size of the transmitted packet is greater than the stream’s grant. When the stream’s grant is negative, the stream becomes non backlogged.

If the frame ends right before processing stream i in class J, OCGRR flags this stream in the Last List Ptr J parameter related to class J. At a future frame when it is the turn of class J to be processed, the scheduler would start processing class J from the stream referenced by Last List Ptr J, but not from the beginning of the streams referenced by Active List J. This is handled by “ListPtr GetStartingStreamRefInRound”. This process ensures fairness for stream i. It is also in the direction of reducing the inter transmission time from the same stream.

In OCGRR, packets from the head of streams have almost the same chance of being transmitted under MRR because coherent transmission of packets from the same stream is reduced. Moreover, a packet in a newly backlogged stream encounters a lower latency in OCGRR.

In addition, jitter among packets of the same stream is reduced in OCGRR.



5 Implementation

Our performance evaluation is carried out on a test bed consisting of Windows servers connected by a 100Mbps Ethernet. Each server is equipped with a single Pentium D 2.8GHz processors and 1GB memory.

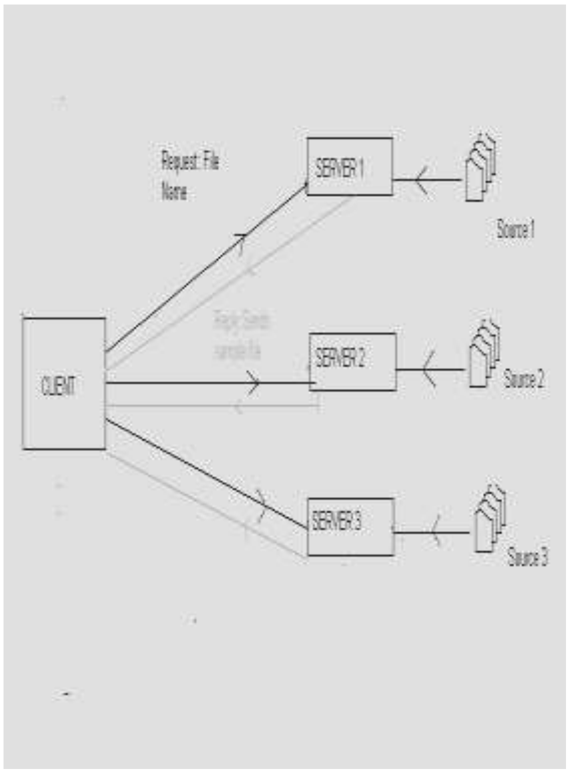
Initial lab set up is done with clients and servers connected and communication between them is shown. The clients send the request to servers based on the number of jobs waiting to get processed in each server currently. The server with the least number of jobs is given the request. Dynamically the decisions are taken based on the database table values. Three tables are maintained of which load table contains the load present at each server currently. Based on this table only the loads are distributed by the clients. The account table contains the account number and the amount details of the customers.
Further optimizations were done by implementing a middle ware which makes decisions based up on mean arrival rate, processing time, idle time, etc.

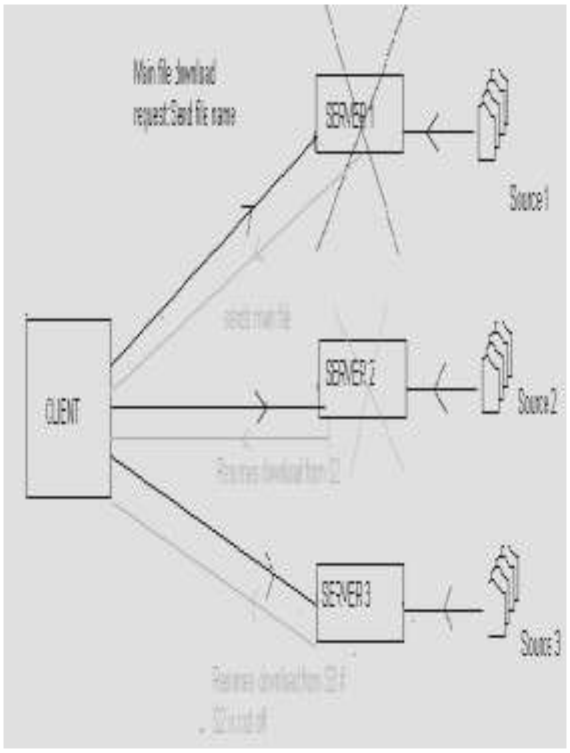
Initial download of a sample file from the server's .Server receives the file name and streams the packets to the network. Client reads and writes the packet in to destination The time taken to download that file from each server is calculated and they are sorted. The server with the shortest time value is connected with the client and through it only the main file to be obtained is downloaded. Now, if this server is cutoff in the middle of downloading process that packet with which the download was cutoff is marked and the next server with the shortest response is contacted and again the download is resumed. This scenario is implemented with three servers using the concept of socket in java. The entire file is streamed as equal size packets in to the network using the buffers.
For accomplishing these tasks we pass messages between the client and the servers.
When the client gets the "Initial connection message" it does the initial normal streaming from beginning of the packet. I.e. it's the first connection.

When the client gets the "Reset connection message" it does the streaming from the cutoff packet. I.e. it's the Reset connection. So, through this message we also send further information such as the cut off packet number and the remaining file size, etc.

The file reading at the server side is done from three different sources namely source1, source2 and source3. Same copies of file to be downloaded are maintained at these locations. I.e. server 1 reads the required file from source1, server 2 reads the required file from source2 and so on .I.e. they are all not read from the same location.

Similarly, the outputs are also stored at different locations namely downloaded1, 2, and 3 to show that download has been resumed from the cutoff point.





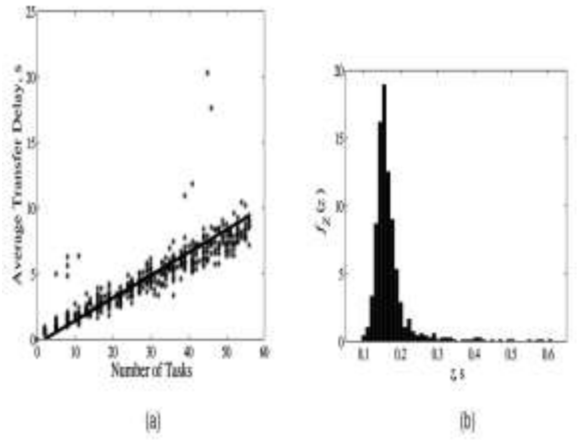


Fig. 2. (a) Mean delay as a function of the number of tasks transferred between nodes. The stars are the actual realizations from the experiments.
(b) Empirical pdf of the transfer delay per task on the Internet under a normal work-day

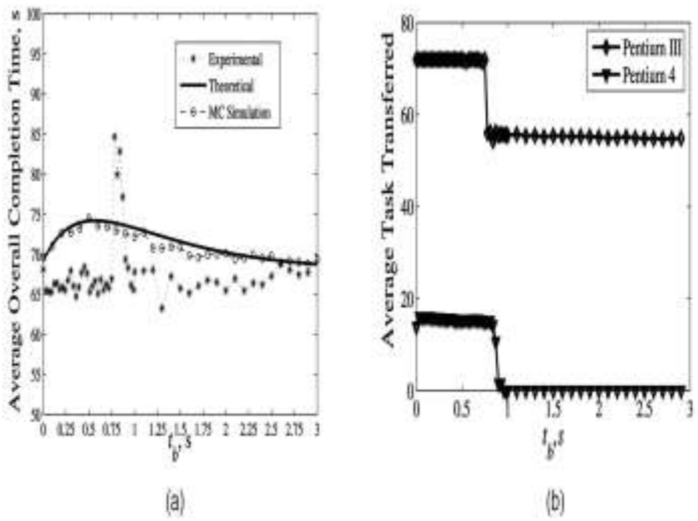


Fig.3.(a) The AOCT as a function of LB instants for the experiments over the Internet. The LB gain was fixed at 1.
(b) The amount of load transferred between nodes at different LB instants.

6 Conclusion

In this project I implemented the initial architectural lab setup. Our method is based on the dynamic global optimization scheme. Initially the load was split up based on the server node with the least number of jobs to be processed. More optimizations were done to this considering parameters like arrival rate and mean service time. Basic monitoring agent was developed which makes these decisions based on dynamic results obtained from calculations of the above said parameters periodically.

This work is further extended to design efficient centralized priority scheduler. And then I am going to implement intelligent agents. More parameters will be brought to picture to show dynamism. Demo will be shown like how prevention will be done when all the servers get hanged at the same time. More concentration will be laid on avoidance of the congestion, fault tolerance and centralized point of failure. More real time problems will be studied and more concentration will be laid on the performance related issues.

References

[1] Ali M. Alakeel, “A Guide to Dynamic Load Balancing in Distributed Computer Systems,” in IJCSNS International Journal of Computer Science and Network Security,June 2010, vol10,p-6

[2] Jagdish Chandra Patni M.S. Aswal Aman Aswal Paras Rastogi "A Dynamic and Optimal Approach of Load Balancing in Heterogeneous Grid Computing Environment" in Emerging ICT for Bridging the Future vol. 2 pp. 447-455 2015 Springer International Publishing.

[3] S.E. Dashti, A. masoud Rahmani, "A New Scheduling Method for Workflows on Cloud Computing", *International Journal of Advanced Research in Computer Science*, vol. 6, no. 6, 2015.

[4] Neeraj Rathore Inderveer Chana "Variable Threshold-Based Hierarchical Load Balancing Technique in Grid" <em>Springer Engineering with Computer</em> 2014.

[5] Rohit Saxena Ankur Kumar Anuj Kumar Shailesh Saxena "AHSWDG: An Ant Based Heuristic Approach to Scheduling and Workload Distribution in Computational Grids" <em>IEEE International Conference on Computational Intelligence & Communication Technology</em> pp. 569-574 2015.

[6] S. Blake et al., An Architecture for Differentiated Services, RFC 2475,Dec. 1998.

[7] V. Jacobson, K. Nichols, and K. Poduri, An Expedited Forwarding PHB, RFC 2598, June 1999.

[8] J. Heinanen et al., Assured Forwarding PHB Group, RFC 2597, June 1999.

[9] S.J. Golestani, “A Self-Clocked Fair Queueing Scheme for Broadband Applications,” Proc. IEEE Infocom ’94, pp. 636-646, June 1994.

[10] A.G.P. Rahbar and O. Yang, “The Output-Controlled Round Robin Scheduling in Differentiated Services Edge Switches,” Proc.
IEEE BROADNETS ’05, Oct. 2005.

[11] D. Bertsekas and R. Gallager, Data Networks. Prentice Hall, 1992.

[12] S. Kanhere, A. Parekh, and H. Sethu, “Fair and Efficient Packet Scheduling Using Elastic Round Robin,” IEEE Trans. Parallel and
Distributed Systems, vol. 13, no. 3, pp. 324-336, Mar. 2002.

[13] Y. Ito, S. Tasaka, and Y. Ishibashi, “Variably Weighted Round Robin Queueing for Core IP Routers,” Proc. IEEE Int’l Performance,
Computing, and Comm. Conf. (IPCCC ’02), Apr. 2002.

[14] M. Shreedhar and G. Varghese, “Efficient Fair Queuing Using Deficit Round Robin,” IEEE/ACM Trans. Networking, vol. 4, no. 3,
June 1996.

[15] Y. Jiang, C.-K. Tham, and C.-C. Ko, “A Probabilistic Priority Scheduling Discipline for Multi-Service Networks,” Elsevier Computer Comm., vol. 25, no. 13, pp. 1243-1254, 2002.

[16] M. MacGregor and W. Shi, “Deficits for Bursty Latency-Critical Flows: DRR++,” Proc. IEEE Eighth Int’l Conf. Networks (ICON ’00),
Sept. 2000.

[17] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, “Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip,” IEEE J. Selected Areas in Comm., vol. 9, no. 8, pp. 1265-1279, Oct. 1991.

[19] J. Mao, W.M. Moh, and B. Wei, “PQWRR Scheduling Algorithm in Supporting of DiffServ,” Proc. IEEE Int’l Conf. Comm. (ICC ’01), vol. 3, June 2001.

[20] L. Ji, T.N. Arvanitis, and S.I. Woolley, “Fair Weighted Round Robin Scheduling Scheme for Diffserv Network,” IEE Electronic
Letters, vol. 39, no. 3, Feb. 2003.

[21] C. Guo, “SRR: An O(1) Time-Complexity Packet Scheduler for Flows in Multiservice Packet Networks,” IEEE/ACM Trans. Networking, vol. 12, no. 6, Dec. 2004.

[22] C. Zhang and M. MacGregor, “Scheduling Latency-Critical Traffic: A Measurement Study of DRR+ and DRR++,” Proc. IEEE High Performance Switching and Routing (HPSR), June 2002.

[23] S.S. Kanhere and H. Sethu, “Fair, Efficient and Low-Latency Packet Scheduling Using Nested Deficit Round Robin,” Proc. IEEE High Performance Switching and Routing (HPSR), May 2001.

[24]http://www.opnet.com/products/modeler/home.html, 2007.

[25] V. Paxson and S. Floyd, “Wide Area Traffic: The Failure of Poisson Modeling,” IEEE/ACM Trans. Networking, vol. 3, no. 3, pp. 226-244, 1995.

[26] ttp://www.caida.org/analysis/AIX/plen_hist, 2007.

[27] R.R.-F. Liao and A.T. Campbell, “Dynamic Core Provisioning for Quantitative Differentiated Services,” IEEE/ACM Trans. Networking,
vol. 12, no. 3, pp. 429-442, June 2004.
[28] J. Yang, “A Proportional Congestion Control Solution for Real-Time Traffic,” MS thesis, Carleton Univ., Ottawa, ON, Canada,
2005.
[29] F. Agharebparast and V.C.M. Leung, “A New Traffic Rate Estimation and Monitoring Algorithm for the QoS-Enabled
Internet,” Proc. IEEE Globecom ’03, Dec. 2003.
[30] H.M. Chaskar and U. Madhow, “Fair Scheduling with TunableLatency: A Round-Robin Approach,” IEEE/ACM Trans. Networking,
vol. 11, no. 4, pp. 592-601, Aug. 2003.
[31] A. Habib, S. Fahmy, and B. Bhargava, “Monitoring and ControllingQoS Network Domains,” ACM/Wiley Int’l J. Network Management,vol. 15, no. 1, pp. 11-29, Jan. 2005.