Situation based Load Balancer for Distributed Computing Systems

P Beaulah Soundarabai
Dept. Of Computer Science
Christ University
Hosur Main Road

Thriveni J,
K R Venugopal,
University Visvesvaraya College of Engineering,
Bangalore University,
Bangalore

L M Patnaik
Honorary Professor,
Indian Institute of Science,
Bangalore.

ABSTRACT
In this paper, one of the major objectives of distributed systems is performance. Load balancing is a concept used in computer networks to distribute the workload across the replicated system resources. Load balancing is the key driving factor to enhance the performance of the system. The requests of various clients are redirected to the available servers considering its existing workload. The main aim of the load balancing algorithms is to equally distribute the load the available servers. The algorithms also need to consider the processing power of each server. There are many load balancing algorithms available. Dynamism is also used in these algorithms to throw the task to the next eligible server. Simple load balancers use random choice and round robin algorithms. But the system use only one algorithm for the load balancing such as round robin or weighted or priority etc. But each algorithm would be efficient in one aspect and might be inefficient otherwise. In our paper, we try to use few algorithms and invoke them during a particular situation when they are efficient. Simulation results show that our load balancer significantly improves the average and total response time of client tasks and thus increases the performance of the overall system.

Keywords
Load Balancing, FCFS, Round Robin, Optimized weight, Server Process Reporter, File Load Controller, Thinker, and Distributed Systems..

1. INTRODUCTION
A distributed system is a connection of many individual machines which together form a single system image. The software is tightly coupled to make the virtual unique system. Distributed systems aims at bringing in all the system resources such as memory, processing power, special devices, network etc., together and allow the independent machines to share them and use them effectively through which the overall system gets the better performance. The system also achieves a fault-tolerance by replication of the resources and the system variants.

A distributed system can also be envisioned as a collection of computing and communication systems and resources that are shared by many users in parallel. Because of the replicated resources especially like the file servers and the processors, there may be queues to utilize them in one copy of the resource and at the very same time, many more copies of the same resource are idle. The overall performance of the system goes down when the demand for the processing power goes up.

Motivation: Load Balancing is driving force of distributed computing systems which brings in the advantages of sharing of resources. The key factor here is, all the copies of the resources should be utilized equally without leaving few free and few overloaded. The system should keep a watch on the servers' load. There are many load balancing algorithms available with their own advantages. We have tried to collaborate the algorithms making the system get all their advantages .

Contribution: In this paper we have proposed a situation-based-load balancer which works along with thinker module. Our load balancer invokes the thinker to decide which among the existing load balancing algorithms would be the best in that particular time, and makes that algorithm to execute. The thinker module keeps deciding the load balancing algorithm at every time interval, trying to yield the best performance while balancing the load.

Organization: The remainder of the paper is organized as follows: Literature review is available in Section II; System model is given in Section III; Section IV discusses the Problem Definition and the Algorithm for Situation Based Load Balancer; Performance Evaluation details are available in Section V and the conclusions are present in Section VI.

2. LITERATURE SURVEY
Considerable amount of research is done on load balancing with various factors of the system. The authors have focused their load balancing on P2P systems with high concentration on middleware-level based load balancing strategies over a large dynamic peer-peer network. The studies have been done on both structured and unstructured P2P systems [1-3]. The load balancing algorithm based on the network utilization has been developed by Saito et al., [4] which controls the traffic flow in the network and hence minimizes the over utilized and underutilized network links. The various system level resources like the memory, processor power and the system files have been considered for the operating system level load balancing by Stoica et al., [5] for P2P network. The load balancing taxonomy focuses on various factors such as centralized and decentralized, client oriented and server oriented, heuristic, deterministic and non-deterministic algorithms which have been explained in detail by Karthik [6-7]. The algorithms that are based on hash tables with the homogenous objects, virtual servers with heterogeneous objects are also discussed and a comparison has been carried out [8]. Triantafillou et al. [9] have developed load balancing algorithms for P2P systems which distributes global meta-data contents over over a two-level hierarchy unstructured system.

Barazamdej et al., [10] develped two methods for the load balancing in the distributed systems which were based upon hierarchical structure. They have develped the algorithms dynamically to which work on the current load weight. They also have improved the round robin algorithm by having the high priority for the servers which has lesser load state. Since the centralised approach is used, the algorithms suffer the traditional problems of single source of failure and the bottle neck delays. Yang Jiao et al., [11][12] have presented the problems pertaining to the load balancing algorithms. They have mainly focused on unscientific and inaccurate algorithm, imperfect and impractical load-balancing system design. They also have proposed and implemented web server load balancing algorithm. Grosuand D et al., [13] have designed a load balancing algorithm which is very much dynamic wherein the users and the systems are eligible to manipulate it according to their interest. Their study was based on the techniques from mechanism design theory. They have also proposed a fair load balancing protocol.

State-of-the-art core routers provide terabit and petabit switching capacity with the help of multipath switching techniques. Lei Shi et al., [14] discussed the limitation of flow based hashing algorithms and proposed flow slice algorithm that cuts off every flow into slices through which balancing the load with fine granularity. Using trace driven prototype simulation, they have proved their theoretical claim.

3. MODEL

A. Network Model

In this work, the aithors have considered well connected distributed computing network N with *n* number of nodes. The network can be represented as N = *(c,s)*, where *c* is the numbr of client nodes and s is the number of server nodes. Along with these we have few software components working very close with these clients and servers under the leadership of "Situation-Based Load Balancer". Fig. 1. Depicts the system model of our Situation Based Load Balancer and Table I describes the various notations used in our algorithm.

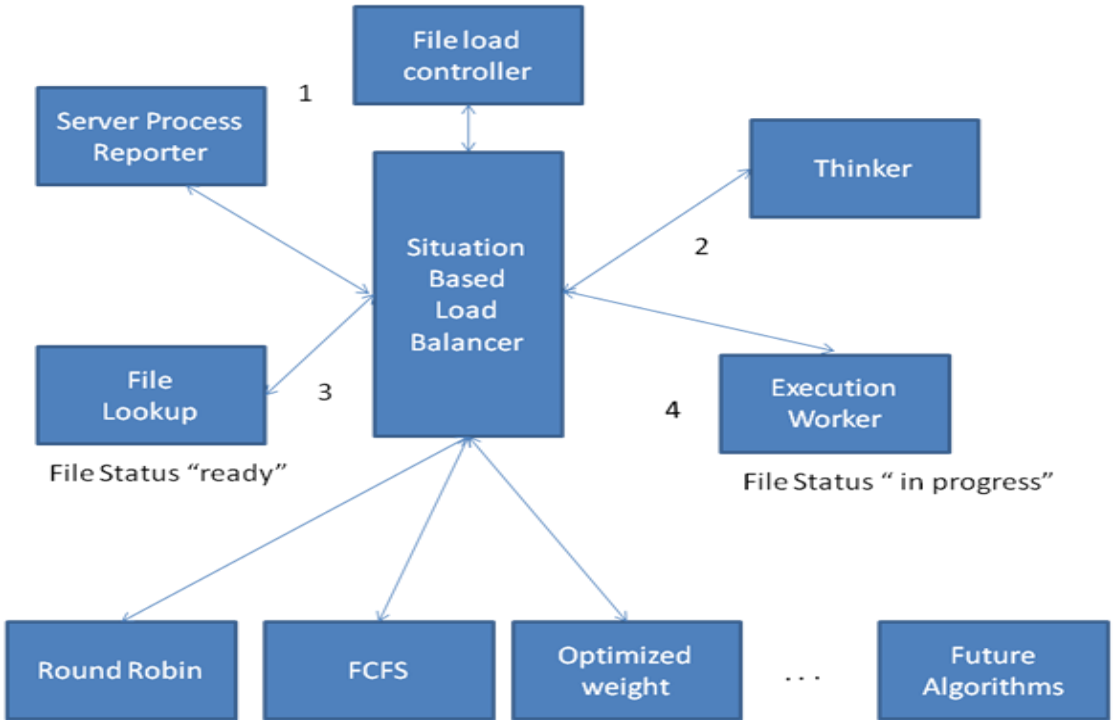

Fig 1: System Model of Situation-Based Load Balancer

B. Situation-Based Load Balancer

The description of all these components is as follows:

File Load Controller (FLC)

As soon as the client job(s) arrives, the role of FLC is to collect all these jobs from various nodes and load them into our Load Balancer. Whenever any client job arrives, its flag becomes true and FLC starts working.

Server Process Reporter (SPR)

SPR is another important module which works along with the FLC. At every time interval, it checks for the number of available servers, their processing power reports the same to the load balancer.

Thinker (Th)

Thinker is the heart of our load balancer which examines the number available processor and the processing power of them and decides the LB algorithm best for the moment and informs the load balancer.

File Lookup (FL)

File lookup component comes to action once after the 'thinker' and marks all the jobs that are to be serviced as "ready"

Execution Workers(EW)

Each EW fetches the jobs which are ready from the SLB and immediately sets them as "in Progress" before servicing them. This is mainly done to ensure that the same job should not fetched by some other EW.

Situation-Based Load Balancer:

This is the centralized component which includes all the above described components and work very close with them to keep on loading the client jobs for execution and invokes the best algorithm for the situation by which the response time of the job is minimized. It decides from the existing three algorithms namely RR, FCFS and OW.

Table 1. Notations used

Symbols	Defintion
LB	Load Balance
RR	Round Robin
FCFS	First Come First Served
OR	Optimized Weight
SLB	Situation-BasedLoad Balancer
FLC	File Load Controller
SPR	Server Process Reported
EW	Exection Workers
Th	Thinker
T	Clock Interval
N(client)	Number of Client Jobs
N(server)	Number of Available Servers

The various situation for them to show their best performance are as follows:

FCFS:

The FCFS works well when there is only one server available at present. The thinker decides to choose FCFS when there is only one server and all the other servers are down. All the "ready" jobs are kept in a single queue as there is only one server available.

RR:

Round Robin algorithm works fine when all the servers are available. The performance of the system goes up due to the availability of all the servers and load is also evenly balanced by this algorithm. If the all servers' processing power are the same and if few servers are down but n(s) >1 then also RR gets selected for invocation.

Optimized Weight: Optimized Weight is a dynamic LB algorithm, which works fine with servers of high processing power. There may be a situation wherein there are few servers down, and the available servers of high processing power. If a server can take up 6 jobs at a time, in a normal situation it would be given a maximum of 3 to 4 jobs instead of 6. This is to make sure that the servers are not overloaded. But during the failure of few servers, they can be overloaded so that the performance of the system is the almost the same even during the time of partial failure of the servers. The thinker decides to choose this algorithm during partial failure of servers provided the available server should be of high processing power.

If we process 100000 processes in one second, then it should be the same for ever even during the partial failures. There would be micro seconds delay may be due to the critical region usage. If we have four servers and even if three servers fail, the last node should be able to take the load of other three servers load by taking up 1.5 times of the existing load.

The big advantage of our model is that, it not only works with the algorithms which are loaded presently. It also gives room for the future algorithms which may be proved good for load balancing. The only requirement would be that we need to recode the thinker module. Any time the new LB algorithms can be included and the old LB algorithms can also be removed from the load balancer.

The present model has the centralized "situation load balancer", which if fails, the load balancing would be a failure. It suffers the problem of single source of failure which can be eliminated by distributing it into n number of components.

4. PROBLEM DEFINITION AND ALGORITHM

A. Problem Definition

Consider a distributed system with many clients and servers. Gven a number of client jobs to be processed by one of the available server from a pool of replicated servers. To decide which server would be feasible, we have Situation Based Load Balancer along with Thinker and its various components then the main objectives of our proposed Load Balancer are as follows:(i) The client jobs have to be serviced with minimum response time.

(ii) Server's computing power has to be used efficiently.

(iii) Load of all the replicated server has to be balanced evenly.

(iv) The overall performance of the system to be improved.

The algorithm of our Situation Based Load Balancer is present in Table II.The main focus of our algorithm is to choose the best LB algorithm at every time interval and through that decreasing the response time of the client jobs and increasing the performance of the overall system.

Thinker executes at every clock interval , decides which algorithm is best, decision is informed to the load balancer and goes back to sleep mode.

Table 2. Algorithm : Sitation Based Load Balancer

Variable: Client Job, Server, Load Balancing algorithms: Round Robin (RR), First Come First Served (FCFS), Optimized Weight (OW) Future Algorithms (FA)

Input:

1. Situation-Based Load Balancer
2. File Load Controller
3. Server Process Reporter
4. File Look up
5. Thinker
6. Execution Worker
7. Load Balancing Algorithms (RR, FCFS, OW, FA)

Output:

- Client jobs are serviced with minimum Response time
- The servers' load is balanced evenly
- The best algorithm among the Load Balancing Algorithm executes at every time interval.

```
if (client job)
set client to "true"
else
set client to "false"
endif
while (client) do
        for (each client) do
        FLC loads the job into SLB
        SPR loads the n(client) and  n(server) into SLB
        endfor
         for (every clock interval)
                if (client)
                        SLB wakesup Th
                        Th decides the Best LB  for this clock interval
                        FL sets each client as "ready"
                        EW sets each client to "in progress"
                        SLB invokes the LB algorithm
```

5. PERFORMANCE EVALUATION

In this section, the performance of our load balancer is evaluated with the individual load balancing algorithms such as FCFS, RR and Optimized Weight(thinker).Simulation set up of our imple mentation is as follows:

File arrival duration	30 minutes
Files Lookup interval	1 minute
Files per Lookup	10 files
Times to process one file	15 Seconds
Total Number of servers	4 Servers
Think Time interval	15 secs
Scenario	1 server fails / 8 mins

We have made 10 files to arrive at every 10 seconds interval.

Fig. 2 Compares the processing time of individual set of jobs at each interval. FCFS takes higher time as it has a single queue. Our model (thinker) has the constant time because it keeps deciding the best algorithm at every time interval. Fig. 3 and Fig. 4 describe the total and average processing time of the three algorithms. Fig. 5 depicts the maximum response time taken by each algorithm and our model has the better response time.

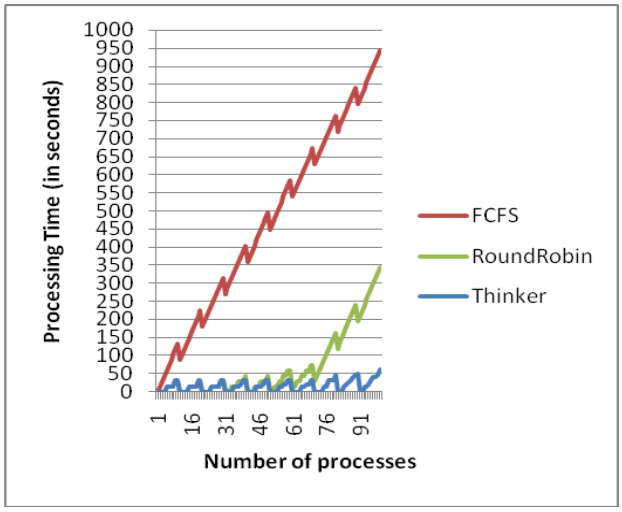

Fig 2: Individual Processing Time Comparison

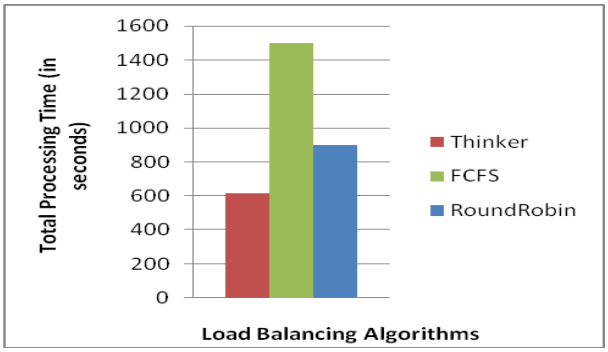

Fig 3: Total Processing Time Comparison

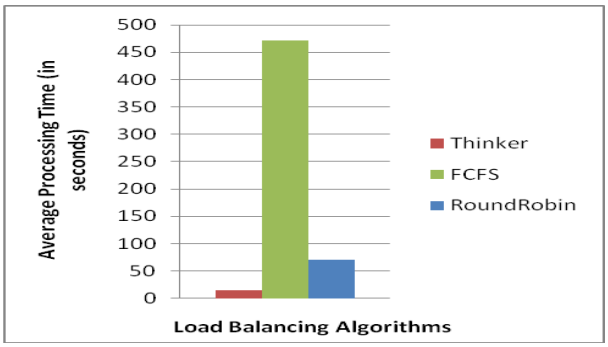

Fig 4: Average Processing Time Comparison

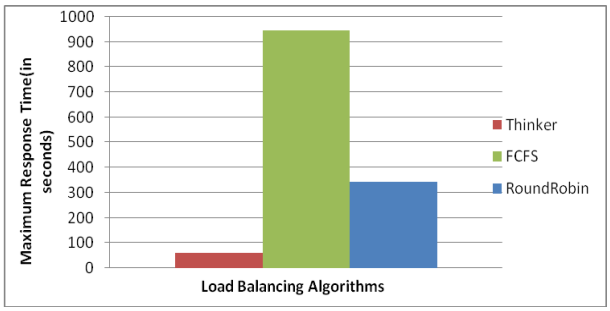

Fig 5: Maximum Response Time of each algorithm

6. CONCLUSIONS

In this paper, we have presented our load balancing thinker model which is a collaboration of all existing load balancing algorithms. Each of these individual algorithms work well only during certain situations and so our thinker decides which algorithm to choose based in a particular situation and invokes it to get the best efficiency. And it switches through the algorithms at every time interval if the situation changes.

Our model also facilitates the inclusion of the newer load balancing algorithms which would be developed by the future researchers. We have used time-outs for the thinker for our implementation. But it is better if the thinker gets invoked whenever the situation changes instead of the fixed clock intervals.

The centralized approach of thinker model might face the problems of bottleneck situation and the single-source of failure. In future, the thinker model can be implemented with a distributed approach.

7. References

[1] H. Saito, Y. Miyao, and M. Yoshida, "Traffic engineering using multiple multipoint-to-point LSPs," in *INFOCOM (2)*, 2000, pp. 894–901.

[2] W. G. Krebs, "Queue load-balancing/distributed batch processing and local rsh replacement system."

[3] O. Othman and D. C. Schmidt, "Issues in the Design of Adaptive Middleware Load Balancing," Proceedings of the 2001 ACM SIGPLAN workshop on Optimization of middleware and distributed systems, pp. 205–213, 2001.

[4] R. Krahl, J. Nolte, and L. Bttner, "A load balancing approach for the peace operating system." 2004.

[5] A. R. Karthik, "Load balancing in structured p2p systems," Berkeley, CA, 2003.

[6] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos, "Towards high performance peer-to-peer content and resource sharing systems," 2003.

[7] Barazandeh I and Mortazavi S S, "Two Hierarchical Dynamic Load Balancing Algorithms in Distributed Systems," *ICCEE '09. Computer and Electrical Engineering, pp. 516-521*, 2009.

[8]O. Othman, C. O'Ryan, and D. C. Schmidt, "The Design of an Adaptive CORBA Load Balancing Service," *IEEE Distributed SystemsOnline*, vol. 2, Apr. 2001.

[9] Yang Jiao Zhengzhou and Wei Wang, "Design and Implementation of Load Balancing of Distributed-system-based Web Server," *Electronic Commerce and Security (ISECS)*, 2010.

[10] Grosuand D and Chronopoulos A T, "A truthful mechanism for fair load balancing in distributed systems," *Network Computing and Applications, NCA 2003*, 2003.

[11] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser, "Overview of the CHORUS Distributed Operating Systems," Tech. Rep. CS-TR-90-25, Chorus Systems, 1990.

[12] W. G. Krebs, "Queue Load Balancing / Distributed Batch Processecing and Local RSH Replacement System." 1998.

[13] Lei Shi, Bin Liu, Changhua Sun, Zhengyu Yin,Laxmi N. Bhuyan, and H. Jonathan Chao, "Load-Balancing Multipath Switching System with Flow Slice," IEEE Transactions On Computers, Vol. 61, No. 3, March 2012.