Load Balancing in Heterogeneous Distributed Systems Using Singleton Model

Nikhil Saini
Department of Computer Science & Engineering
IIT Bombay
Mumbai, India
niksarrow196@gmail.com

Jeet Rabari
Department of Computer Science & Engineering
NIT Calicut
Calicut, India
jeet171196@gmail.com

Mamta C. Padole
Department of Computer Science & Engineering
The Maharaja Sayajirao University *Baroda, India*
mamta.padole@gmail.com

Vaibhav Solanki
Department of Computer Science & Engineering
The Maharaja Sayajirao University *Baroda, India*
solankivaibhav789.sv@gmail.com

2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC) | 978-1-7281-7132-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/PDGC50313.2020.9315849

***Abstract*—Load balancing is the process of improving the performance of the system by sharing of workload among the processors. The workload of a machine means the total processing time it requires to execute all the tasks assigned to it. Load balancing is one of the important factors to heighten the working performance of the cloud service provider. The benefits of distributing the workload include increased resource utilization ratio which further leads to enhancing the overall performance thereby achieving maximum client satisfaction. In this paper, we are demonstrating the use of the singleton model for load balancing.**

***Keywords*—Singleton Model, Load Balancing, Heterogeneous Distributed Systems, Multithreading, Remote Method Invocation.**

I. INTRODUCTION

Load balancing [1] is a process by which incoming client traffic is distributed among multiple servers. Load balancing enhances the performance of the system, it leads to better utilization of system resources, and it improvising responsiveness and reliability of the application. Load balancing is achieved through a load balancer which sits between clients and server farm (i.e. group of servers).

There are two types of load balancer:

- Layer 4: Layer 4 load balancer distributes traffic based upon network and transport layer protocol such as IP, TCP, UDP, FTP, etc.
- Layer 7: Layer 7 load balancer distributes traffic based upon application layer protocol such as HTTP.

The process of load balancing is straightforward, whenever the client sends a request for the service, the request is first received by the load balancer. It then applies some algorithm that identifies, to which server the request should be sent, and then forwards request to that server. The server processes the request and sends the response to the load balancer which further redirects the response to the client.

A load balancer reduces the load from a single application server and prevents an application server from becoming a single point of failure. So even if some application server fails system does not fail since the load balancer diverts load to other servers.

II. LITERATURE SURVEY

A load balancer uses a different algorithm for load balancing.

A. Round Robin:

Round robin [2] is a static load balancing algorithm. This algorithm selects the first server randomly and then, allocates the new request to other servers in round-robin fashion. This round-robin algorithm is very similar to round-robin scheduling in process scheduling.

The Pros of the Round-robin technique are:

- This algorithm tries to distribute load equally among all servers.
- It utilizes all resources in balanced manner.
- Easy to implement, it requires only one

scheduler. The Cons of Round robin technique

are:

- It does not consider current load on machine.
- It neither consider size of the task or processing power of machine.
- It is static and centralized in nature so not suitable for cloud computing.

B. Least connection method:

Least connection method [3] takes current load of server into consideration. In this method current request goes to server that is serving least number of active sessions. It is dynamic scheduling algorithm and it requires active connection to each server in pool. This algorithm assumes that all servers have same processing capacity. The Pros of Least connection method are:

- It considers current load of server.
- This algorithm is a better choice if there is a high degree of variation in the request load.

The Cons of Least connection method are:

- This algorithm performance is reduced when processing capacity of server are different.

C. Min-Min Algorithm:

Min-Min algorithm [4] consider minimum execution time and minimum completion time. This algorithm prefers Minimum execution time task before maximum execution time task. This algorithm selects task with maximum execution time and assign it to server which provide minimum completion time. Minimum completion time of server is obtained by adding current minimum completion time of server with execution time of task being assigned.

The Pros of Min-Min algorithm are:

- Easy to implement.
- Dynamic in nature, it consider processing capacity of server, size of the task and current load on server.
- Preforms better when number of jobs having small execution time is more than the jobs having large execution time.

The Cons of Min-Min algorithm are:

- Centralized in nature.
- Starvation of task is possible.

D. Max-Min Algorithm:

Max-Min algorithm [4] consider maximum execution time and minimum completion time. This algorithm prefers Maximum execution time task before minimum execution time task. This algorithm selects task with maximum execution time and assign it to server having minimum completion time.
The Pros of Max-Min algorithm are:

- Easy to implement.
- Dynamic in nature, it consider processing capacity of server, size of the task and current load on server.
- Preforms better when number of jobs having small execution time is less than the jobs having large execution time.

The Cons of Max-Min algorithm are:

- Centralized in nature.
- Starvation of minimum execution task is possible.

E. Join Shortest Queue:

This algorithm [4] uses single scheduler, whenever it receives task scheduler transfer the task to server whose queue length is small. In this no queues are maintained at scheduler level, queues are maintained at serve.
The Pros of Join Shortest Queue algorithms are:

- Easy to implement.
- Suitable for system where job arrival rate is slow.

The Cons of Join Shortest Queue algorithm are:

- It do not consider processing capacity or size of task while assigning task to machine.
- Centralized in nature.

III. METHODOLOGY

The following steps describe in detail the proposed approach for load balancing:

1. The pool of servers continuously communicate to the Name Node and send their status of resource statistics such as memory and CPU usage using Multicasting.
2. The Name Node maintains the HashMap of all the resource details of the pool of slaves/servers available at the given instance of time and keeps it updated in a file based on Heartbeat mechanism and regular communication from slaves. The HashMap is a key value pair. The key of HashMap is the IP address of a slave machine and the value corresponds to an object of a POJO containing CPU free (in percentage) and RAM free (in MB). The removal of details of inactive slaves, timely updation in a common file, and it being concurrently used by the decision engine is achieved using Java Collections, File Handling, Singleton Pattern design and Process Synchronization.
3. The Client machines submits a request containing their query to the Name Node via TCP connection. The query formation is done after reading a file on client machine and transmitting it in byte format.
4. On reception of a client request, the Name Node reads the current server farm information in a HashMap (using Singleton Pattern design, Process Synchronization and File Handling) and executes decision engine algorithm on it to decide the target slave server.
5. The Decision Engine module returns the IP address of target machine which is the outcome of implemented algorithm. A naive approach is implemented which filters out slaves based on some threshold CPU free (in percentage) and picks the one with highest RAM free (in MB) among the remaining candidates.
6. Name Node creates a thread to service the request of the current client and itself remains idle waiting for other clients over TCP. The thread possesses information like TCP socket to the client, target IP address and port, and the query received.
7. The thread is responsible for reading clients' queries and calling a remote method on target (server machine) using RMI (Remote Method Invocation). The result received from the remote method is passed back to the requesting client and the TCP connection is terminated.
8. RMI servers run on each slave of the server pool. The Name Node calls the remote method on them and retrieves the computed result. On boot-up of a slave machine, the RMI server and Multicast Client automatically start up using a .bat (batch file) which is put in the startup folder (in case of Windows OS).
9. While one slave is busy servicing one client, its new resource parameters are concurrently updated at Name node.
10. A new request will be directed to some other slave and the process continues.

IV. TECHNOLOGIES USED & THEIR PURPOSE

A. Multicasting:

The main purpose of using multicasting is to create a multicast group to transfer system information from the slave server to the Name Node. Any machine can join or leave this multicast group. Having a multicast group provide us flexibility for scaling system resource as per need. A new slave server can easily be added to an existing system without affecting the working of the system. Moreover, the server can easily be removed from the multicast group. So multicasting increases system scalability.

B. RMI

- Parallel Computing: RMI [6] is multi-threaded, allowing your servers to exploit Java threads for better concurrent processing of client requests. RMI handles threads and sockets by itself.
- Fast, safe and secure: RMI provides a means for clients behind firewalls to communicate with remote servers. Traversing the client's firewall can slow down communication, so RMI uses the fastest successful technique to connect between client and server. The technique is discovered by the reference for UnicastRemoteObject on the first attempt the client makes to communicate with the server by trying each of three possibilities in turn:

  Communicate directly to the server's port using sockets.

  If this fails, build a URL to the server's host and port and use an HTTP POST request on that URL, sending the information to the skeleton as the body of the POST. If successful, the results of the post are the skeleton's response to the stub.

  If this also fails, build a URL to the server's host using port 80, the standard HTTP port, using a CGI script that will forward the posted RMI request to the server.

  This three-stage back-off allows clients to communicate as efficiently as possible.

- Object Oriented: RMI can pass full objects as arguments and return values, not just predefined data types. This means that you can pass complex types, such as a standard Java hashtable object, as a single argument. This empowers this architecture to serve any kind of clients query.
- Runtime Execution: The call to slave servers is made when needed. This enhances their availability for other requests and improves resource utilisation.
- Distributed Garbage Collection: RMI uses its distributed garbage collection feature to collect remote server objects that are no longer referenced by any clients in the network.
- Dynamic loading of classes is available. Server side changes in code can be made without altering the client code.

C. TCP

TCP communication is used between clients and Name Node. The main purpose of using TCP to provide reliable communication between clients and Name Node. Since the client will be sending a file to the Name Node server containing information required for serving a request, a TCP connection is the best choice for transferring a file from the client to Name Node. Moreover, the file may be large size containing large information and the nature of the request is also not real-time which makes use of TCP more preferable.

VI. IMPLEMENTATION

This section describes the structure of our implementation. The proposed architecture is shown in "Fig. 1".

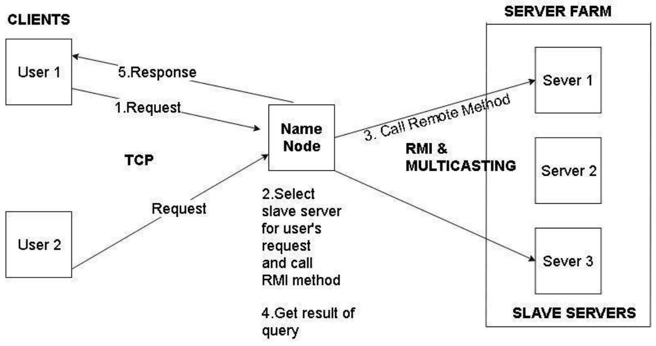

Fig. 1. Proposed Architecture

A user submits its request to the name node "Fig. 1". A large number of users can access the utility simultaneously. The user is never aware of the slave which will execute its request. The user module communicates with the Name node only using TCP. The connection-oriented protocol is the pillar of reliability and efficiency.

A. NameNode:

The NameNode (Load Balancer) is the central manager.

It handles the following duties:

- Multicast Sender
- Decision Engine
- TCP Server
- RMI Client

The multicast module at name node and slaves accomplishes the task of having an updated set of slave server's resource statistics at the Name node. The RMI module where the Name node acts as RMI client serves the core task of submitting the job at the RMI server(at slave) and sending back the results obtained from it back to the requesting client.

B. Server Farm:

Server farm consists of multiple number of Servers that provide services requested by clients, also referred to as Slave Servers. A Slave server is responsible for processing actual client request. Two services run on the slave server.

- RMI service: RMI server is run on each slave server and executes the methods available with it on the parameters received from Name node and returns the respective result.

- Multicast Service: Multicast clients run on each server machine and they use oshi [5] library to retrieve their system stats and then transmits them to the Name node. getSystemCpuLoad() & getAvailable() methods which are accessed by acquiring object reference by the methods getProcessor() & getMemory() of HardwareAbstractionLayer class are used to retrieve resource utilisation.

VII. RESULTS

The results of the experiments performed are described in Table I & Table II.

TABLE I. TEST CASE I

Machine Name	*Key (IP Add.)*	*RAM Free(MB)*	*CPU Free(%)*
Slave 1	192.168.43.57	1605.45	33.59
Slave 2	192.168.43.65	2204.15	45.88
Slave 3	192.168.43.78	1476.14	67.45
Slave 4	192.168.43.34	2315.45	49.45
Slave 5	192.168.43.64	1703.24	57.56

TABLE II. TEST CASE II

Machine Name	*Key (IP Add.)*	*RAM Free(MB)*	*CPU Free(%)*
Slave 1	192.168.43.57	1634.15	23.65
Slave 2	192.168.43.65	3124.21	78.54
Slave 3	192.168.43.78	1547.65	46.57
Slave 4	192.168.43.34	2451.24	65.54
Slave 5	192.168.43.64	2145.14	63.54

In both test cases, Slave 1 has the following configurations:

RAM: 2 GB, Processor: intel i3@2.2Ghz OS: Windows 10, and Slave 2, 3, 4, 5 have configurations as: RAM: 4 GB, Processor: intel i5@2.2Ghz OS: Windows 10.

Table I shows that, CPU free threshold: 40%. According to the naive approach described in methodology (point 5), Slave 1 gets rejected since its CPU free is less than 40%. The rest machines compete and Slave 4 wins with the highest RAM free. Therefore, NameNode directs the received request to Slave 4. CPU free threshold: 40%. Slave 1 gets rejected since its CPU free is less than 40%. Now, Slave 2 wins with the highest RAM free. Therefore, NameNode directs the received request to Slave 2.

VIII. CONCLUSION

In this paper, we present an implementation to do load balancing in heterogeneous distributed systems using a singleton model by making use of simple technologies provided by Java Programming language. Since Java is a platform-independent programming language the same solution can work seamlessly on Heterogeneous Distributed Systems. The concept that has been implemented and discussed in this paper, can work for any given kind of application. The Object-Oriented concepts in Java enable us to invoke any POJO containing the Business logic of an Application from the RMI Server program without editing the code. With minor modifications, as to how the client would submit the request to the Name Node, the same concepts of load balancing can be applied to cloud computing also.

IX. FUTURE SCOPE

The naive load balancing algorithm can be improved so that the algorithm effectively calculates the target slave server while maintaining the throughput of the system. Tracking of jobs dispatched to slave servers until their successful completion can be employed which will improve the reliability of the system. Timestamps can be used to monitor dispatched jobs. The proper analysis was done on the frequency and types of jobs submitted and the ways slaves respond through a background process will help to improve the merits of the load balancer.

REFERENCES

[1] https://www.citrix.co.in/glossary/load-balancing.html

[2] Abhijit Aditya , Uddalak Chatterjee and Snehasis Gupta, “A Comparative Study of Different Static and Dynamic Load Balancing Algorithm in Cloud Computing with Special Emphasis on Time Factor” (E-ISSN 2277 – 4106, P-ISSN 2347 – 5161).

[3] Dr. Mustafa ElGili Mustafa, Load balancing algorithms round-robin (rr), leastconnection, and least loaded efficiency, GESJ: Computer Science and Telecommunications 2017|No.1(51) ,ISSN 1512-1232.

[4] http://shodhganga.inflibnet.ac.in/bitstream/10603/16932 4/15/15_chapter3.pdf

[5] https://github.com/oshi/oshi

[6] Dr. Mamta C. Padole “Big Data Analysis on Heterogeneous Distributed Systems using Remote Method Invocation” IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 2, Ver. V (Mar.-Apr. 2017), PP 70-75