Load Balancing in Peer to Peer Networks

Project report submitted in fulfillment of the requirement for the degree of Bachelor of Technology in

Computer Science Engineering

By

Ankit Raina (121269)

Under the supervision of

Miss Ramanpreet Kaur

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

This is to Certify that the work presented in this report entitled,” Load balancing in peer to peer networks” in the fulfillment for the award of degree of **Bachelor of Technology** in **Computer Science & Engineering** from **Jaypee University of Information Technology , Waknaghat** is an authentic record of my own work carried out over a period from August 2k15 to June 2k16 under the supervision of Miss Ramanpreet Kaur, Department of Computer Science & Engineering.

The matter embodied in the report has not been submitted for award of any other degree or diploma.

Student Name: Ankit Raina

Student’s Roll No: 121269

This is to certify that the above statement made by candidate is true to the best of my Knowledge.

Signature of Supervisor:

Name of Supervisor: Ramanpreet kaur

Designation:

Department Name: CSE

Date:

ACKNOWLEDGEMENT

The research work related to this Project has been carried out at Jaypee University of Information Technology, Computer Science and Engineering Laboratory, Department of Computer Science Engineering.

I would like to express my great gratitude to my supervisor, Professor Miss Ramanpreet, for supervising this thesis. Her contributions in directing and supervising the research work carried out here are conclusive. Without her direction and guidance, the Project work would not be possible.

I am obliged to the staff members of the college (JUIT, Solan), for the valuable information provided by them in their respective fields. We are grateful for their cooperation during the period of the project.

Finally, I would like to thank my Parents and Friends for supporting every decision I have made in my life.

Jaypee University of Information Technology
June 2k16

ABSTRACT

Peer to peer systems have emerged as an appealing solution for sharing and locating resources over the Internet. Several P2P systems have been successfully deployed for a wide range of applications, such as Gnutella, Bit Torrent and Skype (Internet telephony system).

A Peer makes a portion of its resources such as processing power, disc storage or network bandwidth directly available to other network (peers) without the need for central coordination by servers or stable hosts. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. One of the most commonly used applications of load balancing is to provide a single Internet service from multiple servers, sometimes known as a server farm. We are evaluating the performance of load balancing strategies in structured overlay networks. This study discusses the pros and cons of the load balancing strategies in structured overlay network. The goal of our project is to select high performing load balancing strategy to evenly divide the load among the processing elements.

It is critical to design a P2P system that is scalable and efficient. To build an efficient P2P system, researchers have turned to structured architectures which offer a bound on search performance as well as completeness of answers. However, one key challenge that has not been adequately addressed in the literature is that of load balancing. In a large-scale P2P system, nodes often have different resource capabilities. Hence, it is desirable that each node has a load proportional to its resource capability. We implement a framework, called Histogram-based Global LOad Balancing (HiGLOB) to facilitate global load balancing in structured P2P systems. In this paper, we present a general framework, HiGLOB, for global load balancing in structured P2P systems. Each node in HiGLOB has two key components:

1) A histogram manager maintains a histogram that reflects a global view of the distribution of the load in the system, and

2) A load-balancing manager that redistributes the load whenever the node becomes overloaded or under-loaded.

TABLE OF CONTENT

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 Peer to Peer:

Peer to peer is a decentralized communication model in which each party has the same Capabilities and either party can initiate a communication session.

A Peer makes a portion of its resources such as processing power, disc storage or network bandwidth directly available to other network (peers) without the need for central coordination by servers or stable hosts.

A Peer can be a Server as well a client.

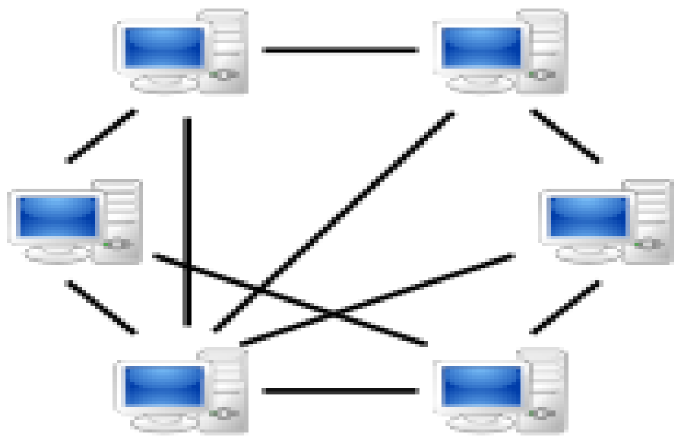


Figure1

In a peer to peer Network the workload is distributed among all the peers which are equally privileged and the distribution is done on the basis of different load balancing techniques.

1.2 Historical Development:

P2P systems had previously being used in many application domains but the concept was popularized by file sharing systems such as music sharing application **'Napster'**, originally released in 1999.

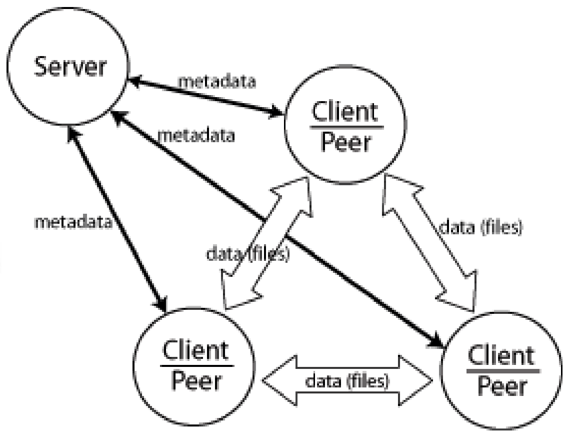


Figure 2

Early vision for world wide web (WWW) was close to a P2p network as it assumed each user of the web would be an active editor and contributor creating and linking content to form an inter-linked web of Links.

1.3 Architecture:

Peer to peer network is designed around the notion of equal peer nodes simultaneously functioning as both, client and server to other nodes on the network. This module of network arrangement differs from the Client server model where communication is usually to and fro from a central server.

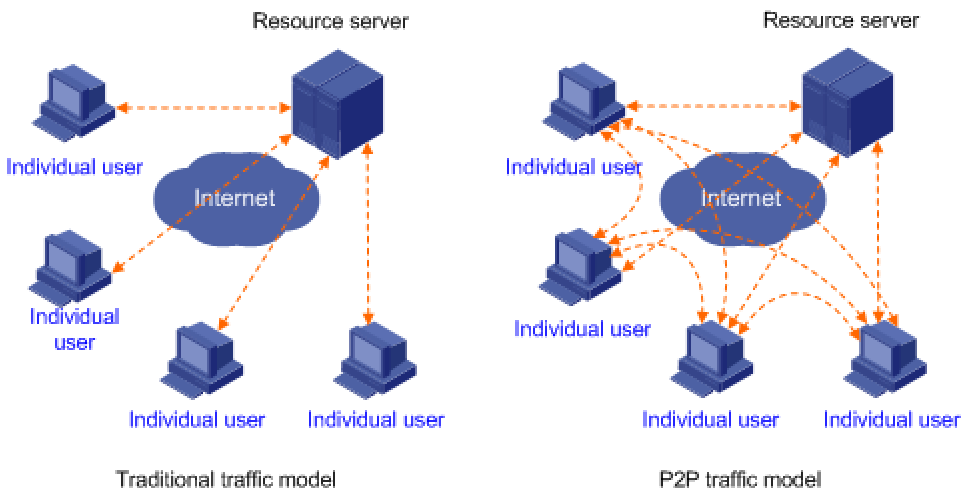


Figure 3

A typical example of a File transfer that uses the client server model is the File Transfer protocol [FTP] in which the client and server programs are distinct, the Clients initiate the transfer and the servers satisfy these requests, whereas in a p2p network any peer can initiate the transfer and any peer can share resources if available.

1.4 Routing and Resource Recovery:

P2P networks generally implement some form of virtual overlay network on top of physical network topology, where the nodes in the overlay form a subset of nodes in the physical networks. Data is exchanged directly over the underline TCP/IP network, but at the application layer peers are able to communicate directly, via the logical overlay links. Overlays are used to make the p2p system independent of the physical network topology. Based on the how the nodes are linked to each other within the overlay network and how resources are indexed, we can classify networks as

1. Unstructured
2. Structured

1.5 Advantages of Peer to Peer networks:

1. It is easy to install and so is the configuration of computers on this network.

2. All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources.

3. P2P is more reliable as central dependency is eliminated. Failure of one peer doesn't affect the functioning of other peers.

4. There is no need for full-time System Administrator. Every user is the administrator of his machine. User can control their shared resources.

5.The over-all cost of building and maintaining this type of network is comparatively very less.

1.6 Load balancing:

In computing load balancing distributes workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives.

Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. One of the most commonly used applications of load balancing is to provide a single Internet service from multiple servers, sometimes known as a server farm. Only Internet, companies whose website get a great deal of traffic usually use load balancing. There are various approaches for load balancing the web traffic, for eg Domain name system [DNS]

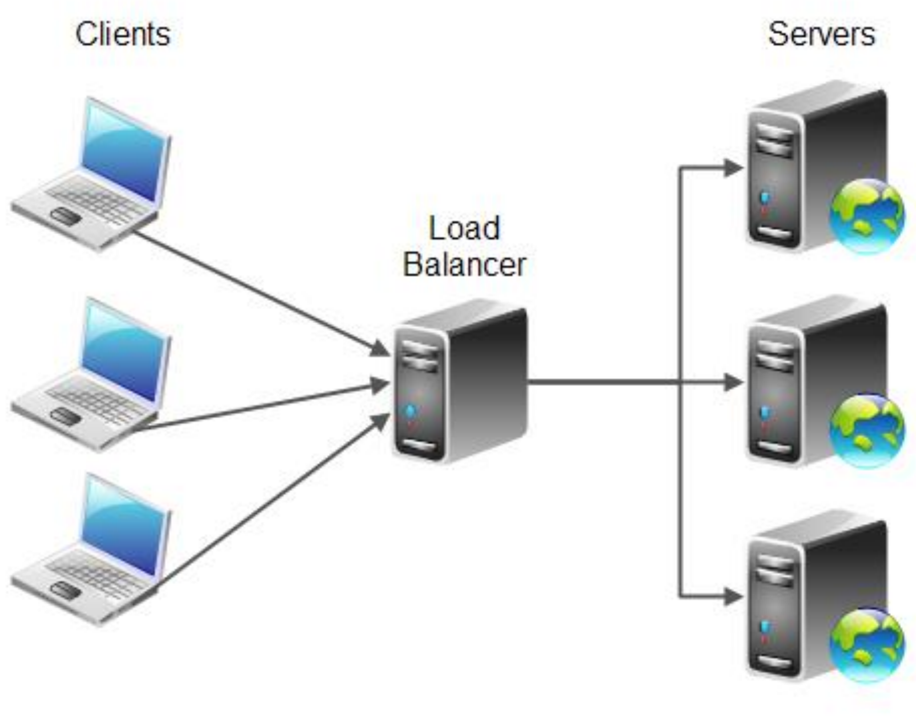


Figure 4

Goals of Load balancing: In order to balance the requests of the resources it is important to recognize a few major goals of load balancing algorithms:

a) Cost effectiveness: Primary aim is to achieve an overall improvement in system performance at a reasonable cost.

b) Scalability and flexibility: The distributed system in which the algorithm is implemented may change in size or topology. So the algorithm must be scalable and flexible enough to allow such changes to be handled easily.

c) Priority: Prioritization of the resources or jobs need to be done on beforehand through the algorithm itself for better service to the important or high prioritized jobs in spite of equal service provision for all the jobs regardless of their origin.

1.7 Problem statement:

The problem that we are planning to tackle is to implement a peer to peer network and then suggesting a Load balancing technique to balance load amongst different peers in the network.

We are implementing a technique called Hi-Glob (Histogram-based Global Load Balancing). The basic approach to load balancing is to find a pair of nodes—one that is heavily loaded and the other lightly loaded—and redistribute the load across these two nodes.

Chapter 2: Literature Survey

This chapter provides an overview of the research topics related to Web Services and P2P converged framework, hierarchical architecture and Performance evaluation.

A lot of information have been gathered and investigated from various books, research papers, journals and online links. In this chapter, the details of research papers and journals are specified from where we have analyzed the contents and formulated the problem.

A number of research scholars and scientists have written a number of research papers and found excellent results. This section underlines all those research papers and their extracts.

2.1 Overview of Research Papers/Journals

2.1.1
Title: Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems.
Authors: Quang Hieu Vu, Beng Chin Ooi, Martin Rinard, and Kian-Lee Tan
Date of Publishing: APRIL 2009

Journal name: IEEE Transactions ON Knowledge and data engineering, VOL. 21

Summary:
In summary the article have proposed that over the past few years, peer-to-peer (P2P) systems have rapidly grown in popularity and have become a dominant means for sharing resources. In these systems, load balancing is a key challenge because nodes are often heterogeneous. While several load-balancing schemes have been proposed in the literature, these solutions are typically ad hoc, heuristic based, and localized. In this paper, they present a general framework, HiGLOB, for global load balancing in structured P2P systems. Each node in HiGLOB has two key components:

1. A histogram manager maintains a histogram that reflects a global view of the distribution of the load in the system, and

2. A load-balancing manager that redistributes the load whenever the node becomes overloaded or under-loaded.

The basic approach to load balancing is to find a pair of nodes—one that is heavily loaded and the other lightly loaded—and redistribute the load across these two nodes. However, it is far from trivial to balance the load in a P2P system. There are two main issues in P2P's load balancing: 1) how to determine if a node is overloaded or under-loaded, and 2) if so, how to find a suitable partner node with which to redistribute the load.

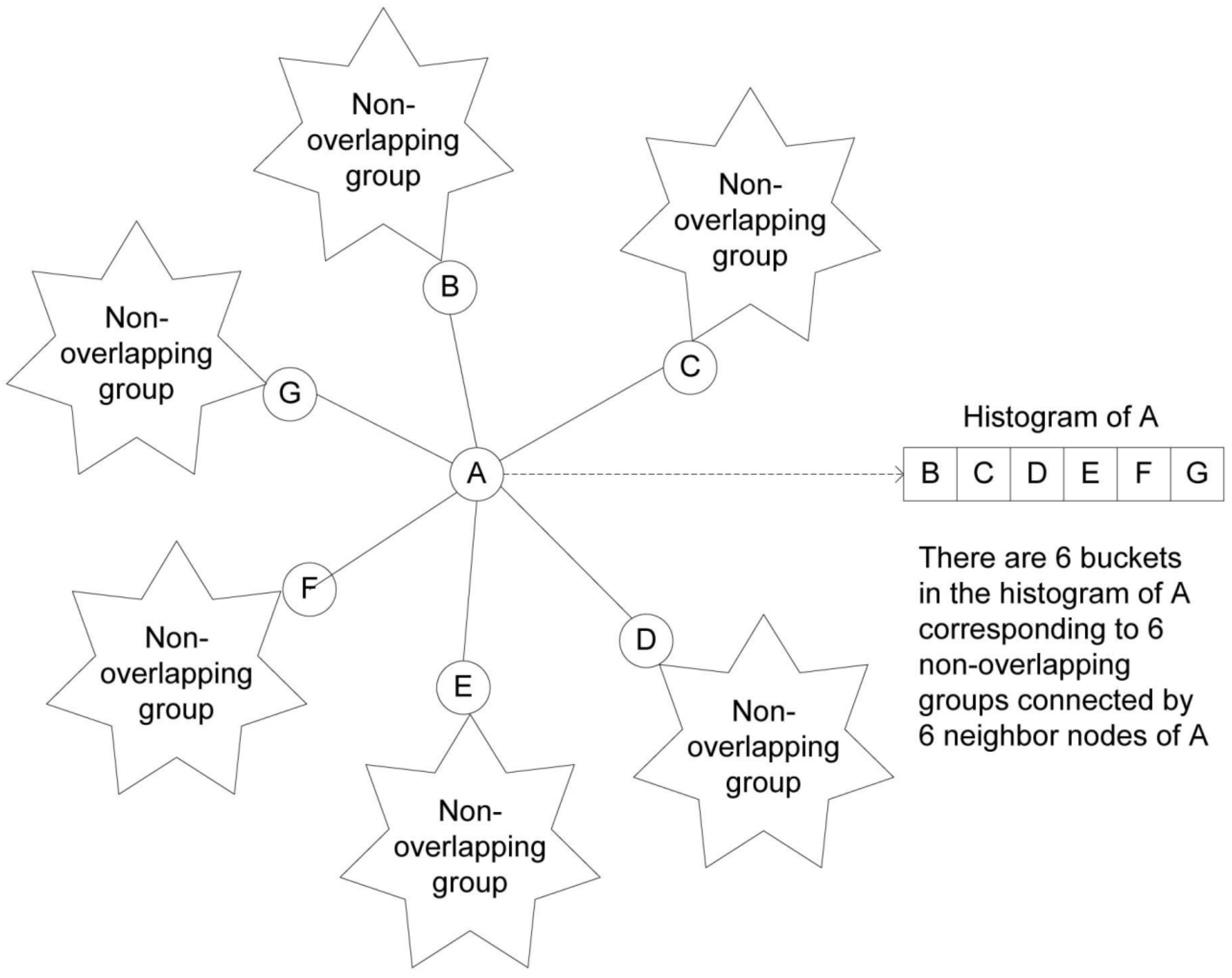


Figure 5

Improvement Techniques

While histograms are useful, the cost of constructing and maintaining them may be expensive especially in dynamic systems. As a result, we introduce two techniques that reduce the maintenance cost.

1. **Reduce the cost of constructing histogram**: Constructing a histogram for a new node may be expensive since it requires histogram information from all neighbor nodes. Additionally, the histograms of the new node's neighbors also need to be updated since adding a new node to a group of nodes changes the average load of that group. Constructing and maintaining histograms may therefore be expensive if nodes join and leave the system frequently. In light of the fact that every new node in the P2P system must find and notify its neighbor nodes about its existence while these neighbor nodes need to send their information to the new node to setup connections after that, we suggest that histogram information can be piggybacked with messages used in this process. In this way, we can avoid sending separate histogram messages and totally eliminate the effect of node join on the histogram construction of the new node and histogram update of its neighbor nodes

2. Reduce the cost of maintaining histogram: Maintaining histograms can be expensive since any load change at a node causes an update to be propagated to all other nodes in the system. To avoid this propagation, we suggest that we do not need to keep exact histogram values. We only need to keep approximate values in the histogram. A node only needs to send load information to other nodes when there is a significant change in either its load or the average load of a non-overlapping group in its histogram.

2.1.2

Title: New Algorithms for Load Balancing in Peer-to-Peer Systems

Authors: David R. Karger, Matthias Ruhl

Date of Publishing: June 2005

Place: MIT Laboratory for Computer Science Cambridge, MA 02139, USA

Summary:

Load balancing is a critical issue for the efficient operation of peer-to-peer networks. We give new protocols for several scenarios, whose provable performance guarantees are within a constant factor of optimal. First, it gives an improved version of consistent hashing, a scheme used for item to node assignments in the Chord system. In its original form, it required every network node to operate $O(\log n)$ virtual nodes to achieve a balanced load, causing a corresponding increase in space and bandwidth usage. The protocol eliminates the necessity of virtual nodes while maintaining a balanced load.

2.1.3

Title: Efficient Load Re Balancing Algorithm for Distributed File Systems

Authors: Revathy R , A.Illayarajaa

Date of Publishing: May 2013

Journal name: International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-6.

Summary:

A node is light if the number of modules it hosts is smaller than the threshold as well as, a heavy node manages the number of modules greater than threshold. A large-scale distributed file system is in a load-balanced state if each module server hosts no more than a specified number of modules. In our proposed algorithm, each module server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. This process repeats until all the heavy nodes in the system become light nodes.

First of all we are going to notice the lightest node to require the set of modules from heaviest node. Load equalization may be a technique to distribute employment across several computers or network to realize most utilization of resources economical output, reducing latency, and take away overload. The load equalization service is sometimes provided by dedicated code or hardware, like a multilayer switch or name server. During this project we have a tendency to use Load rebalancing formula. Then identical method is dead to unleash the additional load on following heaviest node within the system. Then we are going to once more notice the heaviest and lightest nodes, such a method repeats iteratively till there's not the heaviest.

Advantages of Proposed System: Using this we can use it in large scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size. Another advantage of the proposed system is the security consistency provided by it. Various nodes with heavy loads have been proposed as alternatives to central node module so that so many drawbacks of the existing system can be avoided. The proposed system will help in keeping the system consistent so that we can avoid data loss.

2.1.4

Title: Effective load-balancing of peer-to-peer systems

Authors: Anirban Mondal Kazuo Goda Masaru Kitsuregawa

Place**:** Institute of Industrial Science University of Tokyo, 4-6-1, Komaba, Meguro-ku, Tokyo 153-8505, Japan

Abstract:

The paper views the system as comprising several clusters, where peers are assigned to clusters such that the clusters are mutually disjoint. In the proposed strategy, every peer is assigned a unique identifier *peer id* and *all* peers belonging to the same local area network (LAN) are initially assigned to a single cluster. Every incoming query is assigned a unique identifier *Query id* by the peer *Pi* at which it arrives. *Query id* consists of *peer id* and *num* (a distinct integer generated by *Pi*). Every peer maintains its own access statistics i.e., the number of disk accesses made for each of its data items *only* during the last time interval. Each cluster is randomly assigned a leader. The job of the cluster leaders is to coordinate the activities (e.g. load-balancing, searching) of the peers in their clusters. Each cluster leader also maintains information concerning the set of categories stored both in its own cluster as well as in its neighbouring clusters. Category-related update information is periodically exchanged between neighbouring cluster leaders preferably by piggybacking.

Load-balancing can be achieved by transferring hot data from heavily loaded peers to lightly loaded peers via data migration or data replication. Note that unlike replication, migration implies that once hot data have been transferred to a destination peer, they will be deleted at the source peer.

2.2 Types of P2P networks:

2.2.1 Hierarchical P2P architecture:-

As stated in the previous chapter, hierarchical architecture almost always accompanies the large-scale, complex distributed systems.

DNS was even designed as a hierarchy from the date of its birth, and it still shows its benefit in terms of scalability. On the current Internet, routers are grouped into various Autonomous Systems (AS). Different routing protocols are used for the intra-AS routing and inter-AS routing; while the former utilizes Routing Information Protocol (RIP) or Open Shortest Path First (OSPF) protocol, etc., the latter uses Border Gateway Protocol (BGP), etc.

2.2.2 Taxonomy of hierarchical P2P architecture:-

Based on the overlay topology, and the construction and organization of network connections, P2P networks can be classified as structured P2P network and unstructured P2P network. Analogously, hierarchical P2P networks can also be roughly classified as structured hierarchical P2P networks and unstructured hierarchical P2P networks in the same way. However, as hierarchical P2P systems utilize more than one level (or tier) hierarchy to distribute the overlay nodes, multiple options can be taken by each level, either structured or unstructured. Depending on the adopted overlay topology by each level, hierarchical P2P systems can be further classified into three categories: unstructured, structured, and hybrid

Unstructured hierarchical P2P systems utilizes unstructured topology at each level only. Structured hierarchical P2P systems, just as its name implies, utilize only structured topology at each level. Hybrid hierarchical P2P systems combine unstructured and structured overlay topology in its hierarchy. Based on the practical requirements, hybrid hierarchical P2P systems can utilize structured overlay topology at its upper level while utilizing unstructured overlay topology at its lower level, or vice versa.

2.2.3 Unstructured Networks:-

Unstructured p2p networks do not impose a Particular structure on the overlay network by design but rather are formed by nodes that randomly form connections to each other. Eg. of such networks are Gnutella, Gossip and kaZaa.

As there is no structure imposed upon them, Un-structured networks are easy to built. Also, as the role of all peers in the network is the same, they are Highly Robust in the aspect that when a large number of peers are frequently joining and leaving the network. The Primary limitations of Un-structured networks also arrives from this lack of structure.

In particular, when a peer wants to find a designed data in the network, the search query must be flooded through the network to find as many as peers as possible that shared the data.

Flooding causes a very high amount of signaling traffic in the network, uses more CPU / memory, and does not ensures that search queries will always be resolved.

2.2.4 Structured Networks:-

In structured p2p Networks the overlay is organized into a specific topology, and the protocol ensures that any node can efficiently search the network for a file even if the resource is extremely rare. The most common type of Structured p2p networks implement a distributed Hash table [DHT], in which a variant of consistent hashing is used to assign ownership of each file to a particular peer. This enables peers to search for resources on the network using a Hash table. However in order to route traffic efficiently through the network, nodes in structured overlay must maintain list of neighbours that satisfy specific criteria.

2.2.5 Related concepts of P2P computing

In order to make the description of the subsequent sections clearer, the following concepts are firstly introduced:

Peer-to-Peer system: Peer-to-Peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation of peers involved in the system.

Overlay: Overlay network is abstracted from and constructed above the underlying physical computer network, usually IP network. Accordingly, it shields itself from the underlying complicated physical computer network connection details. One hop in overlay network can be mapped onto multiple hops in physical networks involved in the case.

Peer: Peer is the participant of P2P overlay networks which acts both as client and server at the same time. Thus, it is sometimes referred to as servant. Peer is less powerful in capability and responsibility compared with super-peer.

Super-peer: Super-peer is the counterpart of peer. It is usually more powerful and stable compared with peer, accordingly, more responsibilities are supposed to be taken or more services are supposed to be provided by super-peer than peer. For example, super-peers are responsible for routing, storing and forwarding information on behalf of peers, while peers are only responsible for initiating service requests and receiving the associated responses.

Structured P2P network: The P2P overlay network topology is tightly controlled and content is not placed at random peers but rather at specified locations that makes subsequent queries more efficient .Structured P2P network usually utilizes DHT as a substrate. Chord and Kademlia are examples of this category. The differentiation of structured and unstructured is based on the topology of P2P overlay networks, for example, whether the construction of overlay network is based on some special rules or the overlay network is constructed randomly.

Unstructured P2P network: The P2P overlay network organizes peers in a random graph in a flat or hierarchical way, and uses flooding, random walks or expanding ring. Time-to-Live (TTL) search on the graph to query content stored by overlay peers Gnutella v0.4 is a typical example of this category. The distinction between hierarchical and flat P2P network is based on how many levels the network topology is utilizing.

Hierarchical P2P network: As its name shows, hierarchical P2P network utilizes multiple, usually equal to or larger than two, levels of hierarchy. Participants of hierarchical P2P network are divided into different roles, for example, super-peer and peer, as afore mentioned, based on their capabilities and reliability. Generally, the network topology of each level can be different, for example, the upper level can be structured P2P network, while the lower level can be unstructured P2P network.

Hybrid P2P network: Hybrid P2P network refers to combinations of different principles in the system organization. The term hybrid is more general than the term hierarchical. Structured and Unstructured principles can both be used in hybrid organizations. Loo et al. (2005) is an example of this category.

Churn: The phenomenon of the continuous arrival and departure of participating peers of a P2P overlay.

Graceful leaving: A departing peer notifies its upcoming departure to its neighbours and transfers the resource items it stored to some other peer(s).

Ungraceful leaving: A departing peer neither notifies its departure nor transfers resource items to its neighbours. This occurs upon certain failure events of peers, e.g. an abrupt network disconnection or running out of power.

Parallelism degree: The number of same requests that an initiating peer sends in parallel to its routing neighbours. Parallel requests only take place in the resource lookup process.

Flat P2P network: Just as its name implies, participants of flat P2P network are located at one level. Their roles are equal in terms of the undertaking responsibilities. No hierarchy is utilized to distribute the peers. Earlier version of Gnutella, namely Gnutella v0.4 and basic form of Chord are example of this category.

Throughput is utilized to ascertain the number of job whose processing has been accomplished. It ought to be maximized to enhance the execution of the system.

Overhead Associated decides the measure of cost of actualizing a load balancing technique in term of time. It is made out of extra cost because of migration of jobs, between various process interactions and processors. This ought to be least so a load adjusting strategy can perform effectively.

Fault Tolerance: The capacity of an algorithm to perform uniform load balancing despite subjective nod or connection failure. The load balancer ought to be a fault tolerant strategy.

Migration time: The total duration to move the tasks or processes among the nodes available in the system. It ought to be least in order to upgrade the execution of the system.

Response Time: The measure of time elapsed to react by a specific load adjusting mechanism in a disseminated system. It ought to be least.

Resource Utilization: It is utilized to analyses the use of resources. It ought to be advanced for an effective burden adjusting.

Scalability: It is the capacity of an algorithm to operate load balancing efficiently the size of system increased or decreased. It ought to be enhanced.

Performance: It is utilized to check the effectiveness of the system. This must be enhanced at a sensible expense, e.g. minimize tasks response time while keeping worthy deferrals.

2.3 Distributed Hash Tables:-

A distributed hash table is a class of a decentralized distributed system that provides a look up service similar to a hash table. Peers are stored in a DHT and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from key to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption.

2.3.1 History:-

DHT research was originally motivated, in part, by peer to peer systems such as free-net, Gnutella, Napster, which took advantage of resources distributed across the internet to provide a single useful application.

2.3.2 Properties:-

DHT's characteristically emphasise the following properties:-

1. Autonomy in decentralization- The nodes collectively form the system without any central Coordination.

2. Fault Tolerance- The system should be reliable even with nodes continuously Leaving, Joining and Failing.

3. Scalability- The system should function efficiently even with thousands of millions of nodes.

2.3.3 Structure:-

The structure of a DHT can be decomposed into several main components. The foundation is an abstract key-space such as set of 160 bits string. A key-space partitioning scheme splits ownership of this Key-space among the participating nodes.

An overlay network then connects the node, allowing them to find the owner of any given key in Key-space

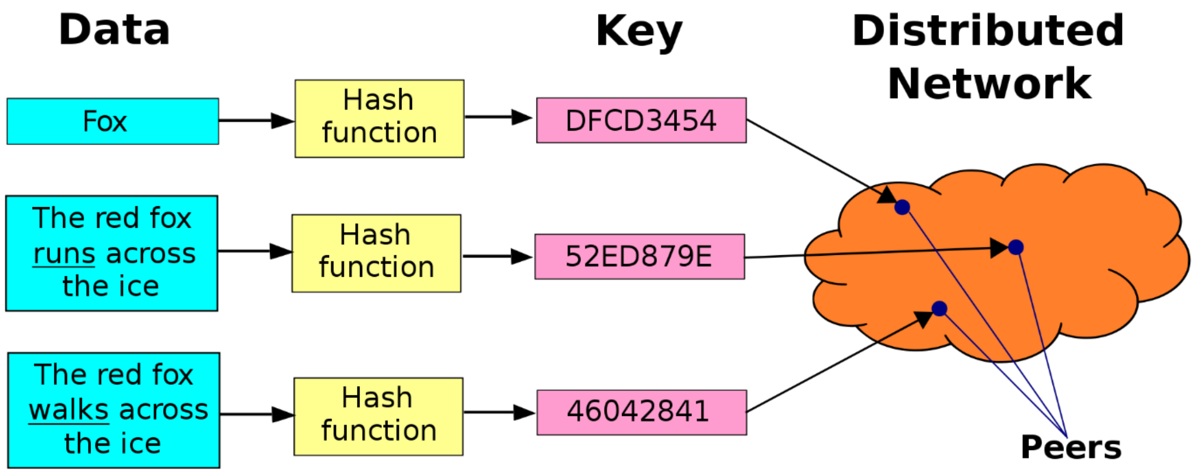


Figure 6

2.3.4 Working of DHT:-

Node Arrival

Bootstrapping/Joining of a new node

1. Calculation of node ID

2. New node contacts DHT via arbitrary node

3 Assignment of a particular hash range.

4 Copying of K/V-pairs of hash range (usually with redundancy).

5 Binding into routing environment (of overlay).

Node Failure / Departure

Failure of a node

-Use of redundant K/V pairs (if a node fails)

-Use of redundant / alternative routing paths

-Key-value usually still retrievable if at least one copy remains

Departure of a node

- Partitioning of hash range to neighbor nodes.

- Copying of K/V pairs to corresponding nodes.

- Unbinding from routing environment.

2.3.5 Key-space partitioning:-

Most DHTs use some variant of consistent hashing and rendezvous hashing to map keys to nodes. The two algorithms appear to have been devised independently and simultaneously to solve the distributed hash table problem. Both have the essential property that removal or addition of one node changes only the set of keys owned by nodes with adjacent IDs and leaves all other nodes unaffected. Contrast this with the traditional hash table in which addition or removal of one bucket causes nearly the entire key-space to be remapped.

2.4 Consistent Hashing:-

Consistent hashing is a special type of hashing such that when a hash table is resized and it is used, only K/n keys need to be remapped on average, where K is the number of keys and n is number of slots. In contrast, in most traditional hash tables a change in number of array slots causes nearly all keys to be remapped.

2.4.1 History:-

The term consistent hashing was introduced by Karzer for use in distributed cashing, the idea has now been extended to other areas also. An academic paper from 1987 introduced the term consistent hashing as a way of distributing requests among a changing population of web servers. The addition and removal of nodes only requires K/n items to be re-shuffled when the number of slots changes.

2.4.2 USES:-

The consistent hashing concept also applies to design of DHT (distributed hash tables). DHTs use consistent hashing to partition a key-space among a distributed set of nodes and additionally provide an overlay network that connects nodes such that the node responsible for any key can be efficiently loaded.

The main idea behind consistent hashing is to associate each cache with one or more hash value intervals where the interval boundaries are determined by calculating the hash of each cache identifier. It has also been used to reduce the impact of partial system failures in large web applications.

2.5 Rendezvous Hashing:-

Rendezvous hashing is an algorithm that allows clients to achieve distributed agreement on a set of K options out of a possible set of n options. A typical application is when clients need to agree on which sites objects are to assign to. When K is 1 it accomplishes goals similar to consistent hashing using an entirely different method.

2.5.1 History:-

It was invented in 1996 by Chinya Ravishankar at the University of Michigan. It appears to have been devised independently and simultaneously at MIT. One of the first application of this was to enable multicast clients on Internet to identify multicast rendezvous points in distributed fashion.It has also been used in applications such as mobile caching, router design and secure key establishment .

2.5.2 Properties:-

Rendezvous hashing has following properties:

1. Low overhead: the hash function used is very efficient so the overhead involved is very low.
2. Load balancing: since the hash function is randomizing, each of the n sites is equally likely to receive the object O therefore loads are uniform across the sites.
3. High hit rate: since all the clients agree on placing an object O into the same site S0 each fetch of O into S0 yields the maximum utility in terms of hit rate.

2.6 Advantages of structured over unstructured peer to peer networks

Why structured Networks?

In structured overlays, the identifier of the joining peer determines the set of peers that it connects to. Early instantiations of Gnutella were unstructured -- keyword queries were flooded widely. Napster had decentralized content and a centralized index, so it only partially satisfies the distributed control criteria for P2P systems. More over Structured graphs can find keys efficiently, irrespective of replication.

Chapter 3: SYSTEM DEVELOPMENT

Chord P2P Network:-

3.1 Over-view:

-Early and successful algorithm

-Simple & elegant

-Easy to understand and implement

-Many improvements and optimizations exist

Main responsibilities:

1. Flat logical address space: l-bit identifiers instead of IPs.
2. Efficient routing in large systems: log (N) hops, with N number of total nodes.
3. Self organization , that is it handles node arrival, departure as well as removal.

3.1.1 Theory:-

In computing, Chord is a protocol and algorithm for a peer to peer distributed hash table. A distributed hash table stores key value pairs by assigning keys to different computers, a node will store the values for all keys for which it is responsible.

They specify how keys are assigned to nodes and how a node can discover the value for a given key by first locating the node responsible for that key.
It is one of the four original distributed hash table protocols, along with CAN, Tapestry, and Pastry. It was introduced in 2001 by Ion Stoica, Robert Morris, David Karger and Hari Balakrishnan, and was developed at MIT.

Nodes and keys are assigned an m-bit identifier using consistent hashing. The **SHA-1** algorithm is the base hashing function for consistent hashing. Consistent hashing is integral to the robustness and performance of Chord because both keys and nodes are uniformly distributed in the same identifier space with a negligible possibility of collision. Thus, it also allows nodes to join and leave the network without disruption. In the protocol, the term node is used to refer to both a node itself and its identifier (ID) without ambiguity. So is the term key. Using the Chord lookup protocol, nodes and keys are arranged in an identifier circle that has at most 2^{m} nodes, ranging from 0 to 2^{m}-1. Each node has a successor and a predecessor.
The successor to a node is the next node in the identifier circle in a clockwise direction. The predecessor is counter-clockwise. If there is a node for each possible ID, the successor of node 0 is node 1, and the predecessor of node 0 is node 2^{m}-1; however, normally there are "holes" in the sequence. The concept of successor can be used for keys as well. The successor node of a key k is the first node whose ID equals to k or follows k in the identifier circle, denoted by successor (k). Every key is assigned to its successor node, so looking up a key k is to query successor (k). Since the successor (or predecessor) of a node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, i.e., the r nodes preceding it and the r nodes following it. This list results in a high probability that a node is able to correctly locate its successor or predecessor, even if the network in question suffers from a high failure rate.

3.1.2 Potential uses Of Chords:-

Cooperative Mirroring: A load balancing mechanism by a local network hosting information available to computers outside of the local network. This scheme could allow developers to balance the load between many computers instead of a central server to ensure availability of their product.

Time-shared storage: In a network, once a computer joins the network its available data is distributed throughout the network for retrieval when that computer disconnects from the network. As well as other computers' data is sent to the computer in question for offline retrieval when they are no longer connected to the network. Mainly for nodes without the ability to connect full-time to the network.

Distributed Indices: Retrieval of files over the network within a searchable database. e.g. P2P file transfer clients.

Large scale combinatorial searches: Keys being candidate solutions to a problem and each key mapping to the node, or computer, that is responsible for evaluating them as a solution or not. Eg Code Breaking.

Flexible naming: Chord places no constraints on the structure of the keys it looks up: the Chord key-space is flat. This gives applications a large amount of flexibility in how they map their own names to Chord keys.

3.1.3 SHA Algorithm:-

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST. SHA-1 is considered insecure against well-funded opponents, and it is recommended to use SHA-2 or SHA-3 instead. SHA-1 produces a 160-bit (20 byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. SHA-1 is a member of the Secure Hash Algorithm family. The four SHA algorithms are structured differently and are named SHA-0, SHA-1, SHA-2, and SHA-3. SHA-0 is the original version of the 160-bit hash function published in 1993 and later versions came after SHA-0.

3.1.4 Joins, Leaves and Maintenance:-

To join the network, a node n performs a lookup for its own id through some first contact in the network and inserts itself in the ring between its successor s and the predecessor of s using a periodic stabilization algorithm. Initialization of n's routing table is done by copying the routing table of s or letting s lookup each required edge of n. The subset of nodes that need to adjust their tables to reflect the presence of n, will eventually do that because all nodes run a stabilization algorithm that periodically goes through the routing table and looks up the value of each edge.

The last task is transfer part of the items stored at s, namely items with id less than o equal to n need to be transferred to n and that is also handled by the application layers of n and s.

Graceful removals (leaves) are done by first transferring all items to the successor and informing the predecessor and successor. The rest of the fingers are corrected by the virtue of the stabilization algorithm.

3.1.5 Replication and Fault Tolerance:

Ungraceful failures have two negative effects:-

First, ungraceful failures of nodes cause loss of items. Second, part of the ring is disconnected leading to the inability of looking up certain identifiers.

Let alone if a set of adjacent nodes fail simultaneously. Chord tackles this problem by letting each node keep a list of the log2 (N) nodes that follow it on the circle. The list serves two purposes:-

First, if a node detects that its successor is dead, it replaces it with the next entry in its successor list. Second, all the items stored at a certain node are also replicated on the nodes in the successor list. For an item to be lost or the ring to be disconnected, log 2(N) + 1 successive nodes have to fail simultaneously.

3.2 Load balancing technique-:

Hi-Glob (Histogram-based Global Load Balancing) -:

In a peer to peer system with Hi-Glob load balancing technique each node ‘p’ has two key components. The first component is a histogram manager that maintains a histogram that reflects a global view of the distribution of the load in the system. The histogram stores statistical information that characterizes the average load of no overlapping groups of nodes in the P2P network. The second component of the system is a load balancing manager that takes actions to redistribute the load whenever a node becomes overloaded or under-loaded. The load-balancing manager may redistribute the load both statically when a new node joins the system and dynamically when an existing node in the system becomes overloaded or under-loaded.

The histogram information can be used for two purposes:

1. To determine whether the node is normally loaded, overloaded or under loaded.

2. To facilitate the discovery of a lightly loaded node or a heavily loaded node at the time of load balancing.

There are five modules in HiGLOB:

1. Create Histogram
2. Send & get Request
3. Calculate Load
4. Load Balance
5. Send & get Response

MODULE DESCRIPTION:

1. Create Histogram

In this module, network is spited in to non-overlapping regions. Histogram Manager will maintain histogram for each peer. In that histogram, that statistical information of loads will be stored. Histogram of each peer contains the load values of neighbor peers.

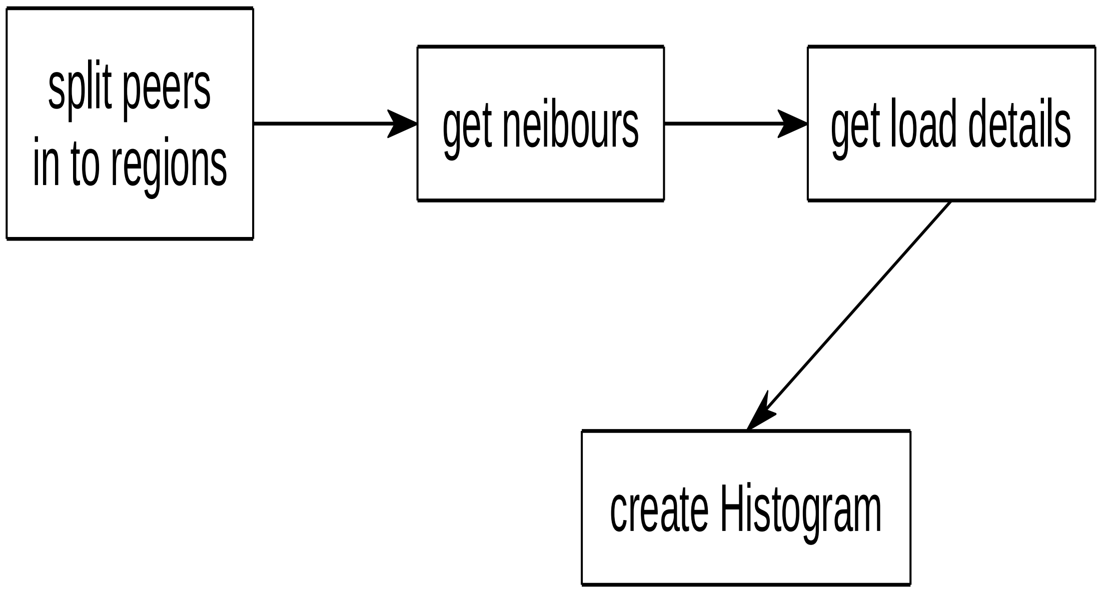


Figure 7

2. Send & get Request

Using this module, each peer can request actions to another peer. The request may be like zip, unzip, compress, decompress and converting to document. The peer will get the request.

This request will be considered as load. Before processing the request the load of the peer will be calculated.

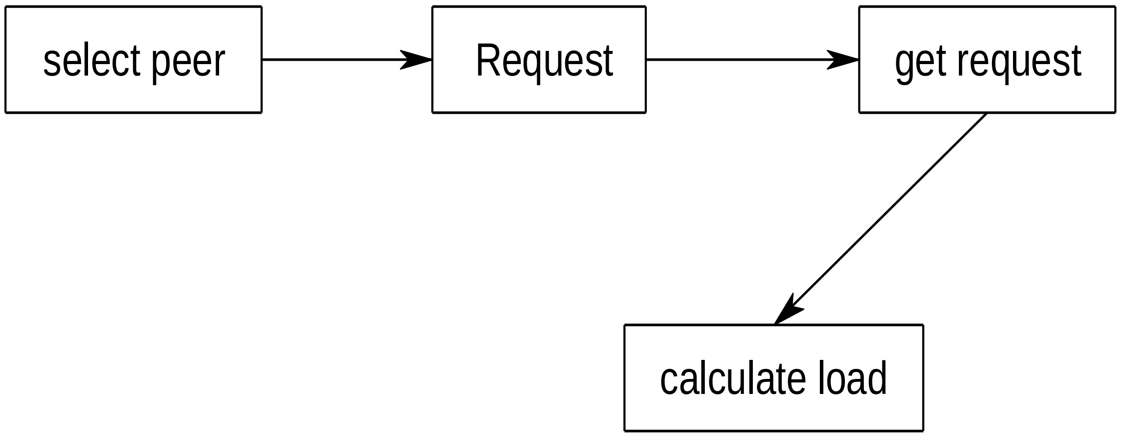


Figure 8

3. Calculate load

The load of the peer will be calculated before processing the request. The individual load of the peer will be calculated first. Then the average load the region will be calculated. If the individual load is greater than 2 times of average load then that peer will be considered as overloaded. Else the peer will be considered as normally loaded peer. After calculated load the load factor will be calculated. Load factor is the ratio between the previous load and current load. If the load factor is greater than 1 the histogram will be updated. To reduce the histogram maintenance cost, the histogram no needs to be updated for small changes in loads. So if the load factor is less than 1 the histogram will not be updated.

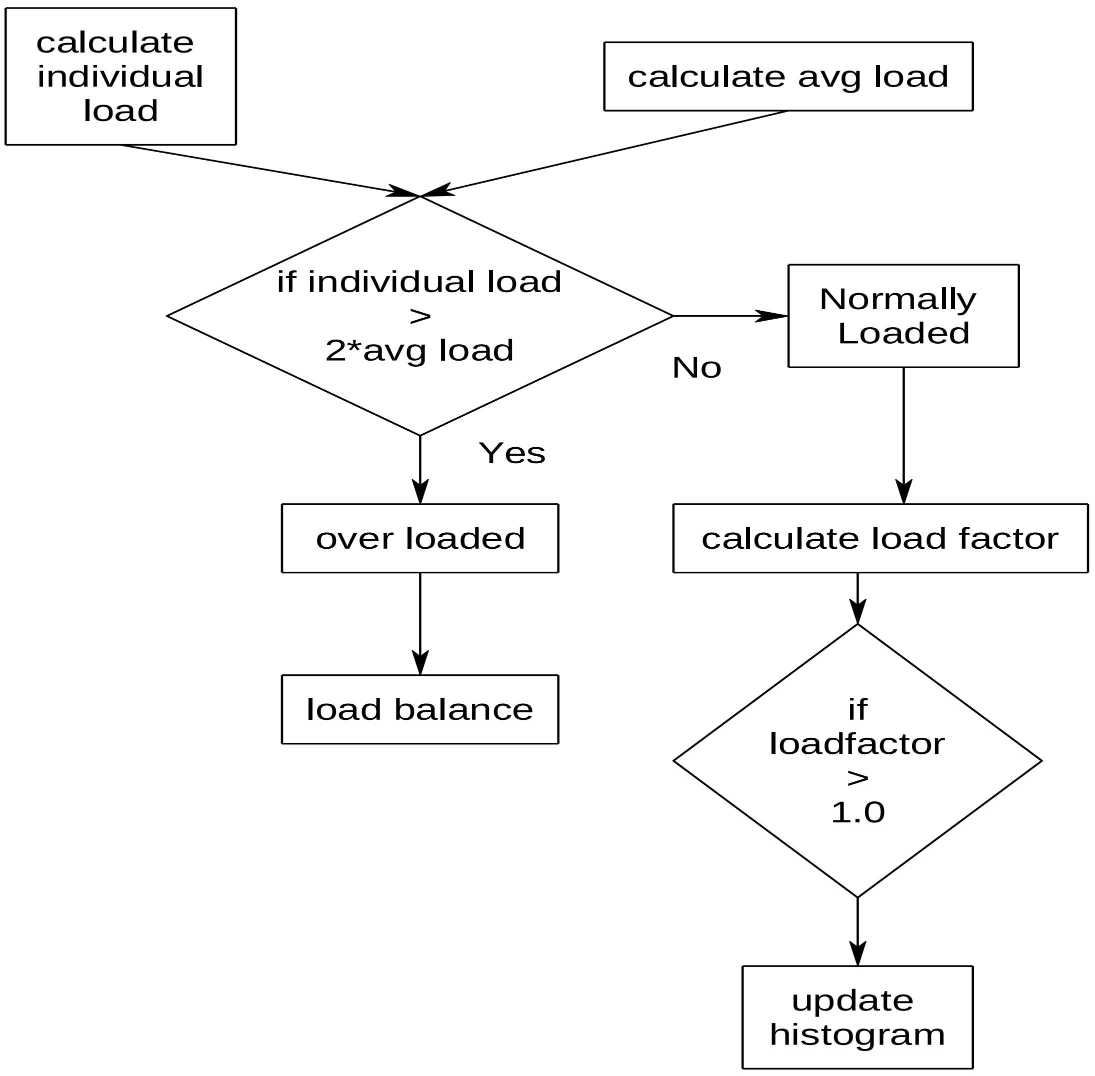


Figure 9

4. Load Balance

If the peer is normally loaded, the load will be handled by the same peer. Else if it is overloaded the number of peers in the region will be counted. If there is only one peer the load will be shared that peer. If the number of peers are even the peers will be split as pair of peers, and the load is shared to the pair which is having low load. If the number of peers is odd then the peers will be split as pair of peers and there will be one more odd peer. The load will be shared to either peer pair or to the odd peer, which is having less load.

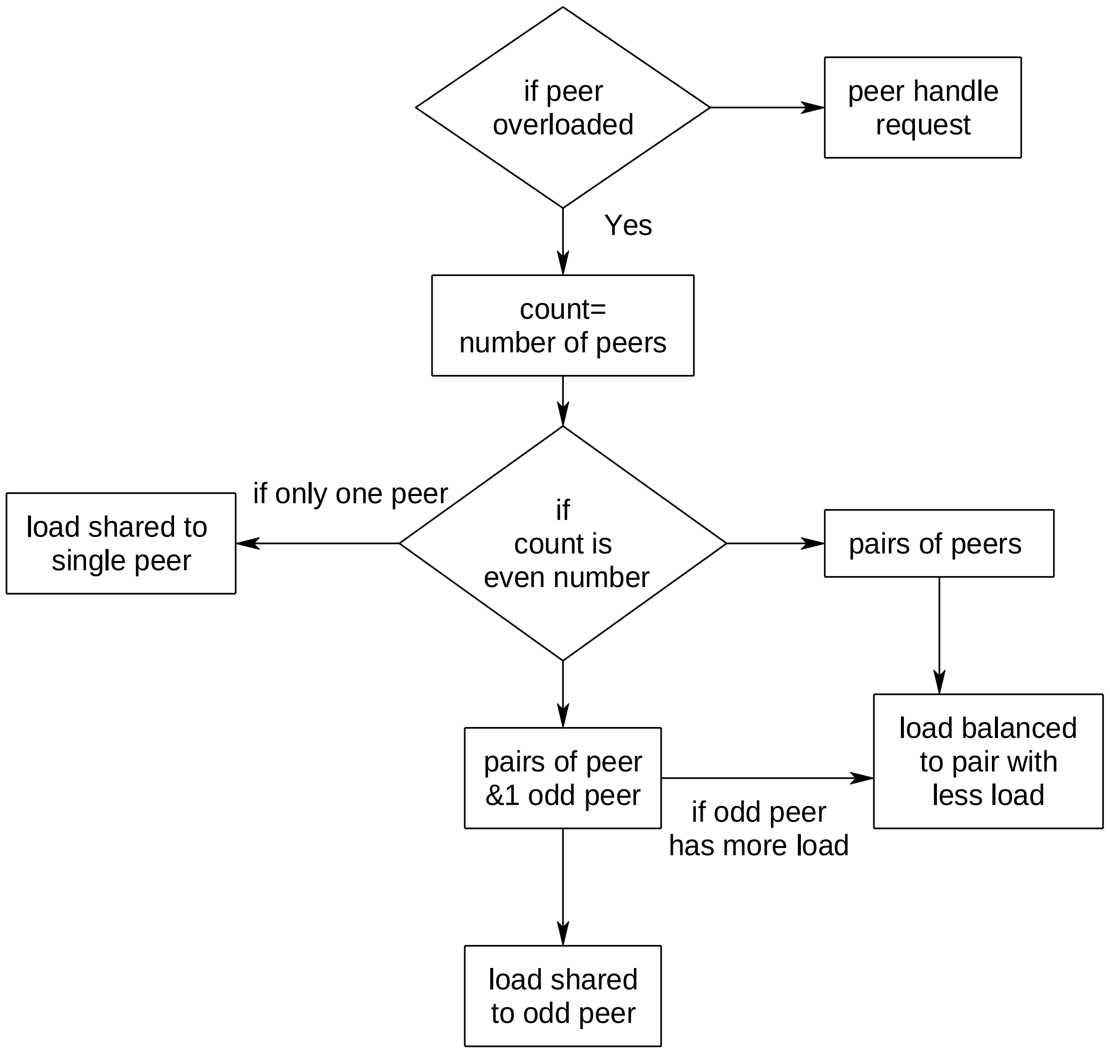


Figure 10

5. Send & get Response:

The response will be send to the peer. The response may be like decompressed file, compressed file, document, zipped folder or unzipped files. The response will be saved in the peer who sent request.

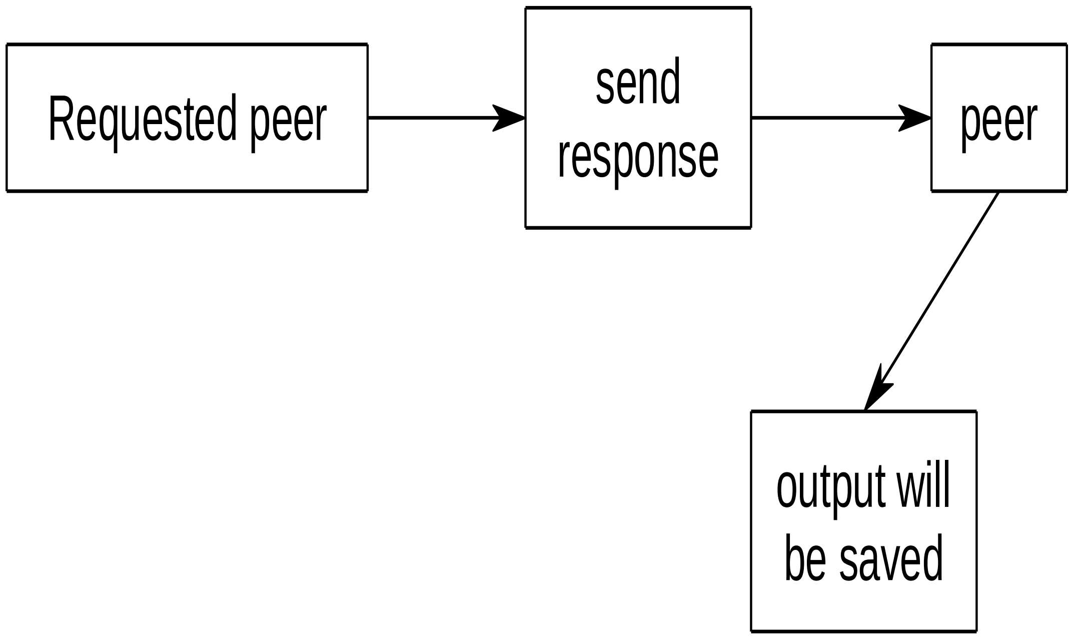


Figure 11

System Architecture :

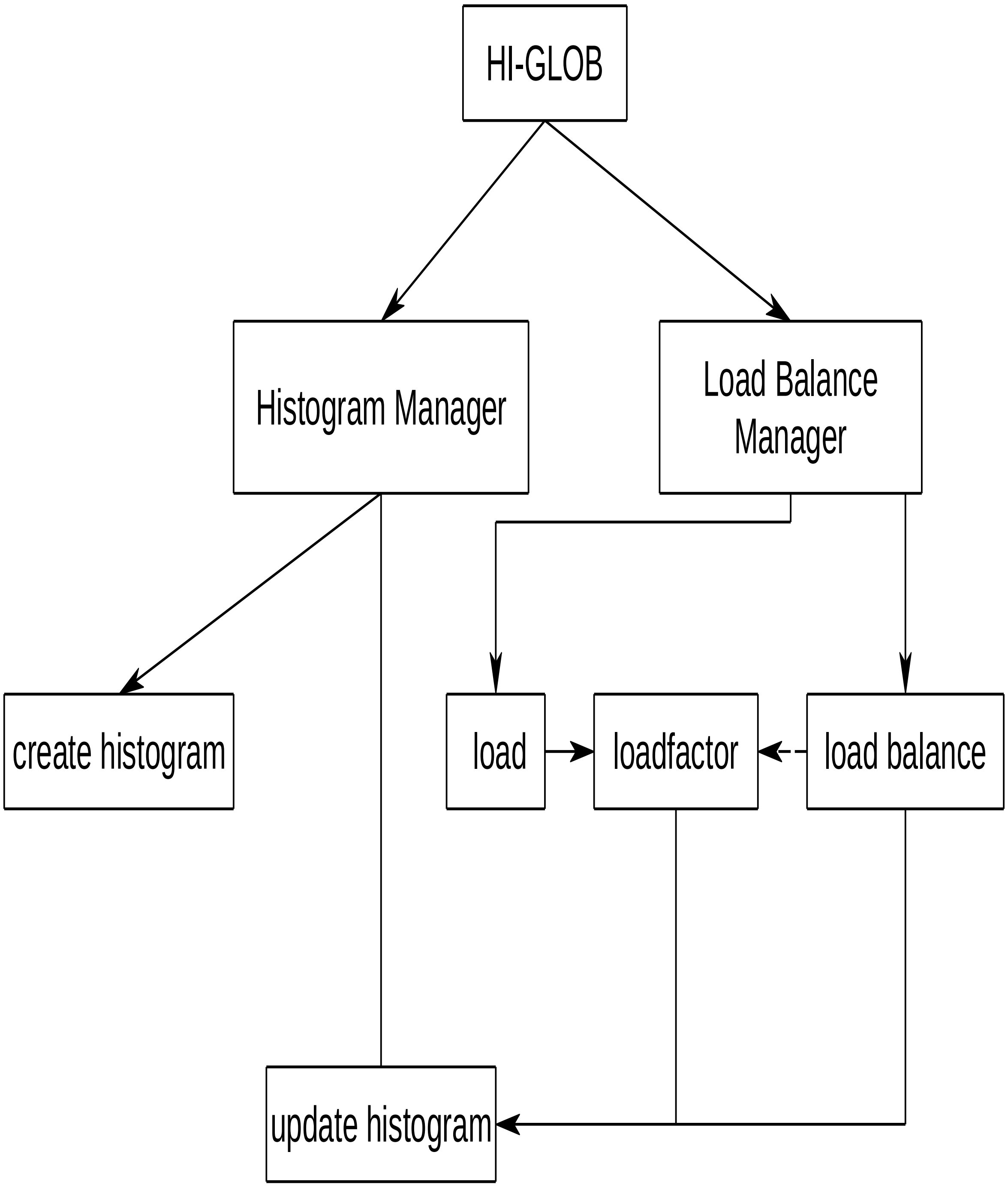


Figure 12

Data Flow Diagram:

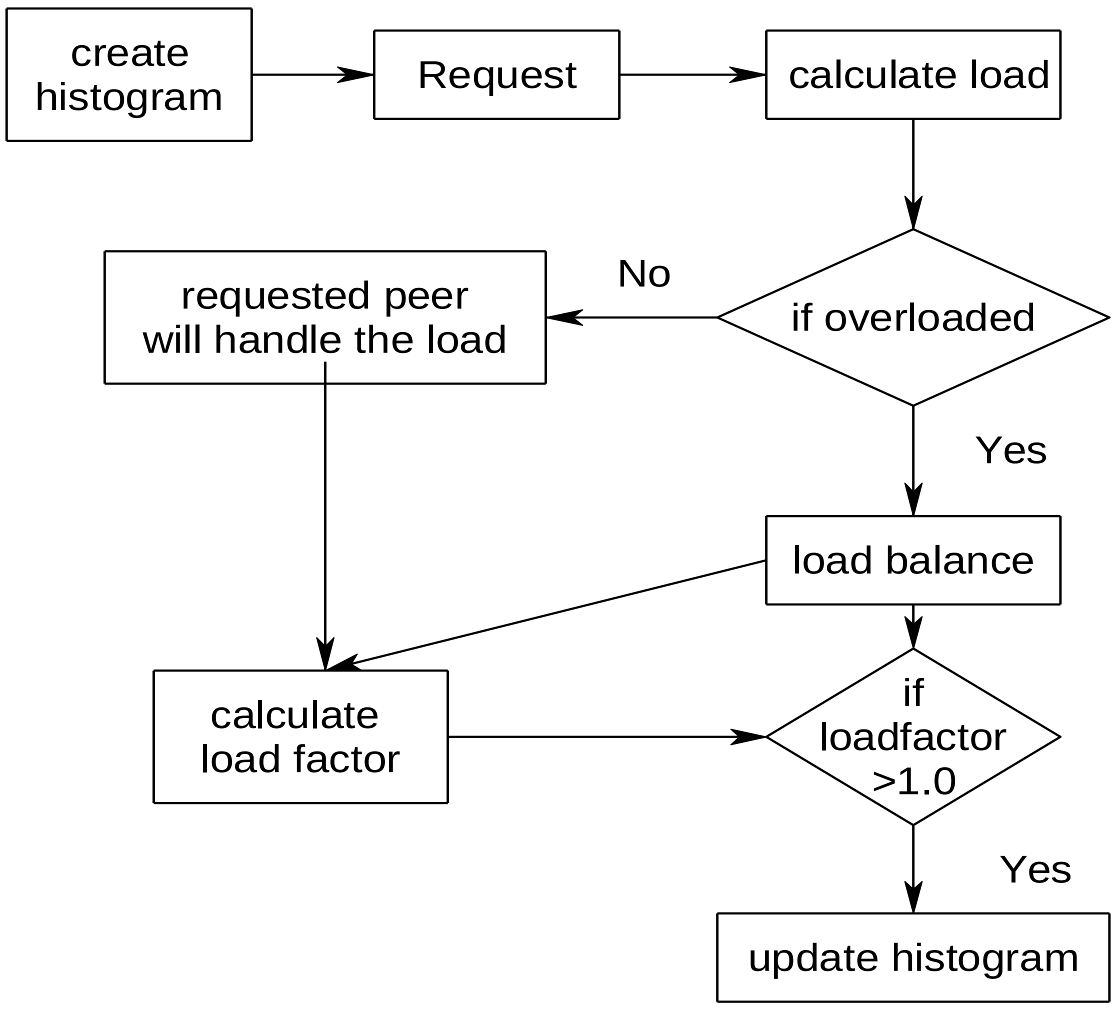


Figure 13

UML Diagram:

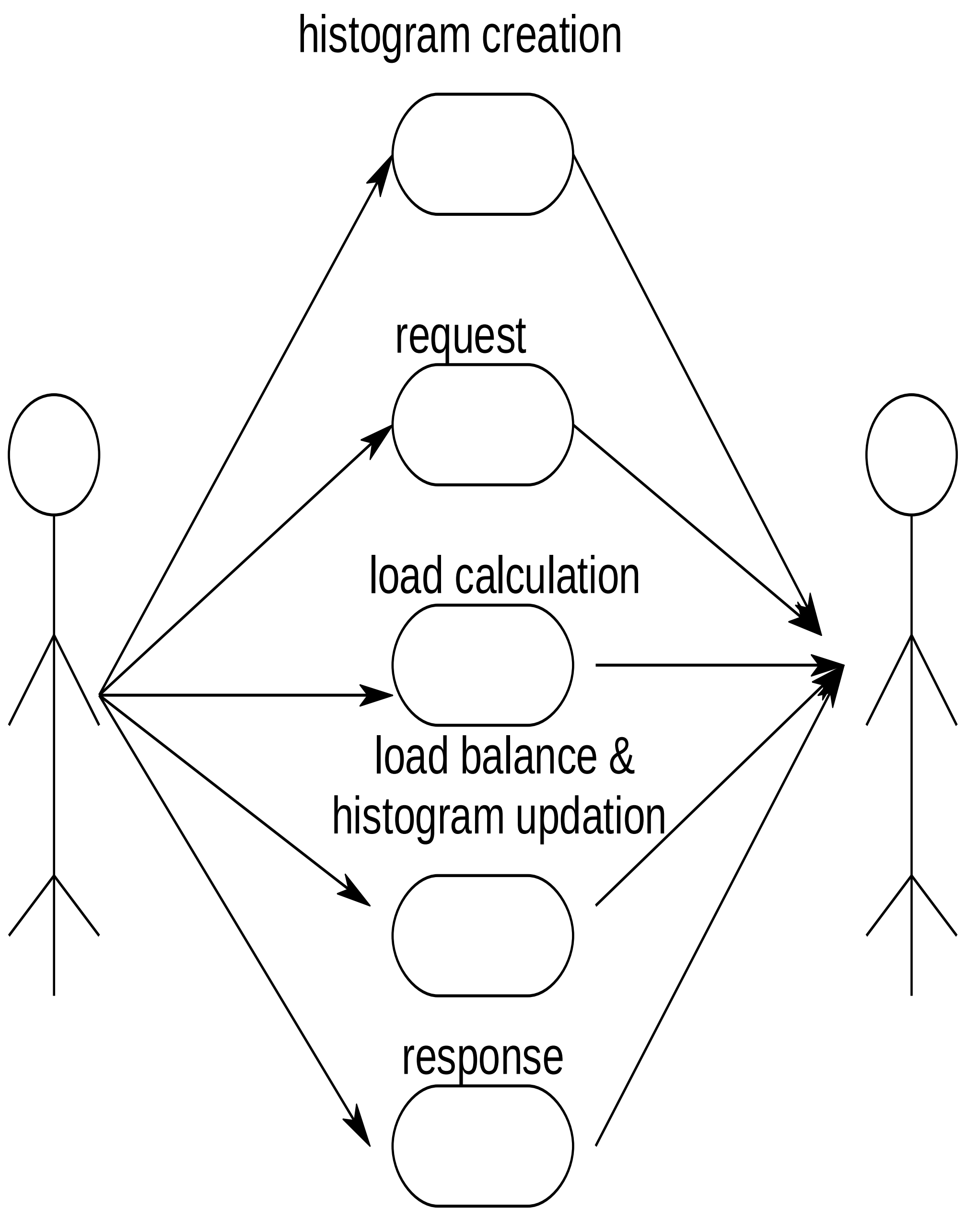


Figure 14

Activity Diagram:

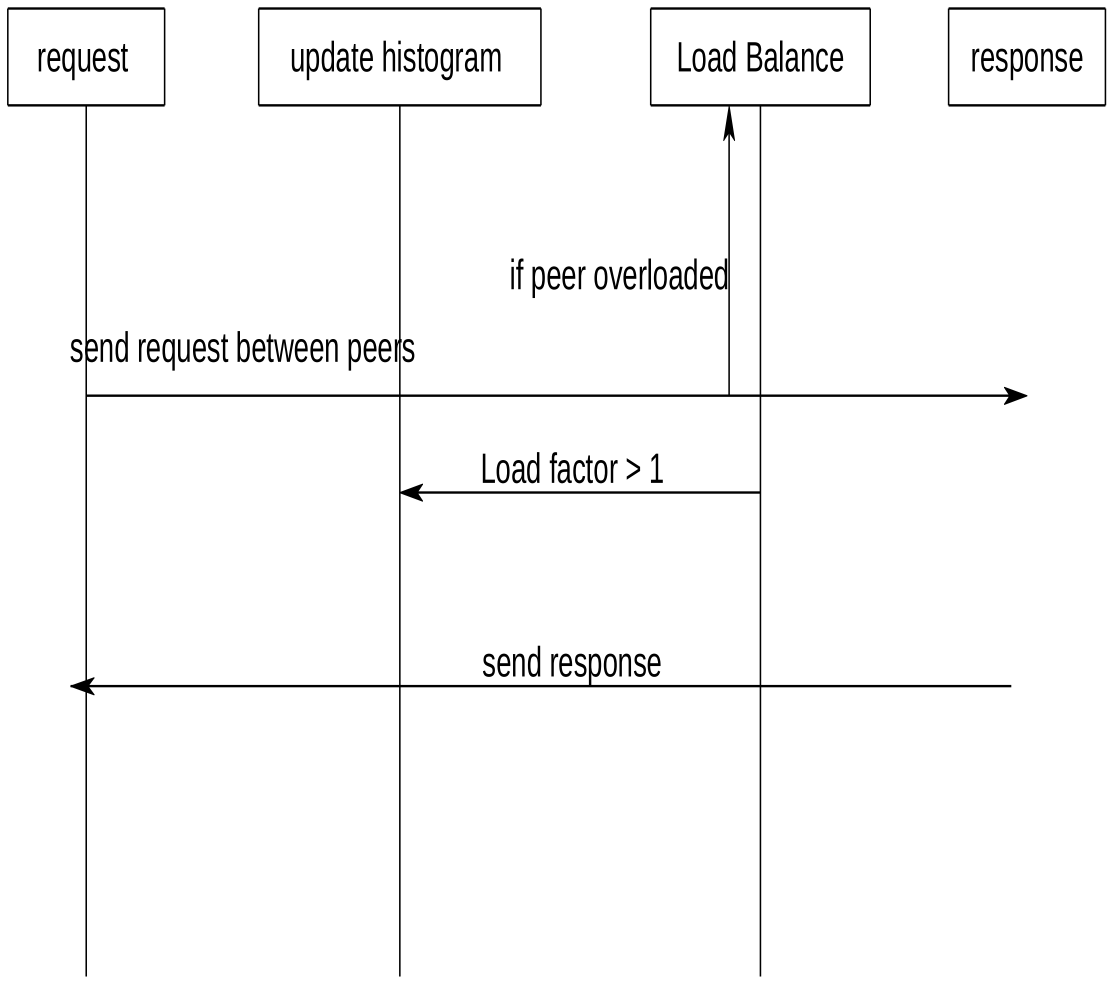


Figure 15

Chapter 4: PERFORMANCE ANALYSIS

Algorithm for Chord implementation:-

```
// ask node n to find the successor of id
n. find_successor(id)
// It is a half closed interval.
if (id ? (n, successor] )
return successor,
else
// forward the query around the circle
n0 = closest_preceding_node(id);
return n0.find_successor(id);
// search the local table for the highest predecessor of id
 n.closest_preceding_node(id)
 for i = m downto 1
 if (finger[i]?(n,id))
 return finger[i];
 return n;
```

The pseudocode to stabilize the chord ring/circle after node joins and departures is as follows:

```
 // create a new Chord ring.
 n.create()
 predecessor = nil;
 successor = n;
 // join a Chord ring containing node n'.
 n.join(n')
 predecessor = nil;
 successor = n'.find_successor(n);
 // called periodically. n asks the successor
 // about its predecessor, verifies if n's immediate
 // successor is consistent, and tells the successor about n
 n.stabilize()
 x = successor.predecessor;
 if (x?(n, successor))
 successor = x;
 successor.notify(n);
 // n' thinks it might be our predecessor.
 n.notify(n')
 if (predecessor is nil or n'?(predecessor, n))
 predecessor = n';
```

```
// called periodically. refreshes finger table entries.
// next stores the index of the finger to fix
n.fix_fingers()
next = next + 1;
if (next > m)
next = 1;
finger[next] = find_successor(n+2^{next-1});
// called periodically. checks whether predecessor has failed.
n.check_predecessor()
if (predecessor has failed)
 predecessor = nil;
```

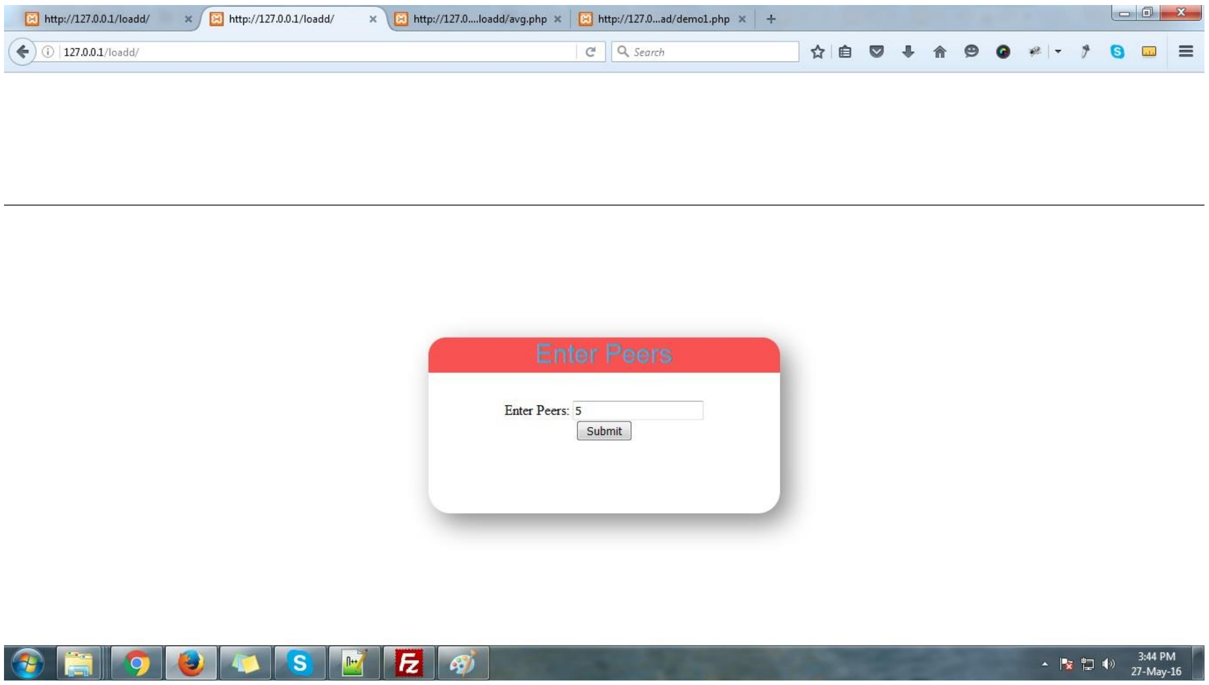
HiGLob technique Alogrithm

- Split peers into Non overlapping regions

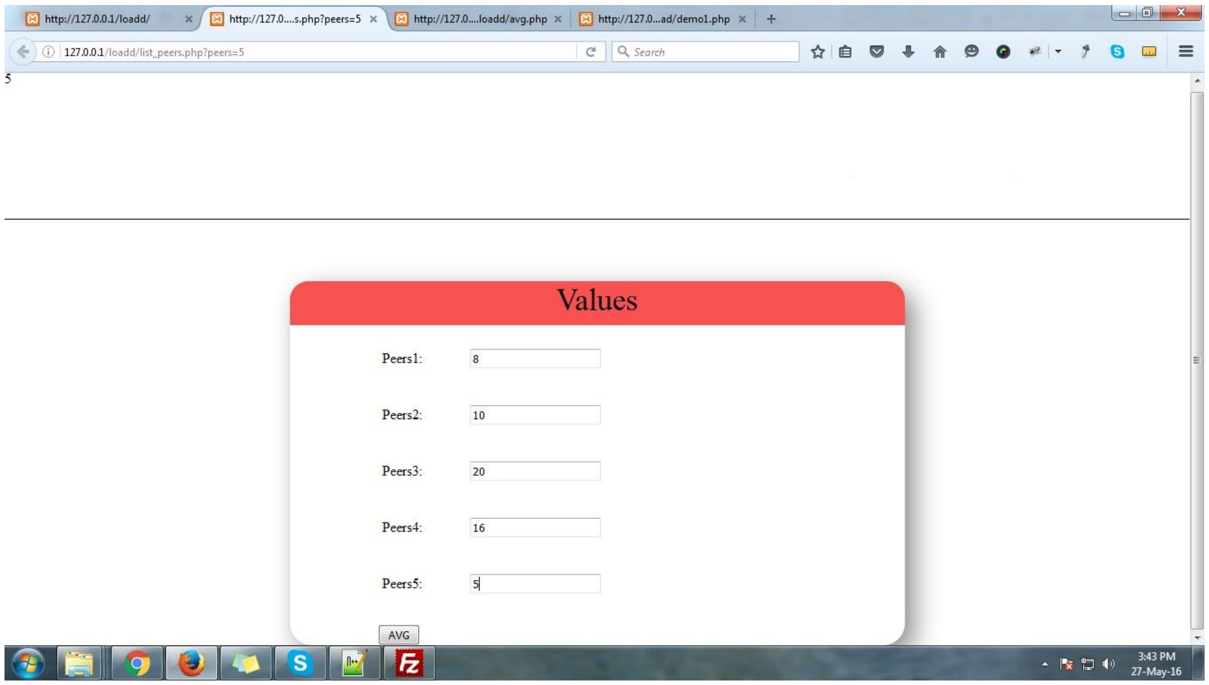
```
get neighbours ()
get load details()
create Histogram()

Select peer()
get request()
Calculate load()

If load > 2*average load
Balance load()
update Histogram()

Balance load()
if no of peers= even
divide it into pair of peers
Load balanced to pair with less load()

If no of peers= odd
divide it into pair of peers and 1 odd peer
```

```
Load shared to odd peer()
if odd peer has more load
Load balanced to pair with less load()

if no of peers=1
Load shared to single peer();
send response()
select peer()
save output().
```

Snapshots are as Follows:



Enter Peers: 5

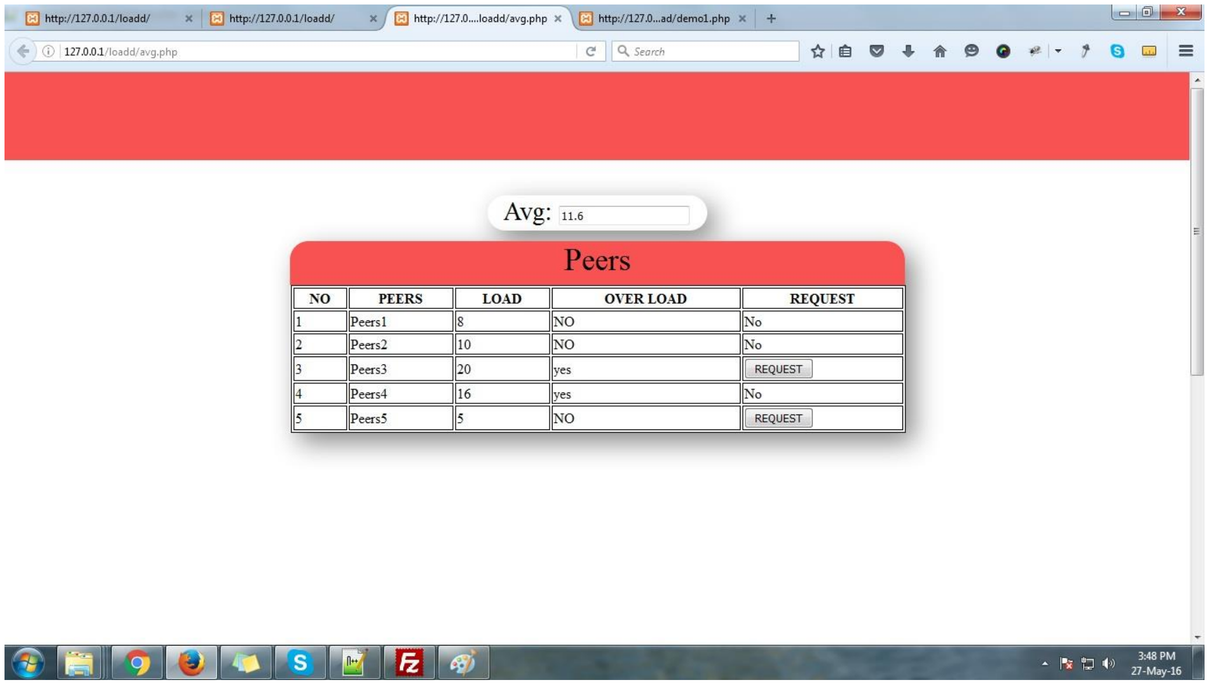

http://127.0.0.1/loadd/
http://127.0....s.php?peers=5
http://127.0....loadd/avg.php
http://127.0...ad/demo1.php
127.0.0.1/loadd/list_peers.php?peers=5
Search
3:43 PM
27-May-16

5

Values

Peers1:	8
Peers2:	10
Peers3:	20
Peers4:	16
Peers5:	5

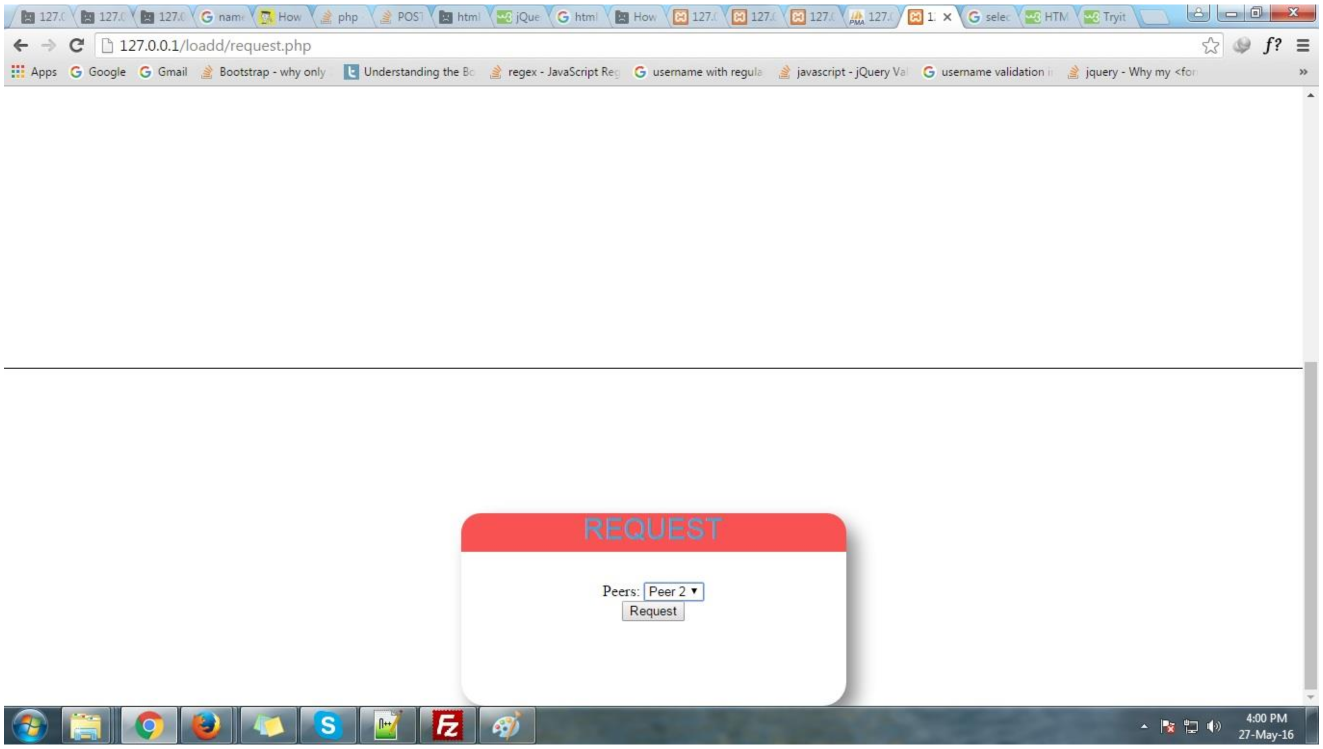
AVG



Avg: 11.6

Peers

NO	PEERS	LOAD	OVER LOAD	REQUEST
1	Peers1	8	NO	No
2	Peers2	10	NO	No
3	Peers3	20	yes	REQUEST
4	Peers4	16	yes	No
5	Peers5	5	NO	REQUEST


127.0.0.1/loadd/request.php
REQUEST
Peers: Peer 2
Request

Chapter 5: CONCLUSION

Load balancing is an important technique for improving distributed system performance by considering the group of hosts in the system to share their workloads. This results in a better utilization of host resources, a high system throughput and quick response time of user requests. In load sharing, the incoming client requests are evenly distributed among the participating hosts. The load on an overloaded host is transferred to an under loaded host to enhance the system throughput. Effective load balancing among the group of hosts in a distributed system relies on accurate knowledge of the state of the individual host. This knowledge is used to judiciously assign incoming computational tasks to appropriate hosts, by using specific load distribution policies so that the requests can be processed quickly. This is achieved by allowing the hosts cooperatively monitor the global system load and distribute / re-distribute the load according to the load sharing algorithms which can be static or dynamic and can have either centralized or distributed control.
For solving the load balance problem in a structured P2P system the first main contribution is an in-depth analysis of the effect of capacity and workload heterogeneity on algorithm performance in both static and dynamic environments and the qualitative relationship between static and dynamic environments. Our system is very adequate if the goals of the project are static in nature or do not change very often. Efforts are been made to have client interaction and include a dynamic nature of goals in the system development phase. We have successfully completed all the desired framework components as set at the beginning of the project.

We proposed a framework, HiGLOB, to enable global load balance for structured P2P systems. Each node in HiGLOB maintains the load information of nodes in the systems using histograms. This enables the system to have a global view of the load distribution and hence facilitates global load balancing. We partition the system into non overlapping groups of nodes and maintain the average load of them in the histogram at a node.

Future Work:

In this report, proposed work presents existing techniques of handling load of servers in peer to peer networks. In future other parameters can be modified to achieve greater utilization and less power consumption.

Balancing multiple resources in P2P networks as we have assumed that there is only one bottleneck of resources to be balanced for P2P networks. However, a system may be constrained w.r.t. multiple resources. To investigate the following important issues in the future is needed: It is intended to explore other cost reducing neighborhoods. Also to perform a more in-depth study of issues in the dynamic scenario in which a node joins and leaves the system.

REFERENCES

[1] S3: Scalable, Shareable and Secure P2P Based Data Management System http://www.comp.nus.edu.sg/~s3p2p, 2008.

[2] Gnutella, http://www.gnutella.com/, 2008.

[3] Bit Torrent, http://www.bittorrent.com/, 2008.

[4] Overnet, http://www.overnet.com, 2008.

[5] SETI@home, http://setiathome.berkeley.edu/, 2008.

[6] Groove, http://www.groove.net, 2008.

[7] Skype, http://www.skype.com/, 2008.

[8] A. Madhukar and C. Williamson, "A Longitudinal Study of P2PTraffic Classification," Proc. Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS), 2006.

[9] D. Karger, F. Kaashoek, I. Stoica, R. Morris, and H. Balakrishnan,"Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. SIGCOMM '01, pp. 149-160, 2001.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," Proc. SIGCOMM '01,pp. 161-172, 2001.

[11] Stoica, I., Morris, R., Karger, D., a. o.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: *Proc. of ACM Sigcomm, San Diego, CA* 2001.

[12] Rao, A., Lakshminarayanan, K., Surana, S., a. o.: Load Balancing in Structured P2P Systems. In: *Proc. of* 2*nd Intern. Workshop on P2P Systems*. Berkeley. February 2003.

[13] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," Proc. ACM Symp. Parallelism in Algorithms and Architectures (SPAA), 2004.