An Efficient Client- Server Assignment for Internet Distributed systems

A Thesis Submitted in Partial Fulfillment of the Requirements for

The Degree of

Bachelor of Technology

In

Computer Science and Engineering

ABDUL HADI SHARIFI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA 2015

An Efficient Client- Server Assignment for Internet Distributed Systems

A thesis

Submitted by

Abdul Hadi Sharifi

(111CS0561)

In Partial Fulfillment of the Requirements for

The Degree of

Bachelor of Technology

In

Computer Science and Engineering

Under The Guidance of

Prof. P. M. Khilar

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Orissa - 769008, India

2015

**NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA,
ORISSA-769008, INDIA**

CERTIFICATE

This is to certify that the thesis entitled, **" An Efficient Client- Server Assignment for Internet Distributed Systems"** submitted by **Abdul Hadi Sharifi** in partial fulfillment of the requirement for the award of **Bachelor of Technology degree in Computer Science and Engineering** at the National Institute of Technology Rourkela is an authentic work carried out by him under our supervision and guidance. To the best of our knowledge, the matter embodied in the thesis has not been submitted to any other University/ Institute for the award of any degree or diploma.

Place: Rourkela   Research Guide

Date: Dr. P. M. Khilar

                                                                   Professor

                                                                   National Institute of

                                                                   Technology, Rourkela

ACKNOWLEDGEMENTS

Abstract

The World Wide Web is used by millions of people every day for various purposes including email, reading news, downloading music, online shopping or simply accessing information about anything. Using a standard web browser, the user can access information stored on Web servers situated anywhere on the globe. This gives the illusion that all this information is situated locally on the user's computer. In reality, Internet is a collection of several distributed systems consisting of various clients and servers. These clients communicate with each other with the help of transitional servers. As this field is the most useful area of the computer science, optimizing on the whole performance of such a system then can be formulated as a client server assignment problem whose aim is to allocate the clients to the servers in such a way to satisfy some pre-specified necessities on the communication cost and load balancing.

Servers recover from failures and get back information needed. It provides fault tolerance by doing work in the background and during client operations that are rare.

I propose an approach based on an algorithm which obtains better efficiency than the existing client server assignment and load balancing.

Table of Contents

Chapter 1
Introduction

1.1   Overview

An Internet distributed system consists of a number of nodes (e.g., computers) that are linked together in ways that allow them to share resources and computation. An ideal distributed system is completely decentralized, and that every node is given equal responsibility and no node is more computational or resource powerful than any other. However, for many real world applications, such a system often has a low performance due to a significant cost of coordinating the nodes in a completely distributed manner. In practice, a typical distributed system consists of a mix of servers and clients. The servers are more computational and resource powerful than the clients. Typical examples of such systems are e-mail, instant messaging, e-commerce, etc. For sending a mail from node A to another node B, the data first flows to email server of node A. Then the data flows towards the email server of node B and finally it reaches node B. Hence, it is the responsibility of email servers to send receives emails for the clients assigned to it. Clients do not communicate with each other directly. Servers communicate on behalf of their clients. Similar kind of process can be used for instant messaging services and ecommerce services.

Client server assignment can be designed based on the following observations:

1. If two clients are assigned to the same server, the server will receive message from one client and forward to another one. If they are on different servers, the sender client first sends to its server. The sender's server will receive data and forward it to the receiver's server. The receiver server will receive data and forward it to receiver client. Thus the total communication between two frequently interacting clients increases if they are assigned to two different servers. Assigning these two clients to a single server makes all information exchange locally. It is more efficient to have all the clients assigned to few servers to minimize total communication.

2. On the contrary having fewer servers, results in those servers being heavily loaded while others are left underutilized. If a server is heavily loaded it results in low performance due to excessive resource usage on that server. Thus it is necessary to consider load balance on the server while assigning clients. From the above observations it is clear that total communication load and load balancing are two contradicting features. Hence equilibrium must be maintained between overall communication load and load balancing of the servers.

A classical example of such systems is Email. When a client A sends an email to another client B, A does not send the email directly to B. Instead, A sends its message to its email server which has been previously assigned to handle all the emails to and from A. This server relays A's email to another server which has been previously assigned to handle emails for B. B then reads A's email by downloading it from its local server. Importantly, the email servers communicate with each other on behalf of their clients. The main advantage of this architecture is specialization, in the sense that the powerful dedicated email servers release their clients from the responsibility associated with many tasks including processing and storing emails, and thus making email applications more scalable.

(a) A and B are connected
    to the same server

(b) A and B areconnected
    to two different servers
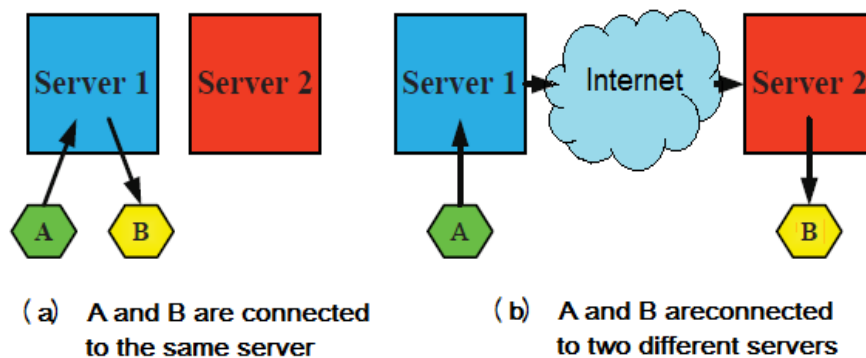
An example of client assignment to servers

Fig. 1

1.2　Client - Server

The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

The client–server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer.

In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Clients and servers exchange messages in a request-response messaging pattern: The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language,

and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol.

1.3 Internet Distributed Systems

The Internet is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to link several billion devices worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries an extensive range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), the infrastructure to support email, and peer-to-peer networks for file sharing and telephony.

1.4 Motivation

➢ Internet is the largest distributed system in the world
➢ Internet usage is growing fast in Asia
➢ Load balancing in Internet is achieved by efficient assignment of clients to servers.

1.5 Project organization

Chapter one of this thesis gives a wide introduction over the client-server and Internet distributed system along with the motivations, chapter 2 provides the literature, in chapter 3 the proposed algorithm and the result is being discussed, chapter 4 discusses one the most essential part of the work which is fault tolerance of the system and finally chapter 5 cocludes the work and ends with bibliography.

Chapter 2

Literature Review

2.1 The Normalized cut Algorithm

The normalized cut algorithm [1] can cluster a set of elements based only on the values of a similarity measure between all possible pairs of elements. The approach is a spectral clustering method. It is based on the properties of eigenvectors from a matrix computed using the similarities between each pairs of elements.

In the normalized cut scheme, the clustering is seen as a graph partitioning problem. The nodes of the graph are the elements and the weights on the graph edges connecting two nodes are the similarities measured between the two corresponding elements. We use the similarity measure described in the previous section. We seek to partition the graph into sub graphs with high similarities between the nodes of the same sub graphs and low similarities between nodes from different sub graphs.

Let **G** be that graph with nodes **V** and an adjacent graph weight matrix **W**. The element **W(u,v)** of **W** is the similarity measured between the elements represented by the nodes **u** and **v**. We can break **G** into two disjoint sets **A** and **B** so that **A** ∪ **B** = **V** and **A** ∩ **B** = **Ø** by simply removing the edges connecting the two parts. The cost of removing those edges is computed as the total weight of the edges that have been removed:

$$cut(A, B) = \sum_{u \in A, v \in B} W(u, v)$$

This cost is called a "cut" in the graph theory language. We want to minimize this cut value when partitioning **G**.

However, minimizing the cut value encourages cutting isolated nodes in **G**. To avoid that problem, the cost of removing edges from **A** to **B** is considered as a fraction of the total weight of connections from nodes in **A** to all nodes in the graph. This leads to a new cost measure called the normalized cut:

$$Ncut(A, B) = \frac{cut(A, B)}{asso(A, V)} + \frac{cut(A, B)}{asso(B, V)}$$

Where **asso (A, V)** is the sum of **{u in A, v in V} W(u,v)** is the total connection weight from all the nodes in **A** to all the nodes in **V**. **asso(B,V)** is defined in a similar way. [1]
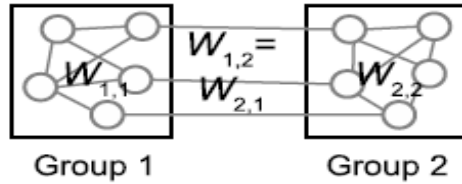
Example of Normalized Cuts

Fig. 2

In [2] it is shown that the optimal client server assignment for pre-specified requirements on total communication load and load balancing is NP-hard. A heuristic algorithm based on relaxed convex optimization is used for finding the approximate solution to the client server assignment problem.

2.2   Similar Research Areas

Distributed Interactive applications (DIAs): DIAs are networked systems that allow multiple participants at different locations to interact with each other. Wide spread of client locations in large-scale DIAs often requires geographical distribution of servers to meet the latency requirements of the applications. In the distributed server architecture, client assignment to servers directly affects the network latency involved in the interactions between clients. [3] Focuses on client assignment problem for enhancing the interactivity performance

of DIAs. In this paper, the problem is formulated as a combinational optimization problem on graphs and proved to be NP-complete of assigning clients to appropriate servers. Three heuristic algorithms are proposed, namely, nearest assignment, Greedy assignment, Distributed greedy assignment and are compared and evaluated using real Internet latency data.

[4] is an improved version of [3] and the results show that the proposed algorithms, nearest Assignment algorithm, Modify Assignment and Distributed modify assignment, are efficient and effective in reducing the interaction time between clients. However, both [3] and [4] consider the client assignment in DIAs and do not consider the load on the server and other factors.

Load-distance balancing Problem (LDB): Client server assignment in [5] is modeled on the fact that the delay incurred by a client is dependent on load of the server and the distance to the assigned server. Delay on the client side is the sum of network delay (proportional to distance to its server) and congestion delay at the server. Hence it focuses on distance between the client and the server and the load on the server in client-server assignment. The problem is defined as Load-distance balancing (LDB) problem[6]. This paper targets 2 flavors of LDB. In the first flavor, the objective is to minimize the maximum incurred delay using an approximation algorithm and an optimal algorithm. In the second flavor, the objective is to minimize the average incurred delay, which the paper mentions it as NP-hard and provides a 2-approximation algorithm and exact algorithm.

However to the best of my knowledge there is no clustering algorithm that is designed as a Semi definite programming problem to achieve the goal: Load balancing on servers and minimizing the communication load between servers, for client-server assignment.

Chapter 3 Proposed Algorithm

3.1 Description of Algorithm

Algorithm for load balancing:

1. Maintain a matrix A_{N*N} which stores the rate of data exchanged between clients. Rows and columns represent the clients in system.

2. Maintain a matrix X_{N*M} denoting which client is using which servers for communication. 1 denotes client using the server and 0 denotes not using the server.

3. Maintain matrix L_{N*M} storing total load on server due to client.

4. Calculate total load on all the servers i.e. Fc.

5. Calculate approximate average load that should be given to each server i.e. (Fc/M).

6. Split the number of servers in two groups with m = [M/2] and M-m = [M/2] servers in each group.

7. First perform intra-group load balancing, then perform inter-group load balancing between servers in one group and server from another group.

3.2 Results

Implementing algorithm with an example by applying it on the graph in Fig. 3.

Input
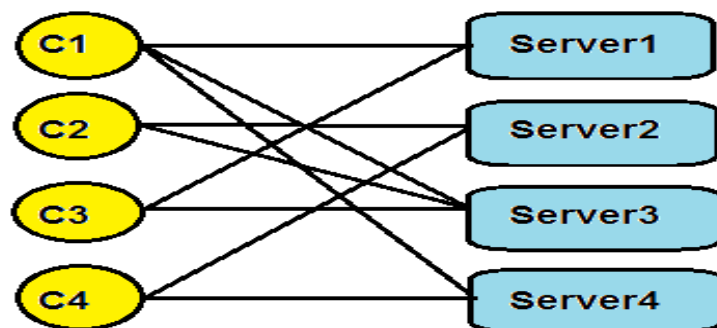
Graph connecting clients and servers

Fig. 3

a) Maintaining matrix A_{N*N} which indicates the rate of the data exchanged between different clients communicating to each other:

	C1	C2	C3	C4
$A_{N*N}=$ C1	0	8	4	4
C2	8	0	10	2
C3	4	9	0	0
C4	4	2	0	0

b) From the graph in fig. 3 maintaining a binary matrix X_{N*M} which denoting which client using which server (1 indicates the client using particular server and 0 not).

	S1	S2	S3	S4
$X_{N*M}=$ C1	1	0	1	1
C2	0	1	1	0
C3	1	0	1	0
C4	0	1	0	1

c) The current load on each server can be calculated from matrix A_{N*N} and binary matrix X_{N*M} and the result is matrix L_{N*M}

	S1	S2	S3	S4
$L_{N*M}=$ C1	6	0	6	4
C2	0	10	10	0
C3	7	0	6	0
C4	0	2	0	4

Load on server1 is the total load its handling from its clients (C1 and C3).

Load on server1 = 6+0+7+0 = 13

Similarly the total load on each individual server can be calculated.

Load on server2 = 0+10+0+2 =12

Load on server3 = 6+10+7+0 =22

Load on server3 = 4+0+0+4 =23
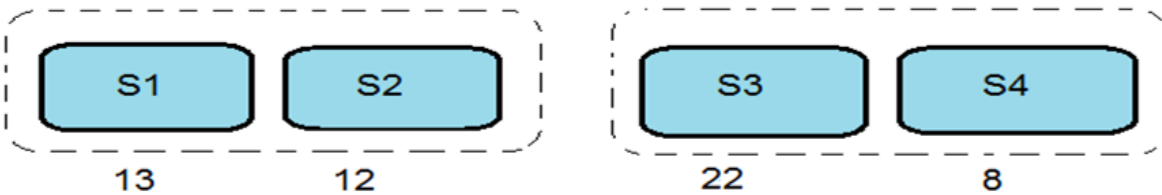
    d) Fc calculated by the total load on all the servers

Fc = 13 + 12 + 23 = 55

    e) The total load is then divided by the number of servers in the system i.e. 4.
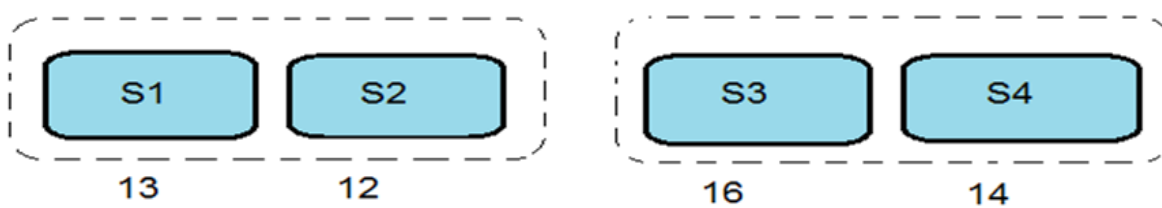
Approximate average load on each server = 55/4 = 13.75 ~ 14

Output

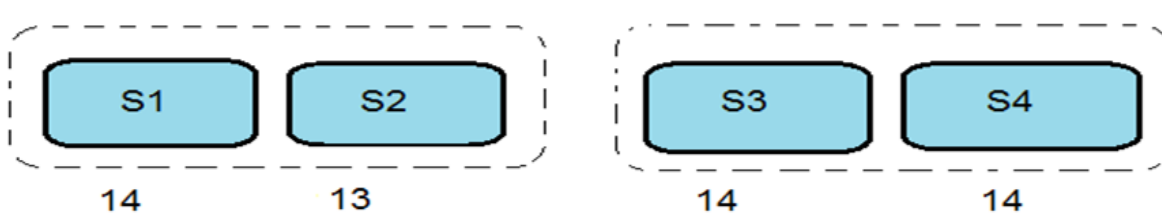Initial load on servers:

S1	S2		S3	S4
13	12		22	8

After Intra-group load balancing:

S1	S2		S3	S4
13	12		16	14

After Inter-group load balancing:

S1	S2		S3	S4
14	13		14	14

To get a real life example the bellow Matlab code calculates for an M number of servers and different M to N number of clients.

Assumptions for generating a real life example for verification of algorithm

1- Considering a matrix A as 160x160 at max

   The content of matrix A is being entered randomly

2- Considering matrix X as 160x10 at max (160 clients and 10 servers)

   Clients being randomly allotted to servers

3- Load matrix L will be generating with dimension 160x10

   Load matrix will be generated from A and X

Matlab Code

```
function sum=func(c)
X=rand(c,10) <0.5 ;
A=zeros(c);
for i=1:c
  for j=i:c
    temp=randi(2*c);
    if(temp>c)
      A(i,j)=0;
      A(j,i)=0;
    elseif(i==j)
      A(i,j)=0;
    else
      A(i,j)=floor(temp/10);
      A(j,i)=floor(temp/10);
    end
  end
end

for i=1:c
  count=0;
  sum=0;
  for ab=1:c
    sum=sum+A(i,ab);
  end
```

```
        for j=1:10
            if(X(i,j)==1)
                count=count+1;
            end
        end
        prop=zeros(1,count);
        for k=1:sum
            sam=randi(count);
            prop(sam)=prop(sam)+1;
        end
        k=1;
        for j=1:10
            if(X(i,j)==1)
                L(i,j)=prop(k);
                k=k+1;
    else

        L(i,j)=0;

    end

  end

end

sum=0;

for i=1:c

   for j=1:10

     sum=sum+L(i,j);

   end

end

sum=sum/10;

end
```

Generating the Client vs Fc/no. of servers

Servers = 10

Clients = 10-160
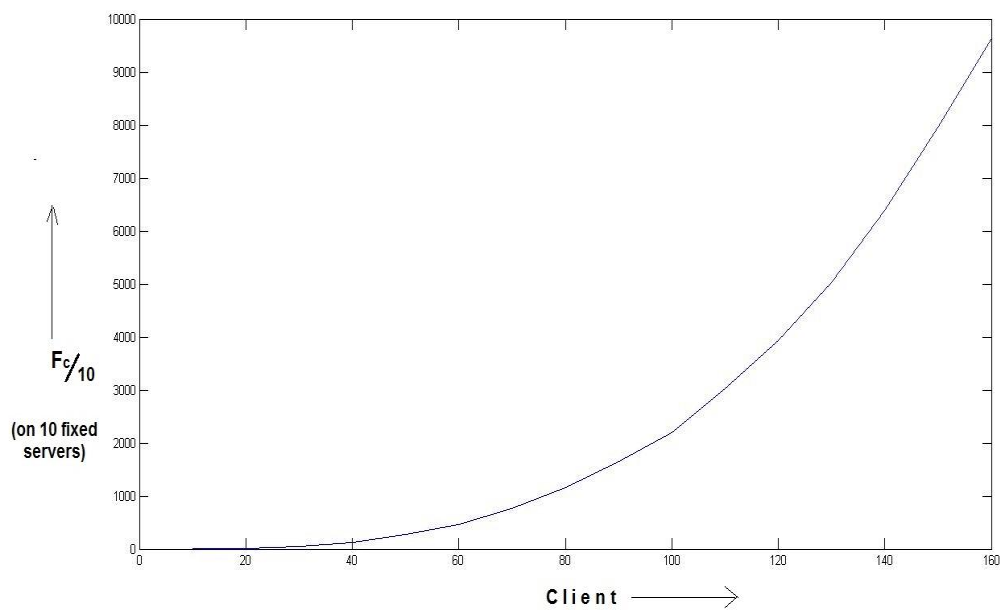
```
clc;

clear all;

count=1;

for i=10:10:160

    x(count)=i;

    y(count)=func(i);

    count=count+1;

end

plot(x,y);
```

Observation

From the graph (No. of clients vs Fc/no of servers) generated can be concluded that the Algorithm is performing well and optimized. Because by increasing the number of clients (Load) the threshold value is increasing exponentially. Hence the Algorithm performs well and can be applied in the real life applications.

Chapter 4
Fault Tolerance
In Client-Server Assignment

4.1 Fault Tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems.

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.

The term is most commonly used to describe computer systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software.

Within the scope of an individual system, fault tolerance can be achieved by anticipating exceptional conditions and building the system to cope with them, and, in general, aiming for self-stabilization so that the system converges towards an error-free state. However, if the consequences of a system failure are catastrophic, or the cost of making it sufficiently reliable is very high, a better solution may be to use some form of duplication. In any case, if the consequence of a system failure is so catastrophic, the system must be able to use reversion to fall back to a safe mode. This is similar to roll-back recovery but can be a human action if humans are present in the loop.

A highly fault-tolerant system might continue at the same level of performance even though one or more components have failed.

In this algorithm approach three types of failure is possible: [7]

1-   Server failure

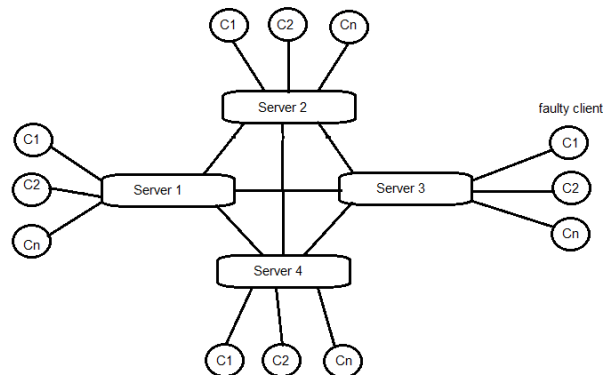2-   Client failure

3-   Connection failure

1-Server failure

If a server is found to be faulty then by increasing the threshold value all its clients will be distributed over the network to other servers. After diagnosis the system the faulty server can be fixed and all the backup data can be stored. When system gives green light to the particular server then the server can be reconnected to the system and threshold will be recalculated.

1   2
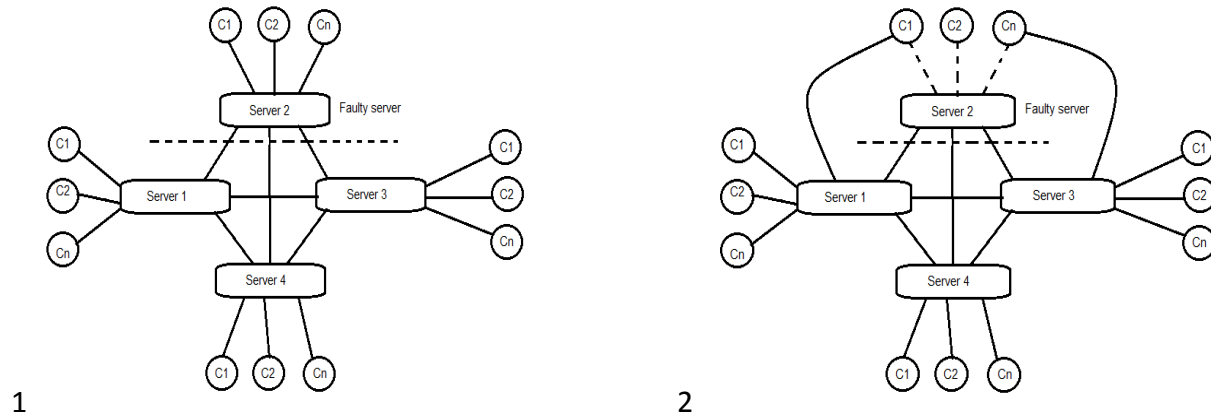
2- Client failure

In the client server over internet distributed system due to distribution of clients over the wide geographical area possibility of having a client faulty is more than server.

3- Connection Failure

In the system when a connection is failed or does not work properly that connection shall be terminated and a new optimized connection will be provided to the particular client/server.

1

2
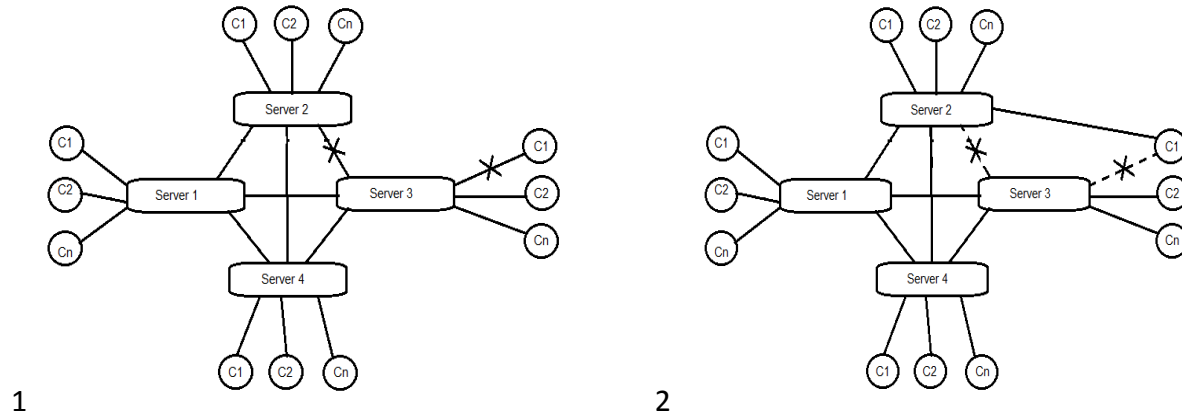
Fault Tolerance in this Proposed Approach

It works in a system in which servers are geographically distributed, while clients also distributed geographically and interact with the servers.

It is integrated with client caching and distributed transactions. It is optimized to reduce overhead on common client operations, by avoiding extra messages, processing and connectivity.

This approach can tolerate both server and client failures. Servers recover from crashes without loss of information; this is achieved without updating stable information when clients fetch data. Clients do not survive failures, yet the system is able to reclaim persistent data referenced only from failed clients. Communication failures such as network partitions may cause some servers to view a live client as failed, but the scheme prevents dangling references at such clients from corrupting persistent data.

Server failure

As an example for the evaluation of the approach we consider server1 as a faulty server and server1 will be temporary disconnected from the system to go under diagnosis and recovery.

The clients of server1 will be transferred to other servers which the particular clients are connected to them as well.

Here is the example 3.1 of chapter3 is being taken under consideration and evaluated
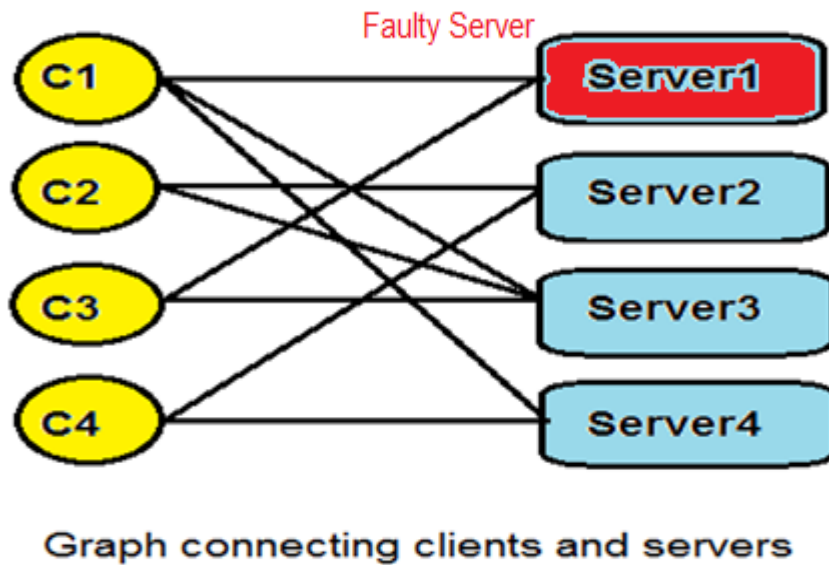
Graph connecting clients and servers

a) Maintaining matrix A_{N*N} which indicates the rate of the data exchanged between different clients communicating to each other:

		C1	C2	C3	C4
$A_{N*N}=$	C1	0	8	4	4
	C2	8	0	10	2
	C3	4	9	0	0
	C4	4	2	0	0

b) From the graph in fig. 3 maintaining a binary matrix X_{N*M} which denoting which client using which server (1 indicates the client using particular server and 0 not).

$X_{N*M} =$

	0	S2	S3	S4
C1	0	0	1	1
C2	0	1	1	0
C3	0	0	1	0
C4	0	1	0	1

c) The current load on each server can be calculated from matrix A_{N*N} and binary matrix X_{N*M} and the result is matrix L_{N*M}

$L_{N*M}=$

	0	S2	S3	S4
C1	0	0	8	8
C2	0	10	10	0
C3	0	0	13	0
C4	0	2	0	4

Loads of server1 (C1 and C3) are being distributed

The total load on each individual server can be calculated.

Load on server2 = 0+10+0+2 =12

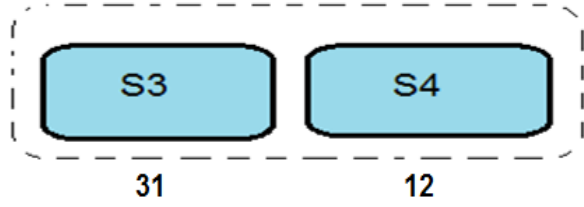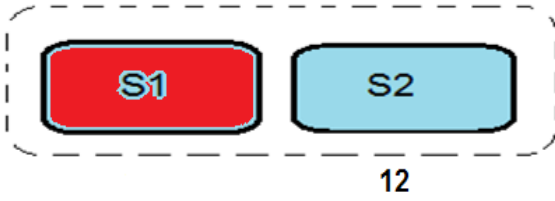Load on server3 = 8+10+13+0 =31

Load on server4 = 8+0+0+4 =12

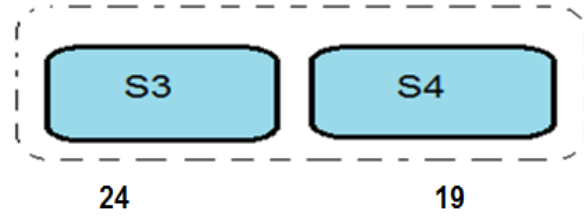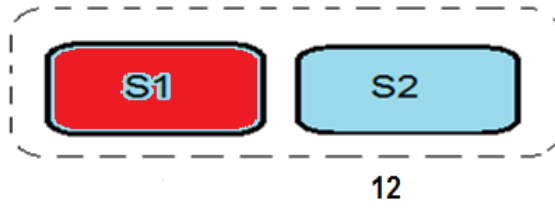d) Fc calculated by the total load on all the servers

Fc = 12 + 31 + 12 = 55

e) The total load is then divided by the number of servers in the system i.e. 3.

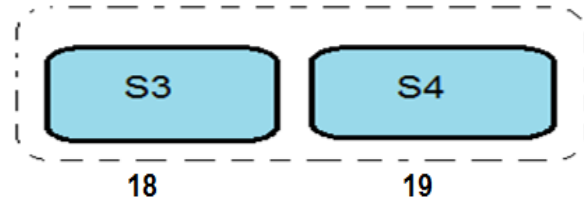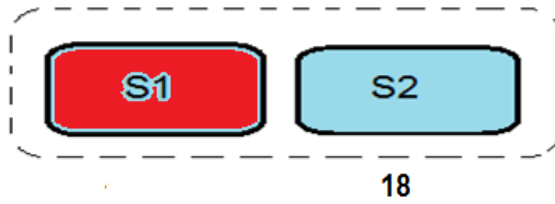Approximate average load on each server = 55/3 = 18.33 ~ 19

Initial load on servers:

S1	S2
12	

S3	S4
31	12

After Intra-server load balancing:

S1	S2
12	

S3	S4
24	19

After Inter-server load balancing:

S1	S2
18	

S3	S4
18	19

Chapter 5 Conclusion and Bibliography

4.1   Conclusion

In this work I present a theoretical model and an algorithmic solution to the client-server assignment problem for optimizing the performance of a class of distributed systems over the Internet.   The algorithm proposed for client server assignment problem is initiated from the Normalized Cut algorithm. My results indicate that the proposed algorithm almost always find the optimal solution. Future work can be progressed on the improvement of the proposed Algorithm, real life application and cloud system. One of the main areas for the future work which can be considered essential for the system is fault tolerance of the system which has one critical situation where all three server, client and connection can fail at the same time.

4.2 Bibliography

[1] http://phdfb1.free.fr/phdthesis/node108.html

[2] H. Nishida and T. Ngyen, "Optimal Client-server Assignment for Internet Distributed Systems," IEEE Trans, vol. 24, no. 3, Mar. 2013.

[3] Lu Zhang; Xueyan Tang; , "Client assignment for improving interactivity in distributed interactive applications" , 2011 Proceedings IEEE , page .3227-3235, 10-15 April 2011

[4] Zhang, L.; Tang, X.; , "Optimizing Client Assignment for Enhancing Interactivity in Distributed Interactive Applications," Networking, IEEE/ACM Transactions on, page no.99.

[5] Bortnikov, E., Khuller, S., Li, J., Mansour, Y. and Naor, J. S. (2012), "The load-distance balancing problem. Networks."

[6] http://docs.oracle.com/cd/E19728-01/820-2550/t3loadbal_config.html

[7] Umesh M. Barbara H. L. "Fault Tolerant Distributed garbage collection", MIT Laboratory for computer science, Cambridge, MA 02139