# Using CDN To Increase The Scalability Of The Server

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

**Computer Science & Engineering**

under the Supervision of

***Mr. Punit Gupta***

By

***Harsh Verma***

***111215***

to


Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

# Certificate

This is to certify that project report entitled “Using CDN to increase the scalability of the server”, submitted by Harsh Verma (111215) in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

**Date:**

**Supervisor’s Name :Punit Gupta**

**Designation: Asst.Professor**

# Acknowledgement

I am highly indebted to Jaypee University of Information Technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & Project Guide for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities

Date: Name of the student: Harsh Verma

# Table of Content

# List of Figures

# Abstract

A content delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centres across the Internet It deliver WebPages and other Web content to a user based on the geographic locations of the user, the origin of the webpage and a content delivery server.CDN serve a large fraction of the Internet content today, including web objects ,downloadable objects , live streaming media, on-demand streaming media, and social networks. A CDN (content delivery network) is a network of servers located in different parts of a country (or the globe) that stores files to be used by your website visitors. The reason they exist is because there is a measurable amount of latency (waiting time) for a website user who is visiting a page that is hosted thousands of miles away. There are also routing issues that can occur when a user is seeing such a webpage. If someone in New York is using a webpage that is hosted in Los Angeles they are seeing a slower version of that webpage because of the above mentioned routing issues and sheer distance the files have to travel. By having your files on several servers across a geographical area you can make sure the user is loading files that are near them, not all the way across the country or ocean.

# Motivation

A content delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centres across the Internet. There are several reasons why a CDN could benefit your website and company.

**1. Different domains**
Browsers limit the number of concurrent connections (file downloads) to a single domain. Most permit four active connections so the fifth download is blocked until one of the previous files has been fully retrieved. You can often see this limit in action when downloading many large files from the same site.

CDN files are hosted on a different domain. In effect, a single CDN permits the browser to download a further four files at the same time.

**2. Files may be pre-cached**
jQuery is ubiquitous on the web. There's a high probability that someone visiting your pages has already visited a site using the Google CDN. Therefore, the file has already been cached by your browser and won't need to be downloaded again.

**3. High-capacity infrastructures**
You may have great hosting but I bet it doesn't have the capacity or scalability offered by Google, Microsoft or Yahoo. The better CDNs offer higher availability, lower network latency and lower packet loss.

**4. Distributed data centers**
If your main web server is based in Dallas, users from Europe or Asia must make a number of trans-continental electronic hops when they access your files. Many CDNs provide localized data centers which are closer to the user and result in faster downloads.

**5. Built-in version control**
It's usually possible to link to a specific version of a CSS file or JavaScript library. You can often request the "latest" version if required.

**6. Usage analytics**
Many commercial CDNs provide file usage reports since they generally charge per byte. Those reports can supplement your own website analytics and, in some cases, may offer a better impression of video views and downloads.

**7. Boosts performance and saves money**
A CDN can distribute the load, save bandwidth, boost performance and reduce your existing hosting costs — often for free.

# Chapter 1

# Introduction

A content delivery network (CDN) is a system of geographically distributed servers around the world. These servers maintain replicas of the content. The primary benefits of a CDN are about speed and high-availability, but a CDN brings an entire range of other advantages. First of all a CDN accelerates websites or data delivery. A CDN significantly reduces the loading time and improves the preserved user experience. Content is served from the closest edge server available. A so-called edge server lowers latency as it is closer to end-users and the closer a server is, the less congestion is along the way.

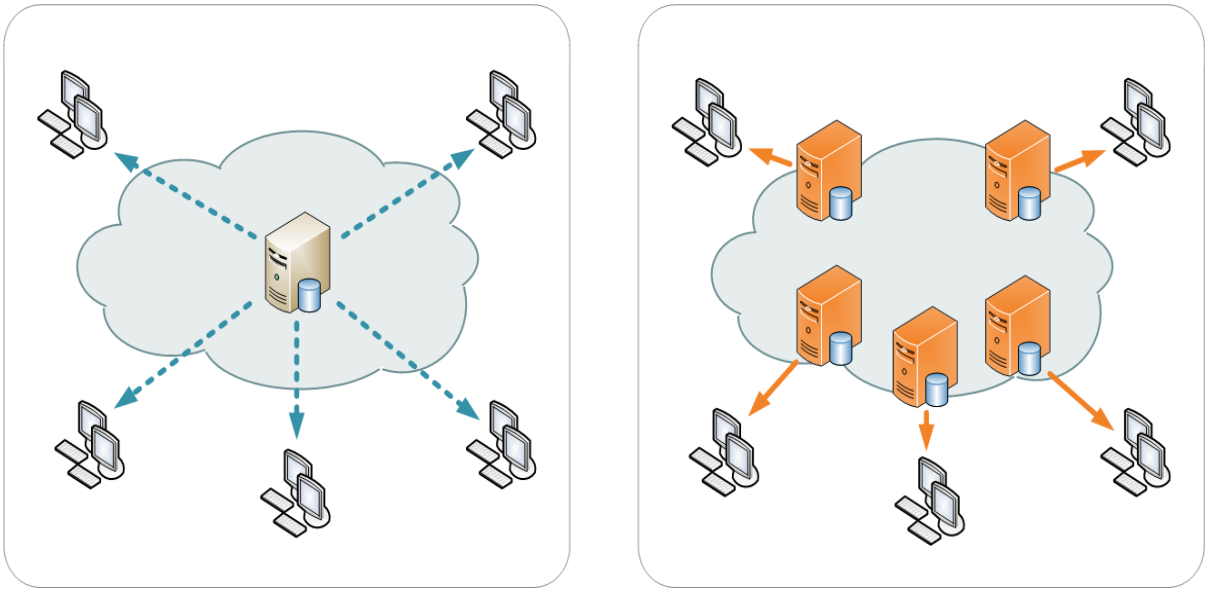


**Figure 1**: Difference between regular and CDN servers

First figure shows the multiple users accessing data from the single server while the second figure shows the data being taken from multiple servers

When it comes to select a Content Delivery Networks (CDN) there are many factors you need to consider such as performance, availability, scalability, reliability, pricing model, features, SLA etc. performance and pricing are key in the decision making process.

# 1.1 Performance

CDN performance is measured in terms of speed of DNS resolution and content delivery. Normally performance measurement includes both,

1. How quickly CDN's DNS system supply the address of the best CDN content server? DNS resolution is based on latency, geo-graphic location or price class. It has been reported that many CDNs employ a huge number of DNS servers that are co-located with their content servers.
2. How quickly content is delivered from CDN server to client?

Quantifying performance of a CDN at large scale requires the ability to either capture traffic on the CDN's servers or control a large number of globally distributed clients.

For CDN performance testing there are not any standard protocols to measure the performance across CDN. Each CDN vendors tends to use different methodologies to measure performance. Given that under laying architecture and network distribution is neither same nor very visible so it is really hard to compare CDN vendors in terms of performance. Geographic location of CDN data centers plays a big role in performance measurements.

Having said that, CDN performance testing can be segment based on delivery path and main Internet bottlenecks

- First mile testing (CDN server level)
- Backbone testing (Internet hubs)
- Last mile testing

# Backbone testing:

There are two quite different types of network performance measurement techniques. The test most popular amongst CDNs is called “Backbone” testing. “Backbone” tests demonstrate how fast your site loads at major Internet hubs. The test ignores the CDN’s performance from the Internet hub to the specific device a customer is using, thus the test is not measuring the true customer experience. Most CDNs use this test because it is easier to implement, takes less time and costs less money.

Of course the problem with this method, as mentioned before, is that it’s not a real-world result. Customers don’t live in data centers, and this testing method ignores the most important element of incorporating a CDN into your IT infrastructure: improving customer experience by more rapidly delivering content to user’s devices. The other problem with testing the

performance of a network on the “Backbone” and at the data center level is that vendors can do some tricks to optimize performance results, such as strategically placing CDN servers at specific locations near known “Backbone” testing agents, thus ensuring optimal tests and results

# Last mile testing:

The alternative type of CDN performance measurement is called “Last Mile” testing. While some may want to debate which measurement method is better, there is no question that “Last Mile” measurement of CDN performance is a much more realistic interpretation of end-user experience. “Last Mile” testing incorporates measurement of how quickly content is delivered from the CDN’s server to the consumer’s device, incorporating the last leg of connectivity services required to deliver content from an Internet service provider (ISP) to the customer. For any customer or enterprise user looking to get real-world network performance results from any vendor, they should insist that the vendor’s methodology includes testing the “Last Mile” of delivering content to consumers.

There are two different ways that you can measure results to the user’s device. The first is to use independent performance testing companies such as Compuware Gomez that can be configured to perform objective “Last Mile” testing of CDN performance. Using a third-party performance testing company is the most common way that CDN vendors measure their own latency for delivering objects to real-world users located around the globe.

A second valid method of “Last Mile” testing can be conducted by incrementing a client application or website to collect real latency statistics from actual users or visitors. This approach is typically used by CDN customers to understand the actual performance they are getting from their CDN provider or providers.

Neither of these approaches is inherently better or worse than the other. Both provide accurate observations that precisely capture end-users’ experiences. Below, we will look at data collected by each of these two ways of measuring “Last Mile” performance..

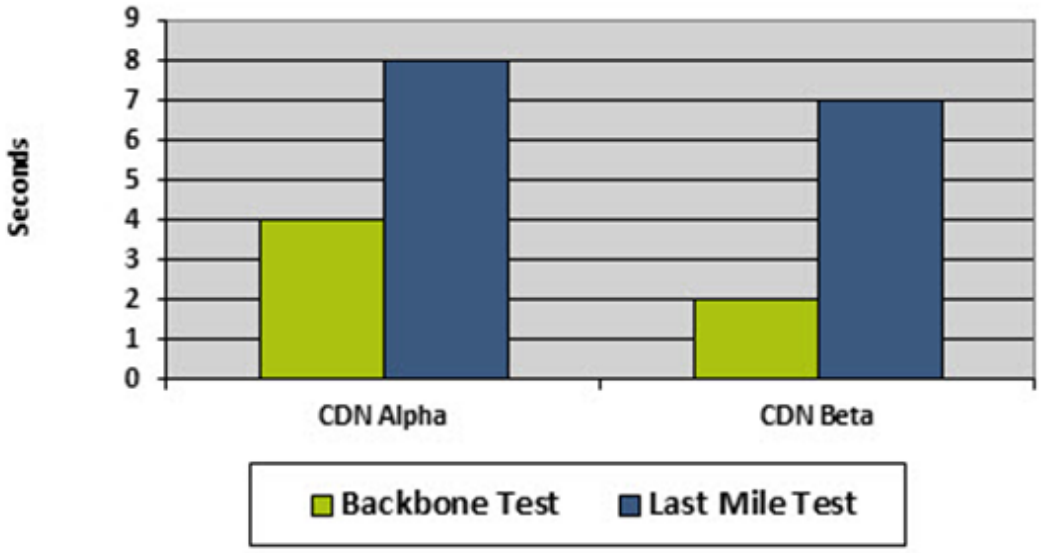



**Figure 2**: Comparison of two different CDNs and two different test types

# 1.2 Availability

Although speed of content delivery is quite important, CDN availability and uptime cannot be ignored. For availability and uptime evaluation, both cluster availability and server availability are measured where "cluster" refers to a collection of CDN servers in the same location. As you may expect that clusters have better availability than individual content servers. Often CDN vendors employ auto-scaling techniques which provisions new content servers during high-load conditions.

# 1.3 Scalability

Load testing is a way to measure scalability and elasticity of a CDN service. Unfortunately traditional load testing methods don't work well with CDNs. For instance if you test by sending requests from a single client to just one of the IP addresses that DNS returns, your requests may resolve to a single server in one

CDN edge location. Often load testing is not very straightforward as each CDN network uses different load handling techniques. Especially due to auto-scaling, clustering of content servers and geo-location based DNS resolution - it is not easy to use something like Apache Benchmark. Moreover IP address range also keeps changing due to adding and removing of the content server as part of auto-scaling.

To perform load testing with CDNs one must,

1. Send client requests from multiple geographic regions (Load testing at edge location level based on DNS resolution).
2. For each geographic region, use multiple measurement clients with each client making an independent DNS request so that each client will receive a different set of IP addresses from DNS (Load testing at cluster level).
3. Now for each client that is making requests, spread your client requests across the set of IP addresses that are returned by DNS, which ensures that the load is distributed across multiple servers (Load testing at server level).

# 1.4 Reliability

Packet-loss measurements are used to determine the reliability of a CDN. High reliability indicates that a CDN incurs less packet loss and is always available to the clients

# Chapter 2:Problem Statement

## Introduction:

As the explosive growth of internet users continues, internet service providers must quickly adapt and scale their network infrastructures in order to provide high quality service to their users According to recent predictions, global IP traffic will increase by 4 times from 2009 to 2014 reaching 64 Exabyte's per month in 2014. However this fast-growing network traffic causing occasional problems in telephone service has been triggering some significant concerns that it might lower the competitiveness of the IT industry in the long run. Even with the current network infrastructure needed to be extended and upgraded to cope with the upcoming traffic explosion, network providers are unwilling to invest on establishing more network infrastructure due to unsatisfactory returns
for all their efforts and costs. So, we are now facing a new phase to come up with new eco system to bring about a win-win situation for all providers in network, content and application as well as service users in need of enhanced user experiences from the network. Responding to various needs from the ever-growing internet user, future network should offer alternative promising service more than end-to-end connection to the network end users. To guarantee the fast connection for the end users when they are using the internet is not only role of network anymore. Future network should focus more on what users want to get from the network rather than
where users want to make a connection through the network . For the purpose of providing optimized high
quality service, especially for the content service to network users directly, the service network is equipped with storages and servers inside the network The content service network consists of several content service nodes and each node is a combination of server, storage and network devices altogether. The network device of the node performs both interconnections amongst network nodes and combining server and storage into the network device to build up the content service node. Another network element for the content service network is the content service network manager performing network monitoring, coordination and portal server of the content or application service. Authentication, consent and path provisioning functions are also embedded in the content service network manager . The rest of this paper is organized as follows. Section II summarizes global server load balancing functional architecture for the distributed content service network. Then, Section III presents hierarchical load balancing operational procedures for the content service network.

# Chapter 3: Related Models

## 3.1Global Server Load Balancing

Global Server Load Balancing (GSLB) is originally created to minimize the damage from the natural disasters such as earthquakes or tornados Its main role is maintaining consistency for the datum in data center safely using backup servers or remote recovery data center. In distributed content delivery network shown in
Figure 1 where a multitude of contents are distributed over the network a special method to manage all of the contents is required. To do this we subdivided the whole network into subnets connecting clients with content servers based on regional locality

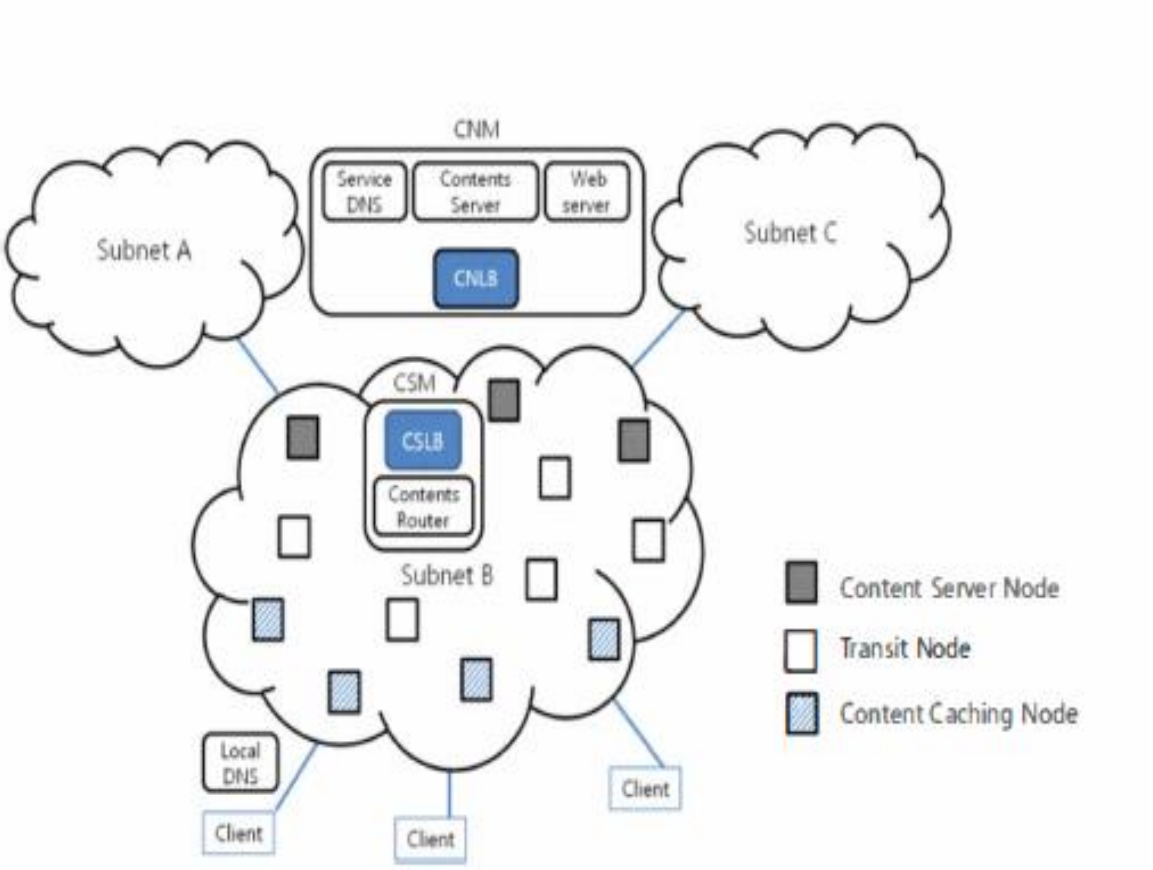



**Figure 3**: An example of Distributed Content Service Network

# A. Content Network Management

Content Network Manager (CNM) performs service gateway for the clients who want to get content service from the network by informing content server address to the clients. CNM has abstract knowledge of subnets in the content network and selects optimal subnet to provide the content requested. Content Network Manager taking care of whole content network cooperates with CNLB (Contents Network Load Balancing) to find the optimal subnet and Content Server for connection between the client subnet which the client sits in and server subnet which holds the requested content in it. It is also needed for service DNS function to translate the content name into IP address for the user side in cooperation with the web server which provide the users web service interfaces it shows structure of Content Network Load Balancing. The gathering function of the subnet abstract information on each subnet resides in the Subnet Information Gateway (SIG) joint with the CSLB in each subnet.

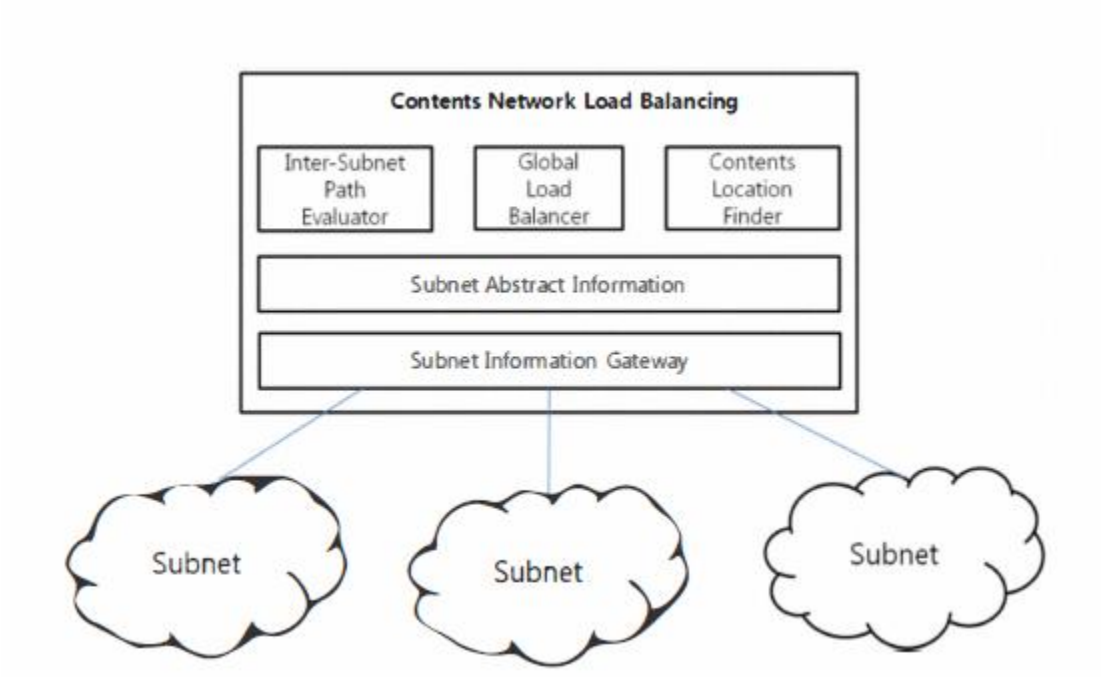



**Figure 4**: Content Network Load Balancing System Structure

The Subnet Abstract Information includes condensed information about contents directory and status for each subnet to identify which subnet contains the requested content without unnecessary information of the subnet.
Inter Subnet Path Evaluator estimates the costs between the server subnets to determine the optimized subnet when multiple server subnets were founded. Also, searching for the content and finding the subnet including the content are the role of Content Location Finder. To sum up these functions Global Load Balancer returns the optimal subnet in response to the content service request.

## B. Content Subnet Management

In each subnet, there are a group of service nodes composing the subnet and Contents Subnet Manager (CSM) for the purpose of collecting and manage the information of network and contents distributed over the subnet. CSM has the CSLB (Contents Subnet Load Balancing) function that finds the optimal content service node in the subnet. Content Router function takes responsibility of connection between the client service node and content service node. The structure of Contents Subnet Load Balancing (CSLB) element in CSM  Subnet Monitoring Element (SME) gathers information of the subnet by monitoring the network nodes, interfaces and links in the subnet. It generates network traffic load status information as well as content server load status. And also SME produce content routing information base by monitoring on content location over the network

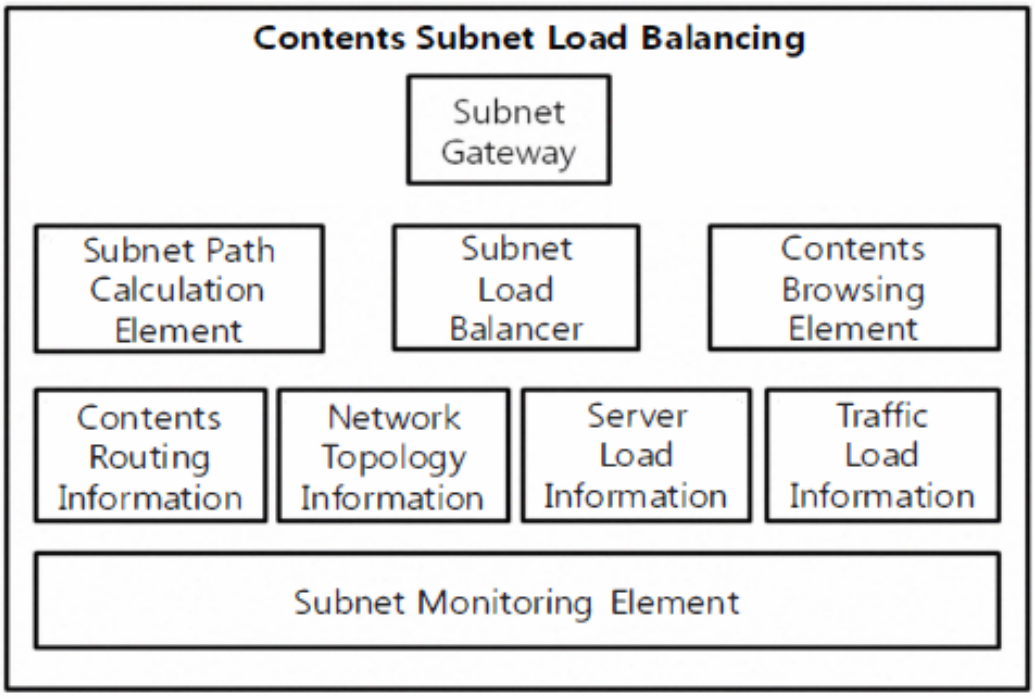



**Figure 5:** Content Subnet Load Balancing System Structure

Confirmation process to decide whether the requested content belongs to its subnet is performing at the Content Browsing Element. Subnet Path Calculation Element computes optimal path from the content service node to the user node using collected information: Network Topology Information, Server Load Information and Traffic Load Information. Subnet Load Balancer controls CSLB procedures after receiving the content service request through Subnet Gateway. Firstly it checks whether its subnet contains the requested content or not with Content Browsing Element and if so, it computes the path using Subnet Path Calculation Element. Finally Subnet Load Balancer returns the optimal content service node to the Subnet Gateway. Preparing for the case that the requested content was not in the corresponding subnet user resides Subnet Gateway notify Subnet Information Gateway in the CNLB of the abstract information of the subnet

# 3.2 .HIERARCHICAL LOAD BALANCING

## A. Optimal Subnet Selection

The optimal subnet selection procedure is shown in the figure below. First of all CNLB confirms if the subnet that a user resides has the content that user wants to get. If exists there is sno need to access other subnet deliberately. Because the content network is composed of several subnets duplicate data could be spread over the entire network. After the CNLB selects the subnet it sends the identification number of the subnet to the Content Server in the CNM. Then Content Router in the corresponding subnet starts to find the content service node according to the request of the Content Server with the content information and IP address of the client.

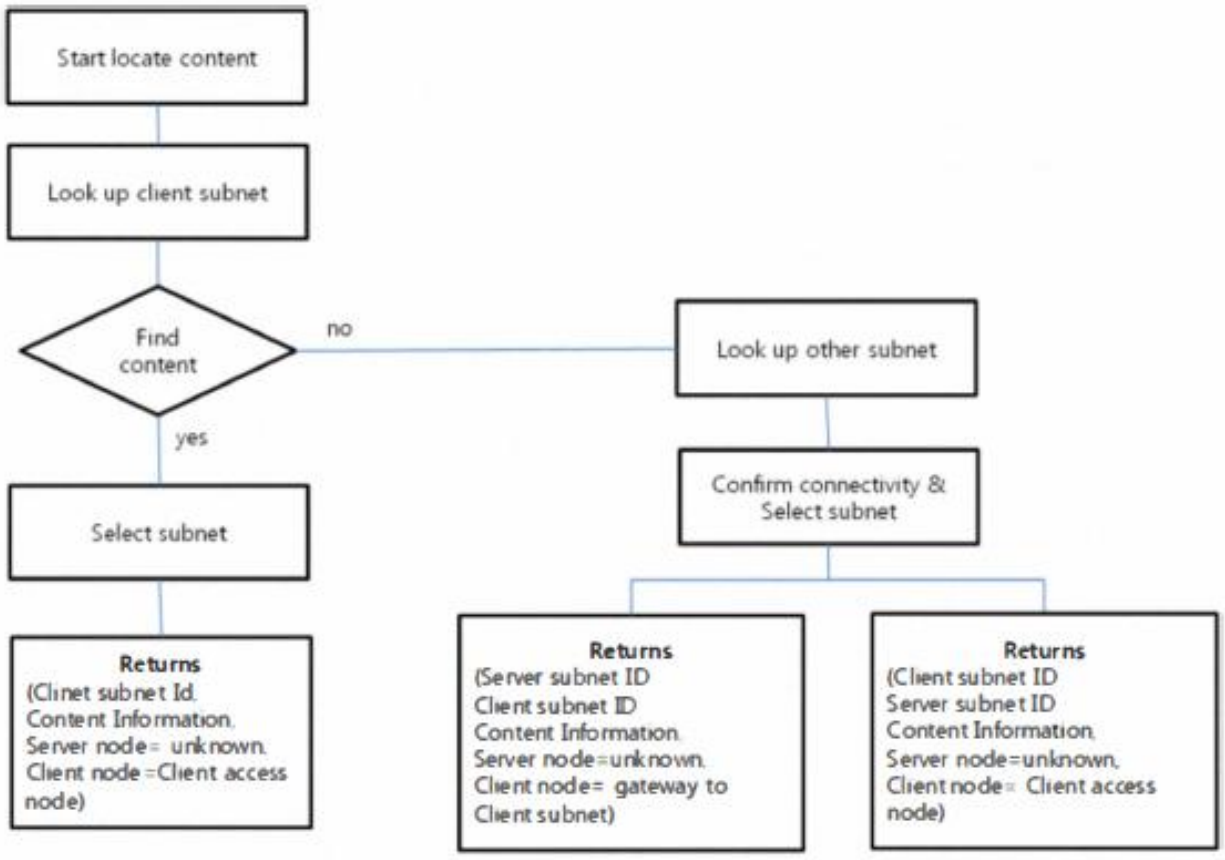



**Figure 6**: Optimal Subnet Selection Procedure

## B. Optimal Content Node Selection

As a response to the request Content Router inquires the CSLB to find the content service node available to offer the service to the client. CSLB uses node information table and real-time streaming flow table keeping content routing information and network topology maps. Figure 5 illustrates the procedure to select optimal content service node. With the information provided candidate nodes providing the requested content will be extracted. An optimal node will be chosen according to the costs for connecting the server nodes with the client node when multiple content service nodes are qualified. Content server node as well as cache node could be given selection as a content service node in the subnet. F or the general content service such as massive file transfer or VoD services content service node will be served using the network caching capability. But, in case of real-time streaming more information should be required to serve streaming service to the client. Real-time streaming service has a feature that it requires to convey the real-time content flow to the destination with minimum delay without saving or caching through the network. Real-time content request from the client node look up the node information table to distinguish real-time streaming server node in the subnet

The available content service node among the corresponding real-time streaming flow that capable of streaming with a higher proximity to the client will be selected for real-time streaming service. With the address of the content service node selected finally, client could make a reliable connection to receive the content from that network. However, the requested content was not found within the subnet which the requested client connected with CNLB will check whether other subnets own the requested content or not. The next step to enable inter-subnet connection for the content is to confirm the connectivity between the client subnet where the client sits and the server subnet where the content belongs to. Upon holding a reliable connection between the subnets in terms of traffic overload and network topology Content Server requests the path provisioning over the server subnet from the content service node in that subnet to the gateway node interconnecting the server subnet with the client subnet. In the client subnet, a path from the gateway node connected to the server subnet to the client node should be established to provide the content from another subnet to the client.

For the effective content network provisioning client network manager have to determine whether the client subnet stores the corresponding content as its own content while the content transit cache node or server node in the client subnet. And also the optimal cache node should be on the path that the content should drop by in terms of operator's policy and network optimization.

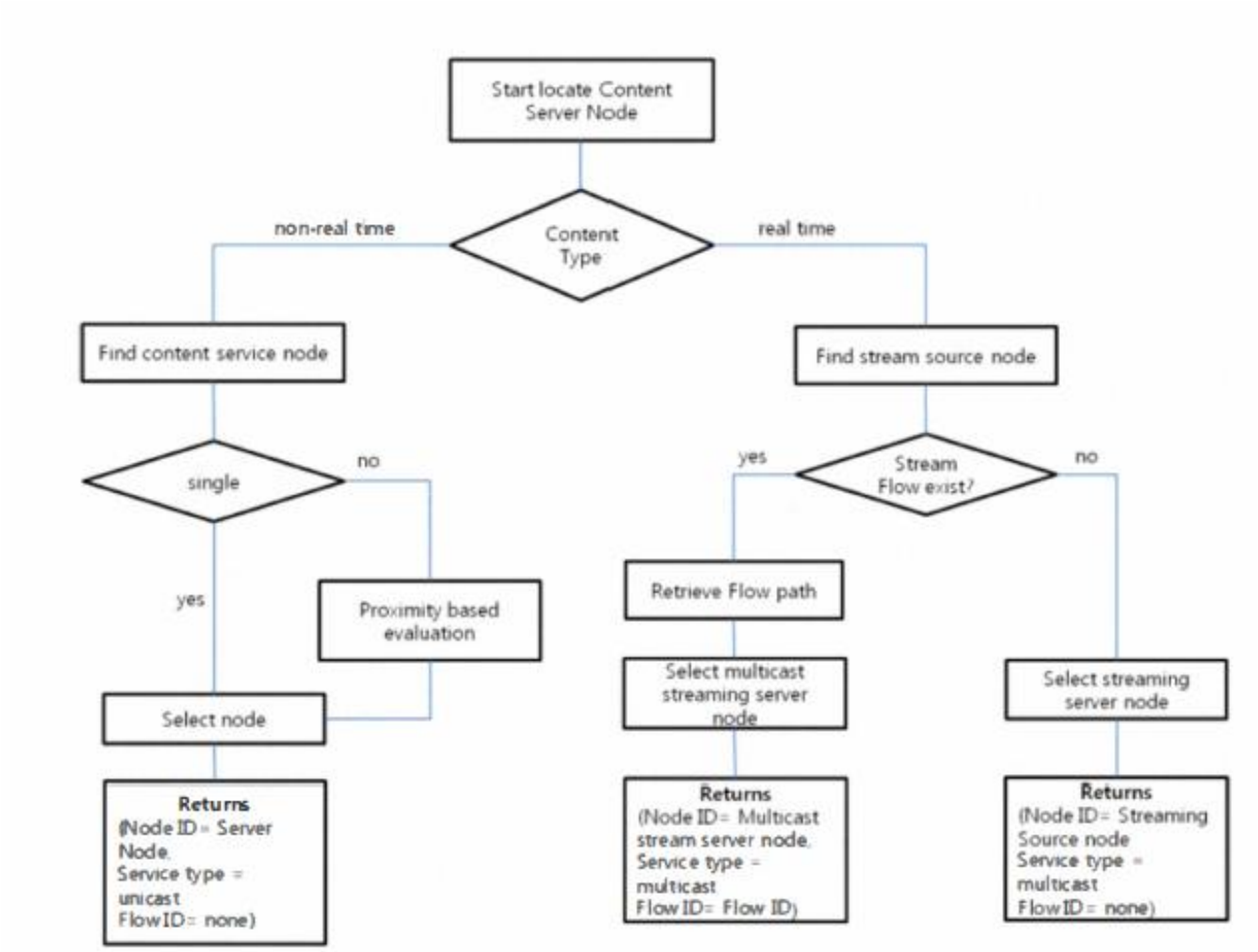



**Figure 7.** Optimal Content Service Node Selection Procedure

## 3.3 CONTENT DISTRIBUTION TECHNIQUE WITH IN VIRTUAL ORGANIZATION(vo) BASED PEERING CONTENT DELIVERY NETWORK

Day by day Internet is facing the problem of unmanageable amount of traffic flow which results many requests are being lost. Content Delivery Network replicates the same content or services over several mirrored Web servers strategically placed at various locations to improve performance and scalability. The user's request is redirected to the nearest server and this approach helps to reduce network impact on the response time of the user

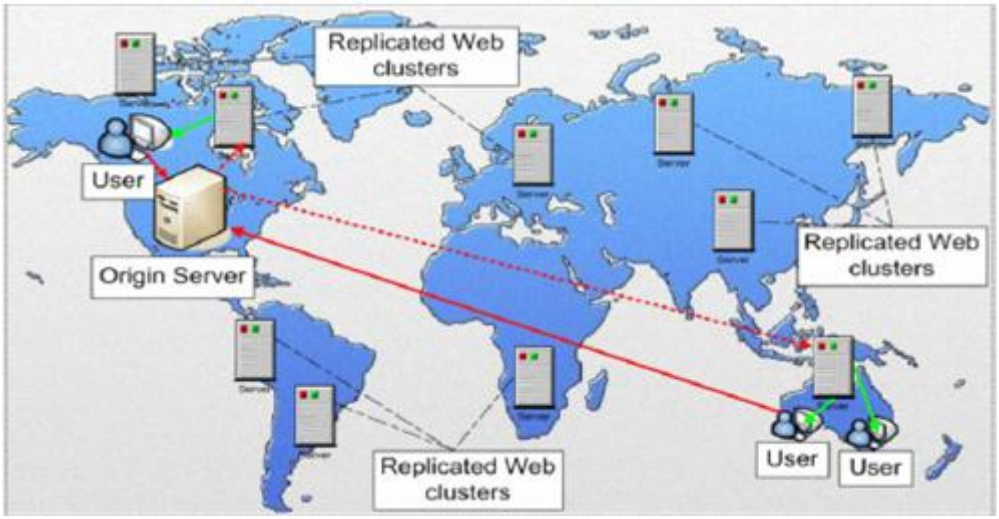



**Figure 8**: A simple model of CDN

In this figure , a user from Australia is requesting for contents from the origin server in USA. As the distance between these two continents is huge it takes a greater response time for the user. In this situation CDN provides a surrogate server (replica of origin server) in Australia to serve the end-user. A CDN focuses on building its network infrastructure to provide the following services and functionalities: storage and management of content, distribution of content among surrogates, cache management, delivery of static, dynamic and streaming content, backup and disaster recovery solutions, and monitoring, performance measurement and reporting.

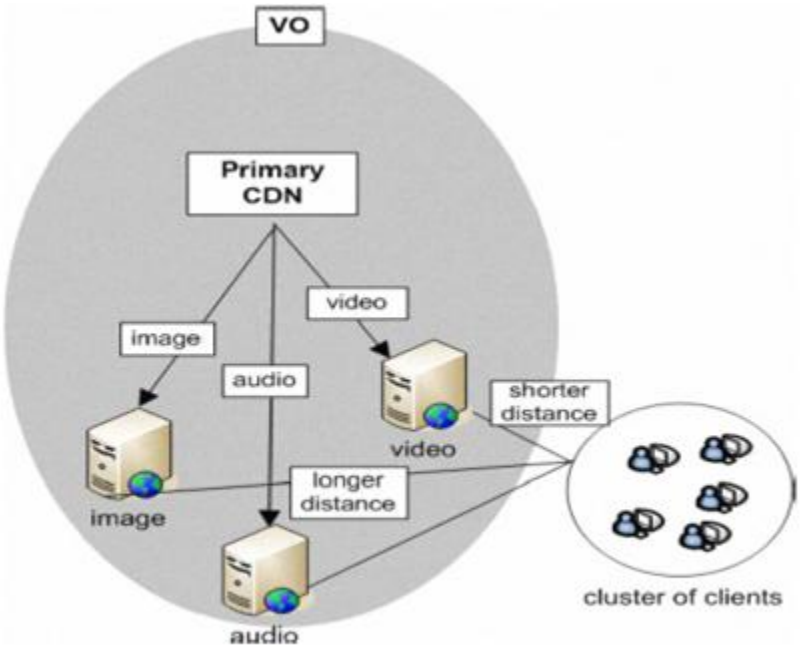
## A. Problems of existing CDN

We have found some problems in existing CDN (especially commercial CDN) based on our observation. Existing commercial CDNs are proprietary in nature, Each of them has created its own closed delivery network, which is expensive to setup and maintain, Running a global CDN is even more costly, requiring an enormous amount of capital and labor, Content providers typically subscribe to one CDN and thus cannot use the resources of multiple CDNs at the same time, which may cause Service Level Agreements (SLA, An SLA is a contract between the service provider and the customer to describe provider's commitment and to specify penalties if those commitments are not met violation in many cases. To solve these problems Virtual Organization (VO) based peering CDN may be a possible solution

## B.VObased Peering

The main purpose of CDN is to provide necessary distributed computing and network infrastructure so that SLAs are met with customers. Better SLAs can be met by groups of cooperating CDN. Cooperation among the peering CDNs is achieved through a Virtual Organization (VO). Virtual organizations are composed of a number of semi-independent

**Figure 9**: Group content distribution



## C. Proposed Similar Content distribution technique

According to similar content distribution technique similar types (e.g. video, audio, images etc) of contents will be grouped together and cached in same surrogate in a virtual organization based CDN. Here the similarity will depend on the weight of the contents. In a VO the place from where most requests are coming will be measured first and will be considered as a cluster. Then distance of the surrogates from the cluster will be calculated. It may be done by measuring number of routers (Hop count) in case of same capacity link. Depending on the distance different types of grouped content will be placed in surrogates. The contents having highest weight will be placed in the closest surrogate and lighter contents will be placed in the distant surrogates. As contents having more weights needs more time to transfer, placing them in closer surrogates will reduce server load. In this technique surrogates can provide faster transfer of content and bandwidth consumption will be less.

# II. Performance evaluation

Experiment Setup

We took two approaches in our experiment. In the first approach different weighted contents are placed in each surrogate (Random Content Distribution Technique). In the second approach we placed almost similar weighted contents in a surrogate. In both case we placed surrogates in random distance from the user but for similar content distribution surrogates having heavier contents are placed closer to the user.

## A. Performance criteria

Here we briefly present the performance criteria used in the experiments, namely

- mean response time
- response time CDF
- hit ratio and
- byte hit ratio

These criteria have been used since they are the most indicative ones for performance evaluation.

- Mean response time. This is the expected time for a request to be satisfied. It is the summation of all request times divided by their quantity. Low values denote that content is close to the end-user.

- Response time CDF. The Cumulative Distribution Function (CDF) in our experiments denotes the probability of having a response times lower or equal to a given response time. The goal of a CDN is to increase the probability of having response times around the lower bound of response times.

- Hit ratio. It is defined as the fraction of cache hits to the total number of requests. A high hit ratio indicates an effective cache replacement policy and defines an increased user servicing, reducing the average latency.

- Byte hit ratio. It is the hit ratio expressed in bytes. It is defined as the fraction of the total number of bytes that were requested and existed in cache to the number of bytes that were requested. A high byte hit ratio improves the network performance (i.e. bandwidth savings, low congestion etc.).

## B. Performance Measurement

For performance measurement we have compared our proposed similar content distribution technique with general random content distribution, where content is distributed in random basis and unstructured way .autonomous entities (including different individuals, departments, and organizations) that come together to share resources and to collaborate on shared goal

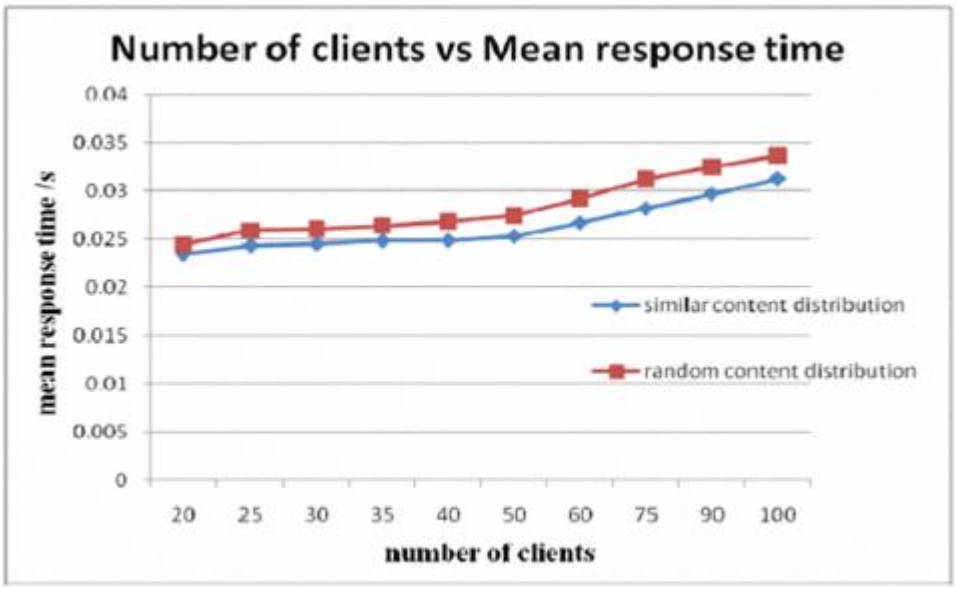



**Figure 10**: Number of clients Vs Mean response time

Another finding is, when client numbers are fixed in a network and the number of requests increases then the mean response time of similar content distribution is always less than random content distribution.

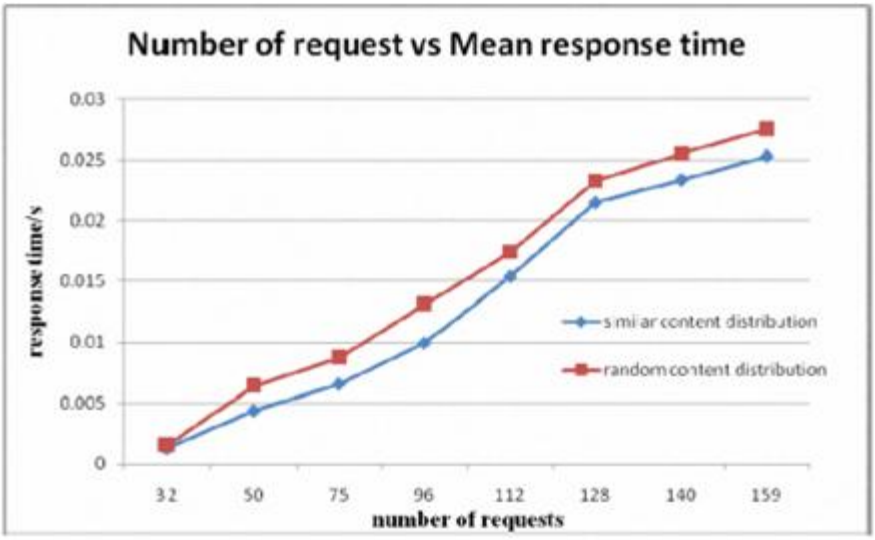



**Figure 11**: Number of requests Vs Mean response time

The reasons that make mean response time of similar content distribution less: M/G/1 queue: M/G/1 queue can be used to show model of a general CDN (Shown in Figure 5). An *MlG/!* queue consists of a FIFO buffer with requests arriving randomly according to a Poisson process at rate I and a processor, called a server, which retrieves requests from the queue for servicing. We assume that the total processing of the Web servers of a CDN is accumulated through the server and the service time is a general distribution. User requests are serviced at a first-come-first-serve (FCFS) order [8]. When multiple request comes at the same time then those request are inserted in this queue of that surrogate and requests are served as first come first serve basis. In this way in random content distribution technique surrogates may have long queue of requests . For this surrogate needs more time to serve multiple requests in random content distribution technique . But in similar content distribution there will be less number of requests in each surrogate due to request distribution between different surrogates. So queue size is reduced. That's why it takes less response time.

**Parallelism**: In general clients request for different types of contents. In random content distribution different types of content can be placed in same surrogate. So it needs more time to serve because it is a sequential process. But in similar content distribution different content are placed in different surrogates based on similarity. So those surrogates can serve the client in parallel basis. The reasons that make mean response time of similar content distribution less:

M/G/1 queue: M/G/1 queue can be used to show model of a general CDN (Shown in Figure 5). An *MlG/!* queue consists of a FIFO buffer with requests arriving randomly according to a Poisson process at rate I and a processor, called a server, which retrieves requests from the queue for servicing. We Generally surrogates serve contents from its cache. Hit ratio percentage is the ratio between the number of contents a surrogate is serving and the number of content request it is receiving. A high hit ratio indicates an effective cache management policy. It improves network performance and bandwidth saving. From Figure 6 we can see that for particular number of request, hit ratio percentage of similar content distribution is always higher than hit ratio percentage of random content distribution.

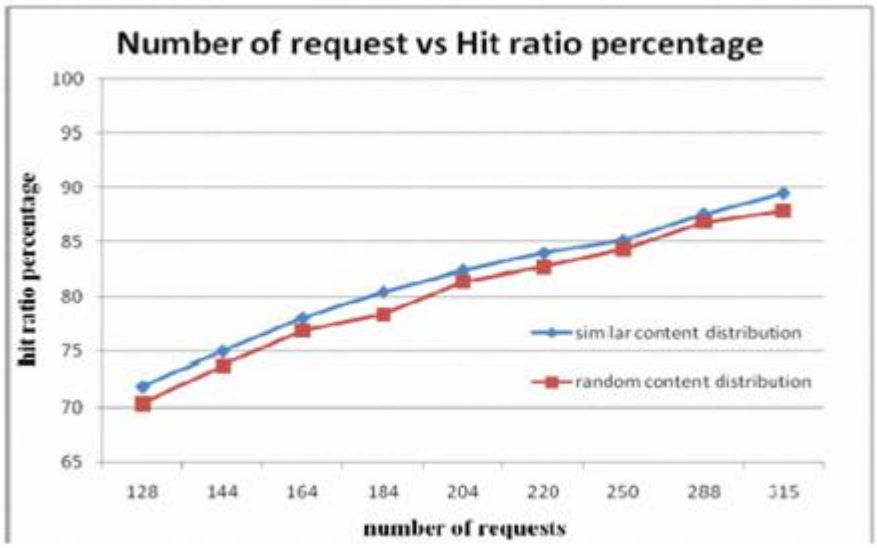



**Figure 12**: Number of requests Vs hit ratio percentage

Less number of redirection: In similar content distribution most of the time the surrogates are able to serve the request as load is almost equally balanced, so redirection probability is less. But in random content distribution redirection this probability is higher, and in worst case it may happen that there is no requested content in a surrogate. So the surrogate redirects the requests to other surrogates that have those contents. Figure 7 shows that when server load is in increase the expected waiting time also increases . But in similar content distribution technique expected time is less in compared with random content distribution technique.

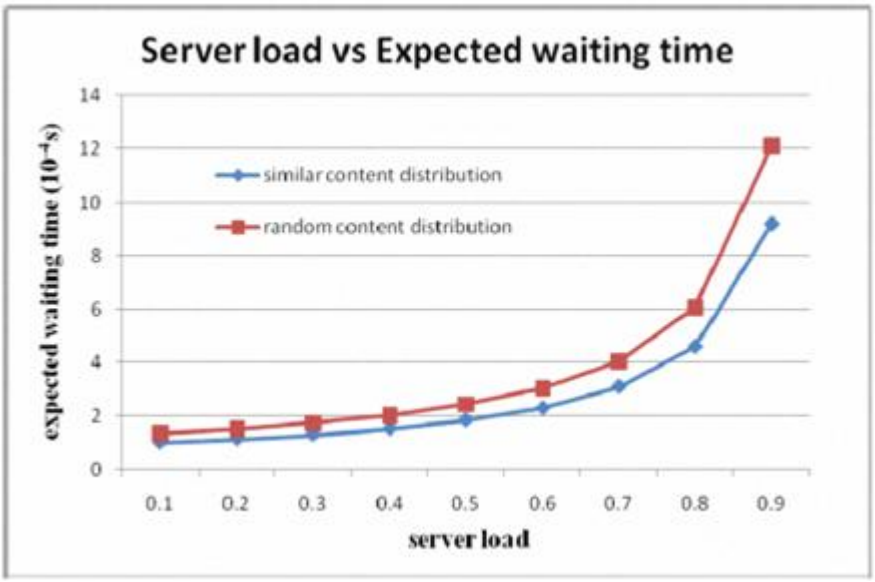



**Figure 13**: Server load Vs Expected waiting time

P-K formula for calculating expected waiting time :
*E (W) = AE(x)/ 2(l -p)*
E(W)
E(x)
A,
J!
P
Expected waiting time
Mean service time
Mean arrival rate
Mean service rate
Load *OJ* J!)

According to P-K formula when server load increases the mean service time and mean arrival rate also increases proportionally. In the parallel content distribution mean service time is less in compared with random content distribution due to M/G/l queue.

# 3.4 Energy-Aware Load Balancing in Content Delivery Networks

Large Internet-scale distributed systems deploy hundreds of thousands of servers in thousands of data centers around the world. Such systems currently provide the core distributed infrastructure for many popular Internet applications that drive business, e-commerce, entertainment, news, and social networking. The energy cost of operating an Internet-scale system is already a significant fraction of the total cost of ownership . The environmental implications are equally profound. A large distributed platform with 100,000 servers will expend roughly 190,000 MWH per year, enough energy to sustain more than 10,000 households. In 2005, the total data center power consumption was already 1% of the total US power consumption while causing as much emissions as a midsized nation such as Argentina. Further, with the deployment of new services and the rapid growth of the Internet, the energy consumption of data centers is expected to grow at a rapid pace of more than 15% per year in the foreseeable future . These factors necessitate a complete rethinking of the fundamental architecture of Internet-scale systems to include energy optimization as a first-order principle. An important Internet-scale distributed system to have
evolved in the past decade is the content delivery network (CDN, for short) that delivers web content, web and IPbased applications, downloads, and streaming media to end-users around the world A large CDN, such as that of a commercial provider like Akamai, consists of hundreds of thousands of servers located in over a thousand data centers around the world and account for a significant fraction of the world's enterprise-quality web and streaming media traffic today The servers of a CDN are deployed in clusters where each cluster consists of servers in a particular data center in a specific geographic location. The clusters are typically widely deployed on the "edges" of the Internet in most major geographies and ISPs around the world so as to be proximal to clients. Clusters can vary in size from tens of servers in a small Tier-3 ISP to thousands of servers in a large Tier-1 ISP in a major metro area. A CDN's servers cooperatively deliver content and applications to optimize the availability and performance experienced by the clients Specifically, each client request is routed by the CDN's load balancing system to an "optimal" server that can serve the content with high availability and performance. Content and applications can typically be replicated on demand to any server of the CDN. The load balancing system ensures high
availability by routing each client request to an appropriate server that is both live and not overloaded. Further, the load balancing system ensures good performance by routing each
client request to a cluster that is proximal to that client. For instance, a client from a given metro area would be routed to a server cluster in the same metro area or perhaps even the same last-mile network. The proximity (in a network sense) of the client and the server ensures a communication path with low latency and loss. A comprehensive discussion of the rationale and system architecture of CDNs

**Problem Description.** In this paper, we focus on reducing the energy consumption of large Internet-scale distributed systems, specifically CDNs. Energy reduction in CDNs is a multi-faceted problem requiring advances in the power usage effectiveness (PUE) of data centers, improvements in server hardware to make them more "energy proportional" [4], as well as advances in the architecture of CDN itself. Our focus is on the CDN architecture, and more specifically, on its load balancing system. Recent work in server energy management

has suggested the technique of utilizing deep-sleep power saving modes or even completely turning off servers during periods of low load, thereby saving the energy expended by idle servers. We explore the potential applicability of this technique in the CDN context where it is important to understand the interplay of the three objectives below.

• Maximize energy reduction. Idle servers often consume more than 50% of the power of a fully-loaded one  This provides the opportunity to save energy by "rebalancing" (i.e., redirecting) the request traffic onto fewer servers and turning the remaining servers off.

• Satisfy customer SLAs. Content providers who are the CDN's customers would like their content and applications to be served with a high level of availability and performance to their clients. Availability can be measured as the fraction of client requests that are successfully served. A typical SLA would require at least "four nines" of end-to-end availability (i.e., 99.99%). To achieve this end-to-end SLA goal, we estimate that any acceptable technique for powering off servers should cause no more than a loss of 0.1 basis points of availability in the data center, leading us to target 99.999% server availability with our techniques. In addition to the availability SLA, the content providers also require good performance. For instance, clients downloading http content should experience small download times and clients watching media should receive high quality streams with high bandwidth and few freezes. Since turning off servers to save energy reduces the live server capacity used for serving the incoming request load, it is important that any energy saving technique minimizes the impact of the decreased capacity on availability and performance.

• Minimize server transitions. Studies have shown that frequently turning an electronic device on and off can impact its overall lifetime and reliability. Consequently, CDN operators are often concerned about the wear and tear caused by excessive on-off server transitions that could potentially decrease the lifetime of the servers. Additionally, when a server is turned off, its state has to be migrated or replicated to a different live server. Mechanisms for replicating content footprint and migrating long-standing TCP connections exist in the CDNs today as well as in other types of Internet scale services. However, a small degree of client visible performance degradation due to server transitions is inevitable. Consequently an energy saving technique should limit on-off server transitions in order to reduce wear and tear and the impact on client-visible performance. The three objectives above are often in conflict. For instance, turning off too many servers to maximize energy reduction can decrease the available live capacity of the CDN. Since it takes time to turn on a server and bring it back into service, an unexpected spike in the load can lead to dropped requests and SLA violations. Likewise, turning servers on and off frequently in response to load variations could enhance energy reduction but incur too many server transitions. Our goal is to design energy-aware techniques for CDNs that incorporate all three objectives and to understand how much energy reduction is realistically achievable in a CDN. Since CDNs are yet to be aggressively optimized for energy usage today, our work hopes to guide the future architectural evolution that must inevitably incorporate energy as a primary design objective. While we focus on CDNs, our work also applies to other CDN-like distributed systems that replicate services within and across server clusters and employ some form of load balancing to dynamically route requests to servers. On a different dimension, it is also important to note that our focus is energy usage reduction rather than energy cost reduction. Note that energy cost reduction can be achieved by dynamically shifting the server load to locations with lower energy prices without necessarily decreasing the total energy usage Research Contributions**.**

Our work is the first to propose energy-aware mechanisms for load balancing in CDNs with a quantification of the key practical tradeoffs between energy reduction, hardware wear-and-tear due to server transitions, and service availability that impacts customer SLAs. The load balancing system of a CDN operates at two levels . The global load balancing component determines a good cluster of the CDN for each request, while the local load balancing component chooses the right server for the request within the assigned cluster. We design mechanisms for energy savings, both from the local and global load-balancing standpoint. Further, we evaluate our mechanisms using real production workload traces collected over 25 days from 22 geographically distributed clusters across the US from a large commercial CDN. Our specific key contributions are as follows.

• In the offline context when the complete load sequence for a cluster is known ahead of time, we derive optimal algorithms that minimize energy usage by varying the number of live servers required to serve the incoming load

• On production CDN workloads, our offline algorithm achieves a significant system-wide energy reduction of 59.5%. Further, even if the average transitions is restricted to be below 1 transition per server per day, an energy reduction of 58.66% can be achieved, i.e., 98.6% of the maximum energy reduction can be achieved with minimal server wear and- tear.

• We propose a load balancing algorithm called Hibernate that works in an online fashion that makes decisions based on past and current load but not future load, much like a real life load balancing system. Hibernate achieves an energy reduction of 56%, i.e., within 94% of the offline optimal.

• By holding an extra 10% of the servers as live spares, Hibernate achieves the sweet spot with respect to all three metrics. Specifically, the algorithm achieves a system-wide energy reduction of 51% and a service availability of at least five nine's (99.999%), while incurring an average of at most 1 transition per server per day. The modest decrease in energy reduction due to the extra pool of live servers is well worth the enhanced service availability for the CDN.

• In a global flash crowd scenario when the load spikes suddenly across all clusters of the CDN, Hibernate is still able to provide five nine's of service availability and maintain customer SLAs as long as the rate at which load increases is commensurate with the percentage of server capacity that the algorithm keeps as live spares.

• Energy-aware global load balancing can redistribute traffic across clusters but had only a limited impact on energy reduction. Since load can only be redistributed between proximal clusters for reasons of client performance, these clusters had load patterns that are similar enough to not entail a large energy benefit from load redistribution. However, a 10% to 25% reduction in server transitions can be achieved by redistributing load across proximal clusters. But, perhaps the key benefit of global load balancing is significantly increased service availability. In our simulations, global load balancing enhanced service availability to almost 100%. In situations where an unpredictable increase in load would have exceeded the live capacity of a cluster causing service disruption, our global load balancing spread the load increase to other clusters with available live capacity. In summary, our results show that significant energy reduction is possible in CDNs if they are rearchitected with energy awareness as a first-order principle. Further, our work also allays the two primary fears in the mind of CDN operators regarding turning off servers for energy savings: the ability to

maintain service availability, especially in the presence of a flash crowd, and the impact of server transitions on the hardware lifetimes and ultimately the capital expenditures associated with operating the CDN. Roadmap**.** After formulating our models and methodology , we study local load balancing in an offline setting with the assumption that the entire traffic load pattern is known in advance and then extend it to the more realistic online situation where future traffic is unknown . Then, we explore the gains to be had by moving traffic between clusters via global load balancing
. Finally, we discuss related work and offer conclusions .

## II. MODEL FORMULATION AND METHODOLOGY

**CDN Model :**
Our work assumes a global content delivery network (CDN) that comprises a very large number of servers that are grouped into thousands of clusters. Each cluster is deployed in a single data center and its size can vary from tens to many thousands of servers. We assume that incoming requests are forwarded to a particular server in a particular cluster by the CDN's load balancing algorithm. Load balancing in a CDN is performed at two levels: global load balancing, where a user request is sent to an "optimum" cluster, and local load balancing, where a user request is assigned a specific server within the chosen cluster. Load balancing can be implemented using many mechanisms such as IP Any cast, load balancing switches, or most commonly, the DNS lookup mechanism [14]. We do not assume any particular mechanism, but we do assume that those mechanisms allow load to be arbitrarily re-divided and re-distributed among servers, both within a cluster (local) and across clusters (global). This is a good assumption for typical web workloads that form a significant portion of a CDN's traffic.

**Energy Model:**
Since our goal is to minimize energy usage, we model how servers consume energy as a function of load. Based on our own testing of typical off-the-shelf server configurations used by CDNs, we use the standard linear model where the power (in Watts) consumed by a server serving load $\lambda$ is
power($\lambda$) $\Delta=$
Pidle + (Ppeak − Pidle)$\lambda$, (1)
where the load $0 \leq \lambda \leq 1$ is the ratio of the actual load to the peak load, Pidle is the power consumed by an idle server, and Ppeak is the power consumed by the server under peak
load.We use typical values of 92 Watts and 63 Watts for Ppeak and Pidle respectively. Though we use the linear energy model above in all our simulations, our algorithmic results hold for any power function that is convex.
In addition to the energy consumed by live servers that are serving traffic, we also capture the energy consumed by servers that are in transition, i.e., either being turned off or tuned on.
Servers in transition cannot serve load but consume energy; this energy consumption is due to a number of steps that the CDN must perform during shutdown or startup. When a server
is turned off, the load balancing system first stops sending any new traffic to the server. Further, the CDN must wait until existing traffic either dies down or is migrated off the server. Additionally, the control responsibilities of the server would need to be migrated out by performing leader election and other relevant processes. Once the server has been completely isolated from the rest of the CDN, it can be powered down. When a server is turned on, these same steps are executed in the reverse. In both cases, a server transition takes

several minutes and can be done automatically by the CDN software To capture the energy spent during a transition, we model a fixed amount of energy usage of $\alpha$ Joules for each server transition, where $\alpha$ typically corresponds to 38 kilo Joules.

**Workload Model:**

The workload entering the load balancing system is modeled as a discrete sequence $\lambda t$, $1 \leq t \leq n$, where $\lambda t$ is the average load in the tth time slot. We always express load in the normalized unit of actual load divided by peak server capacity.1 Further, we assume that each time slot is $\delta$ seconds long and is large enough for the decisions made by the load balancing algorithm to take effect. Specifically, in our experiments, we choose a typical $\delta$ value of 300 seconds.

**Algorithmic Model for Load Balancing:**

While a real-life load balancing system is complex , we model only those aspects of such a system that are critical to energy usage. For simplicity, our load balancing algorithms redistribute the incoming load rather than explicitly route incoming requests from clients to servers. The major determinant of energy usage is the number of servers that need to remain live (i.e., turned on) at each time slot to effectively serve the incoming load. The exact manner in which load is distributed to those live servers is less important from an energy standpoint. In fact, in the linear energy model described in Equation 1, the precise manner in which load is distributed to the live servers makes
no difference to energy consumption.2 In reality, the precise manner in which the load is distributed to the live servers does matter greatly from the perspective of managing footprint and other server state. However, we view this a complementary problem to our own and methods exist in the research literature to tackle some of these issues. The local load balancing algorithm of a CDN balances load between live servers of a given cluster. In each time interval t, the algorithm distributes the load $\lambda t$ that is incoming to that cluster. Let mt denote the number of live servers in the cluster. Servers are typically not loaded to capacity. But rather a target load threshold $\Lambda$, $0 < \Lambda \leq 1$, is set such that the load balancing algorithm attempts to keep the load on each server of the CDN to no more than the fraction $\Lambda$ of its capacity. Mathematically, if li,t is the load assigned to live server i at time t, then _i=mt
i=1 li,t = $\lambda t$ and li,t $\leq \Lambda$, for $1 \leq i \leq$ mt. In addition to serving the current load, the load balancing algorithm also decides how many additional servers need to be turned on or off. The changes in the live server count made in time slot t is reflected in mt+1 in the next time slot.

The global load balancing algorithm works in an analogous fashion and distributes the global incoming load to the various server clusters. Specifically, the global incoming load is partitioned between the server clusters such that no cluster receives more than a fraction $\Lambda$ of its capacity. Further, clients are mapped to proximal clusters to ensure good performance.
Online versus Offline. The load balancing algorithms work in an online fashion where decisions are made at time t without any knowledge of the future load $\lambda t\_$ , $t\_ > t$. However, our work also considers the offline scenario where the load balancing algorithm knows the entire load sequence $\lambda t$, $1 \leq t \leq n$ ahead of time and can use that knowledge to make decisions.

The offline algorithms provide the theoretically best possible scenario by making future traffic completely predictable. Thus, our provably-optimal offline algorithms provide a key baseline to which realistic online algorithms can be compared.

**Metric Definitions:**

We are interested in the interplay of three metrics: energy reduction, service availability as it relates to customer SLA's, and server transitions. The energy reduction achieved by an algorithm that can turn servers on or off equals the percentage energy saved in comparison to a baseline where all servers remain turned on for the entire period. Since most CDNs today are not aggressively optimized for energy, the baseline is representative of the actual energy consumption of such systems. A server cluster that receives more load than the total capacity of its live servers cannot serve that excess load which must be dropped. The client requests that correspond to the dropped load experience a denial of service. The service availability over a time period is computed as 100 $*$ (total served load)/(total input load). Finally, the server transitions are expressed either as total amount over the time period, or as an average amount expressed as the number of transitions per server per day.

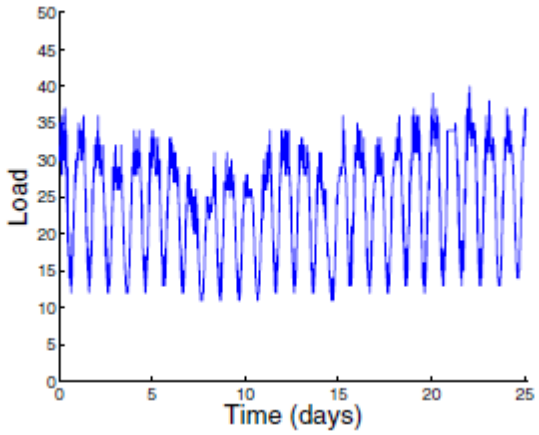



**Figure 14**: Load Vs Time

Fig. 1: Average load per server measured every 5 minutes across 22 Akamai clusters in the US over 25 days. Note load variations due to day, night, weekday, weekend, and holidays (such as low load on day no. 8, which was Christmas Empirical Data from the Akamai Network**.** To validate our algorithms and to quantify their benefits in a realistic manner, we used extensive load traces collected over 25 days from a large set of Akamai clusters (data centers) in the US. The 22 clusters captured in our traces are distributed widely within the US and had 15439 servers in total, i.e., a representative sampling of Akamai's US deployments. Our load traces account for a peak traffic of 800K requests/second and an aggregate of 950 million requests delivered to clients. The traces consist of a snapshot of total load served by each cluster collected every 5-minute interval from Dec 19th 2008 to January 12th 2009, a time period that includes the busy holiday shopping season for e-commerce traffic .

# 3.5 Optimised balancing algorithm for content delivery networks

## Introduction

A content delivery network (CDN) represents a popular and useful solution to effectively provide contents to users by adopting a distributed overlay of servers. By replicating content on several servers, a CDN is capable of partially solving congestion issues owing to high client request rates, thus reducing latency at the same time increasing content availability. Usually, a CDN consists of an original server (called 'backend server') containing new data to be diffused, together with one or more distribution servers, called 'surrogate servers'. Periodically, the surrogate servers are actively updated by the back-end server. Surrogate servers are typically used to store static data, whereas dynamic information (i.e. data that change in time) is just stored in a small number of back-end servers. In some typical scenarios there is a server called 'redirector', which dynamically redirects client requests based on selected policies. CDNs were born to improve accessibility, while maintaining correctness: this is achieved through content replication. They involve an orchestrated combination of heterogeneous techniques, such as content delivery, request routing, information spreading and accounting. The most important performance improvements derived from the adoption of such networks concern two aspects: (i) overall system throughput, that is the average number of requests served in a time unit (optimised also on the basis of the processing capabilities of the available servers); and (ii) response time experienced by clients after issuing a request The decision process about these two aspects could be in contraposition. As an example, a 'better response time' server is usually chosen based on geographical distance from the client, that is network proximity; on the other hand, the overall system throughput is typically optimised through load balancing across a set of servers. Although the exact combination of factors employed by commercial systems is not clearly defined in the literature, evidence suggests that the scale is tipped in favour of reducing response time. Akamai, LimeLight and CDNetworks are wellknown commercial CDN projects, which provide support to the most popular Internet and media companies, including BBC, Microsoft, DreamWorks, EA and Yahoo. Several academic projects have also been proposed, such as CoralCDN at New York University and CoDeeN at Princeton University, both running on the PlanetLab (http:// www.planet-lab.org/) testbed. A critical component of a CDN architecture is the request routing mechanism. It allows to direct users' requests for a content to the appropriate server, based on a specified set of parameters. The proximity principle, by means of which a request is always served by the server which is closest to the client, can sometime fail. Indeed, the routing process associated with a request might take into account several parameters (such as traffic load, bandwidth and servers' computational capabilities) in order to provide the best performance in terms of time of service, delay etc. Furthermore, an effective request routing mechanism should

be able to face temporary, and potentially localised, high request rates (the so-called ‘flash crowds’) in order to avoid affecting the quality of service perceived by other users. Depending on the network layers and mechanisms involved in the process, generally request routing techniques can be classified in domain name system (DNS) request routing, transport-layer request routing, and application-layer request routing . In a DNS-based approach, a specialised DNS server is able to provide a request balancing mechanism based on well defined policies and metrics . For every address resolution request received, the DNS server selects the most appropriate surrogate server in a cluster of available servers and replies to the client with both the selected internet protocol (IP) address and a time-to-live. The latter allows to define a period of validity for the mapping process. Typical implementations of this approach can provide either a single surrogate address or a record of multiple surrogate addresses, in the last case leaving to the client the choice of the server to contact (e.g. in a round-robin fashion). With transport-layer request routing, a layer 4 switch usually inspects information contained in the request header in order to select the most appropriate surrogate server. Information about the client’s IP address and port (and more generally all layer 4 protocol data) can be analysed. Specific policies and traffic metrics have been defined for a correct server selection. Generally, the routing to the server is achieved either by rewriting the IP destination of each incoming packet, or by a packet tunnelling mechanism, or by a forwarding mechanism at the MAC layer. With application-layer request routing, the task of selecting the surrogate server is typically carried out by a layer 7 application, or by the contacted web-server itself. In particular, in the presence of a web-server routing mechanism the server can decide to either serve or redirect a client request to a remote node. Differently from the previous mechanism, which usually needs a centralised element, a web-server routing solution is usually designed in a distributed fashion. ‘URL rewriting’ and ‘HTTP redirection’ are typical solutions based on this approach. In the former case, a contacted server can dynamically change the links of embedded objects in a requested page in order to let them point to other nodes. The latter technique, instead, exploits the redirection mechanism of the HTTP protocol to appropriately balance the load on several nodes. In the following of this paper we will focus our attention on the application-layer request routing mechanism. More precisely, we will provide a solution for load balancing in the context of the HTTP redirection approaches. We present a new mechanism for redirecting incoming client requests to the most appropriate server, thus balancing the overall system requests load. Our mechanism leverages local balancing in order to achieve global balancing. This is carried out through a periodic interaction among the system nodes. The rest of the paper is organised as follows. We will briefly describe some interesting solutions for load balancing in Section 2. Our solution for request balancing will be described in Section 3. Simulation results of the algorithm will be provided in Section 4. We will provide concluding remarks in Section 5.

# Fictitiously starred optimised balancing (FSOB)

In this section we will present a novel algorithm for balancing client requests among several servers: the FSOB algorithm. Before describing the details of our algorithm we briefly discuss some models for distributed load balancing. With a distributed approach, every node usually consists of a scheduler, a buffer for holding requests and a processor. A request might either be served locally or assigned to a remote server based on the specific balancing strategy. Depending on how the scheduler interacts with the other components of the node, it is possible to classify the balancing algorithms in three fundamental models : a 'queue-adjustment model', a 'rate-adjustment model' and a 'hybrid-adjustment model' (Fig. 1). In a queue-adjustment strategy the scheduler is located after the queue and just before the server. The scheduler might assign the request pulled out from the queue to either the local server or a remote server depending on the status of the system queues: if an unbalancing exists in the network with respect to the local server, it might assign part of the queued requests to the most unloaded remote server. In this way the algorithm tries to equally balance the requests in the system queues. It is clear that in order to achieve an effective load balancing the scheduler needs to periodically retrieve information about remote queue lengths. In a rate-adjustment model, instead, the scheduler is located just before the local queue: upon arrival of a new request the scheduler decides whether to assign it to the local queue or send it to a remote server. Once a request is assigned to a local queue no remote rescheduling is allowed. Such strategy usually balances the request rate arriving at every node independently from the current state of the queue. No periodical information exchange, indeed, is requested. In a hybrid-adjustment strategy for load balancing the scheduler is allowed to control both the incoming request rate at a node and the local queue length. Such approach allows to have a more efficient load balancing in a very dynamic scenario, but at the same time it requires a more complex algorithm. In the context of a hybrid-adjustment mechanism, the queue adjustment and the rate adjustment might be considered, respectively, as a fine-grained and a coarse-grained process. The algorithm for load balancing proposed in this paper falls in the class of rate-adjustment approaches. Differently from most of the previous algorithms, we propose a highly dynamic distributed strategy based on the periodical exchange of information about the status of the nodes, in terms of load. By exploiting the multiple redirection mechanism offered by HTTP, our algorithm tries to achieve a global balancing through a local request redistribution process. Upon arrival of a new request, indeed, a CDN server can either elaborate locally the request or redirect it to other servers according to an optimised decision rule, which is based on the state information exchanged among the servers. Let us consider several CDN servers which are 'virtually' connected into an overlay network. We remark that the consistency of the contents is out of the scope of this paper: we suppose that any server is able to serve a request independently from its specific content. FSOB is implemented locally at any node. With such algorithm, the server assumes itself to be the centre (i.e. the 'master') of a 'fictitious star topology'. In Fig. 2, for example, the star topology for the master node B is reported Any time a request arrives at a server, it can decide to either elaborate it or redistribute it, based

on a proper policy, to one of the peers (i.e. the ‘slaves’) in the above described fictitious star topology. Under the assumptions above, any master is capable to ‘trace out’ the loads of the slaves, since, in the master’s view, they exclusively serve requests redirected from it. In this way, an optimised rule for load balancing can be implemented for each fictitious star topology. Since we assume that all servers have the same service rate, we can distribute the load among them based just on the queue length at each server: a new request is assigned by the master to the server, including itself, which has the least queue length value at that point in time . Periodically, the slaves report to the master their queue length. For each request assigned, the master updates the expected value of the queue length of the server which the request is sent to by incrementing it by one. Yet, the real behaviour of the load of each server is different as we assumed fictitious assumptions. In particular, the queue length can significantly change over time with respect to the expected value, owing to local requests traffic at the slave node. The periodic information provided by the servers in the CDN allows to restore the correct value of the slave loads. Although the redistribution mechanism always assigns a new request to the least loaded (in terms of queue length) server, it is possible to identify two alternative behaviours of the algorithm with respect to network load conditions: an ‘equalising’ phase and a ‘cycling’ phase.

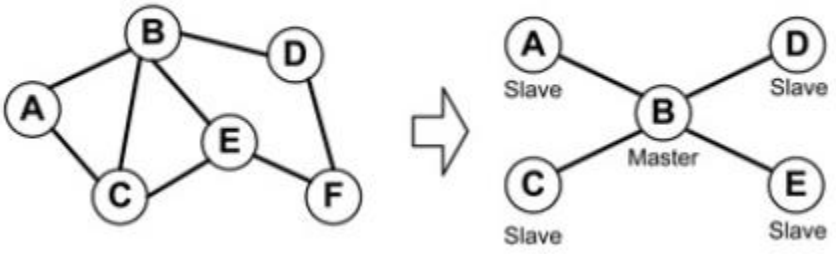



**Figure 15** :FSOB Topology

In the former the algorithm tends to equalise the estimated server loads in the presence of an unbalanced distribution of queue lengths, by assigning new requests to the server with the least loaded queue. Once the load is supposed to be equally distributed among all the servers in the network, the algorithm assumes a 'cycling' behaviour, where the master tends to assign new incoming requests in a manner which approximates a round-robin policy. Once done with server updates, the
master enters again the equalising phase. The sequence of the two phases and their respective durations depend on the existing degree of unbalancing among the master and the slaves. The distinctive feature of this algorithm is its intrinsic capability to adjust itself to the initial condition of the server loads. Indeed, if the load is already balanced the algorithm exhibits a round-robin behaviour for the requests distribution, thus simply preserving the initial equilibrium. Otherwise, if the initial load is extremely unbalanced, the algorithm tends to equalise it by assigning new incoming
requests to the least loaded server. To further improve the estimation of the slaves load, we add to FSOB an additional estimation of the expected queue behaviour relying also on the arrival and service rates experienced locally at the server. Any server in the CDN, indeed, might experience different arrival rates as well as service rates over time. Such information is 'sampled' locally at any slave node and sent during the updating process together with the queue length value. The master node can use this information to optimally estimate the supposed behaviour of the slaves load.

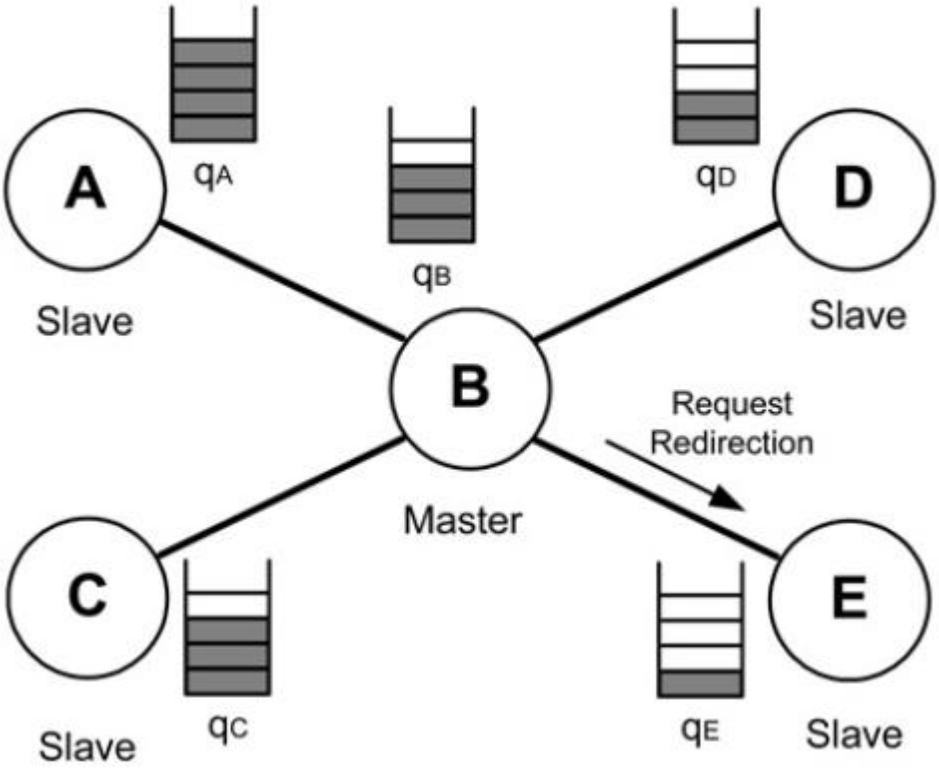



**Figure 16**: FSOB Algorithm

Any slave computes both its expected arrival and service rates by an exponential weighted moving average

$\hat{r}(kT) = (1 - a)\hat{r}((k - 1)T) + ar(kT)$ (1)

where $\hat{r}$ is the expected rate, r is the rate sampled by the slave and T is the updating time. Based on such values the slave computes the difference

$D\hat{r}(kT) = \hat{l}(kT) - \hat{m}(kT)$ (2)

where $\hat{l}$and $\hat{m}$ are, respectively, the estimations of arrivaland service rates computed by formula (1). After every interval of duration T, the slave provides the master with the local load and $D\hat{r}$ measures. Such information is exploited to estimate the expected load $\hat{q}$ of the slave at time kT + Dt by means of the following equation

$\hat{q}(kT + Dt) = qs(kT) + Ns$

$[kT,kT+Dt]$

$+ D\hat{r}(kT)Dt$ (3)

where qs is the queue length provided by the slave during the updating process at time kT, and Ns

$[kT,kT+Dt]$ is the number of requests assigned to such a slave during the interval $[kT, kT + Dt]$.

In Fig. 4 we report the pseudo-code of the FSOB algorithmby highlighting its main events.

# Chapter 4: Proposed model

## 4.1Tomcat:

Tomcat is an application server from the Apache Software Foundation that executes Java servlets and renders Web pages that include Java Server Page coding. Described as a "reference implementation" of the Java Servlet and the Java Server Page specifications, Tomcat is the result of an open collaboration of developers and is available from the Apache Web site in both binary and source versions. Tomcat can be used as either a standalone product with its own internal Web server or together with other Web servers, including Apache, Netscape Enterprise Server, Microsoft Internet Information Server (IIS), and Microsoft Personal Web Server. Tomcat requires a Java Runtime Enterprise Environment that conforms to JRE 1.1 or later.

## 4.2 Tomcat Clustering:

Apache Tomcat is a great performer on its own, but if you're expecting more traffic as your site expands, or are thinking about the best way to provide high availability, you'll be happy to know that Tomcat also shines in a clustered environment. With built-in support for both synchronous and asynchronous in-memory and external session replication, cluster segmentation, and compatibility with all common load balancing solutions, your Tomcat servers are ready for the cluster right out of the box.

Although clustering can seem like a complicated topic, the premise is quite simple. A clustered architecture is used to solve one or more of the following problems:

- A single server cannot handle the high number of incoming requests efficiently
- A stateful application needs a way of preserving session data if its server fails
- A developer requires the capability to make configuration changes or deploy updates to their applications without discontinuing service.

A clustered architecture solves these problems using a combination of load balancing, multiple server "workers" to process the balanced load, and some kind of session replication. Depending on the needs of the application, only some of these components may be used, or additional components such as caching and compression engines

Each tomcat has deployed the same web application. so any tomcat can process the client request. If one tomcat is failed, then other tomcat in the cluster to proceeds the request.

Here one big problem is arrive. each tomcat instances are running in dedicated physical machine or many tomcat instances are running in single machine so each tomcat running on different port and may be in different IP.

The problem is in client perspective, to which tomcat we need to make the request? because there are lots of tomcat part of clustering is running. each tomcat we need to make IP and Port combination. like http://192.168.56.190:8080/
or http://192.168.56.191:8181/

To add one server in-front of all tomcat clusters. to accept all the request and distribute to the cluster. so this server acts as a load balancer. There is lots of server is available with load balancing capability. here we are going to use Apache httpd web server as a load balancer. with mod_jk module. so now all clients to access the load balancer (Apache http web server) and don't bother about tomcat instances

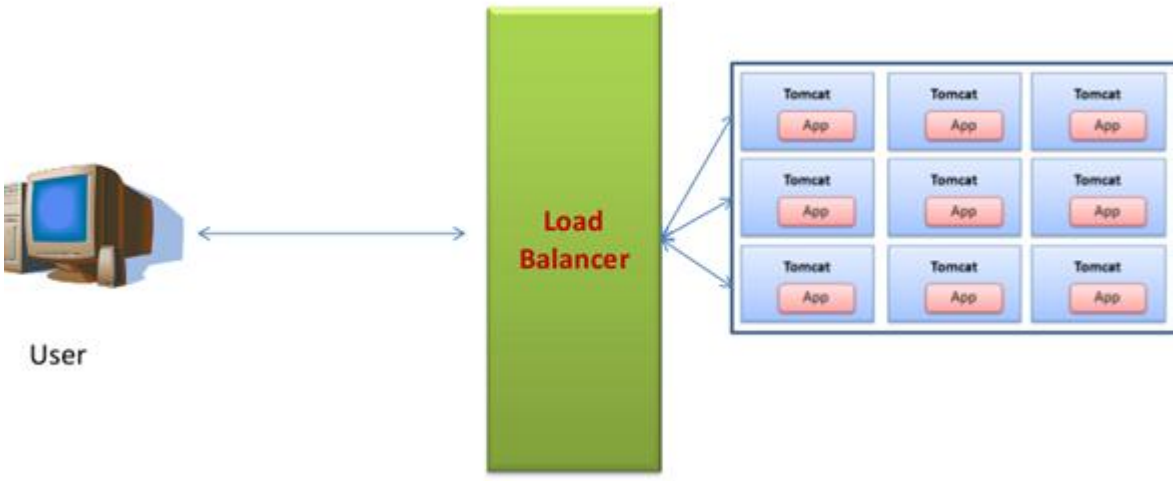



**Figure 17:** Apache load balancer between servers and the user

## 4.3 Apache httpd Web Server

Here we are going to use Apache httpd web server as a Load Balancer. To provide the load balancing capability to Apache httpd server we need to include the either mod_proxy module or mod_jk module. here we are using mod_jk module.

## 4.4 How to setup the Simple Load Balancer

For simplicity purpose I am going to run 3 tomcat instances in single machine(we can run on dedicated machine also) with Apache httpd web server. and single web application is deployed in all tomcat instances.

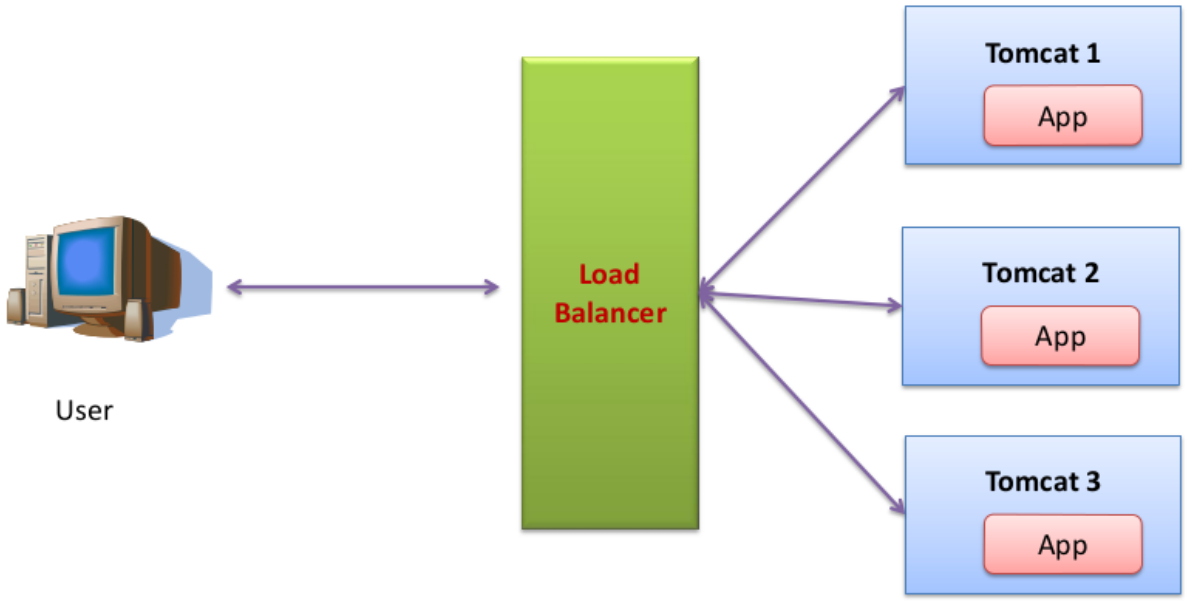



**Figure 18:** load balancer controlling three servers

Here we use mod_jk module as the load balancer. by default its use the round robin algorithm to distribute the requests. now we need to configure the workers.properties file like  virtual host concept in Apache httpd server.

```
worker.tomcat1.type=ajp13
worker.tomcat1.port=8009
worker.tomcat1.host=localhost

worker.tomcat2.type=ajp13
worker.tomcat2.port=8010
worker.tomcat2.host=localhost

worker.tomcat3.type=ajp13
worker.tomcat3.port=8011
worker.tomcat3.host=localhost
worker.list=tomcat1,tomcat2,tomcat3
```

here i configure the 3 tomcat instances in workers.properties file. here type is ajp13 and port is ajp port (not http connector port) and host is IP address of tomcat instance machine.

there are couple of special workers we need add into workers.properties file.

First one is add load balancer worker, here the name is  balancer (u can put any name).

```
worker.balancer.type=lb
worker.balancer.balance_workers=tomcat1,tomcat2,tomcat3
```

here this worker *type* is lb, ie load balancer. its special type provide by load balancer. and another property is  balance_workers to specify all tomcat instances like tomcat1,tomcat2,tomcat3 (comma separated)

Second one, add the status worker, Its optional. but from this worker we can get statistical of load balancer.

.

worker.list=balancer,stat

so from outside there are 2 workers are visible (balancer and stat). so all request comes to balancer. then balancer worker manage all tomcat instances.

Complete workers.properties file

```
worker.list=balancer,stat

  worker.tomcat1.type=ajp13
  worker.tomcat1.port=8009
  worker.tomcat1.host=localhost

  worker.tomcat2.type=ajp13
  worker.tomcat2.port=8010
  worker.tomcat2.host=localhost

  worker.tomcat3.type=ajp13
  worker.tomcat3.port=8011
  worker.tomcat3.host=localhost


  worker.balancer.type=lb
  worker.balancer.balance_workers=tomcat1,tomcat2,tomcat3

  worker.stat.type=status
```

Now workers.properties confiuration is finished. now we need to send the all request to balancer worker.so modify the httpd.conf file of Apache httpd server

```
LoadModule    jk_module  modules/mod_jk.so

        JkWorkersFile conf/workers.properties

        JkLogFile     logs/mod_jk.log
        JkLogLevel    emerg
        JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
        JkOptions     +ForwardKeySize +ForwardURICompat -ForwardDirectories
        JkRequestLogFormat     "%w %V %T"

        JkMount  /*  balancer
```

the above code is just boiler plate code. 1st line load the mod_jk module, 2nd line to specified the worker file (workers.properties file). all others are just logging purpose.

The last 2 lines are important.
JkMount /status stat means any request to match the /status then that request forward to stat worker. Its status type worker. so its shows status of load balancer.

JkMount /* balancer this line matches all the request, so all request is forward to balancer worker. In balancer worker is uses the round robin algorithm to distribute the request to other tomcat instances.

That's it.
now access the load balancer from the browser. each and every request is distribute to 3 tomcat instances. If one of the tomcat instances are failed then load balancer dynamically understand and stop to forward the request to that failed tomcat instances. Other tomcat instances are continue to work. If that failed tomcat is recovered from failed state to normal state then load balancer add to cluster and forward the request to that tomcat

Here big question is How Load balancer knows when one tomcat instance is failed or tomcat is just recovered from failed state?

Ans **:** when one tomcat instance is failed, load balancer don't know
about that instances is failed. so its try to forward the request to all tomcat instances. If load balancer try to forward the request to failed tomcat instance, its will not respond. so load balancer understand and marked the state as a failed and forward the same request to another tomcat instances. so client perspective we not feel one tomcat instances are failed.

when tomcat instances recovered from failed state. that time also load balancer don't

know that tomcat is ready for processing. Its still marked the state is failed.
In periodic interval load balancer checks the health status of all tomcat instances. (by default 60 sec). after checking health status then only load balancer came to know that tomcat instance is ready. and its update the status is OK.

## 4.5 Session Affinity Load Balancer

This second part of the Tomcat Clustering Series . In my first part we discuss about how to setup simple load balancer. and we seen how load balancer distribute the request to tomcat instance in round robin fashion.

Now we will discuss about, what is the problem that occurs in simple load balancer when we introduce sessions in our web application. and we will see how to resolve this issue.

How Session works in Servlet/Tomcat?
Before going into problem, let see the session management in Tomcat .If any if the page/servlet create the session then Tomcat create the Session Object nd attached into group of session (Hash Map kind structure) and that session can identify using ssion-id, it's just random number generated through any one of the hash algorithm. then respond to client with cookie header field. That cookie header field are key value pair. so tomcat create jsessioid is the key and random session-id is the value.

Once response reached to client (Web Browser) its update the cookie value. If already exist, then its overrides the cookie value. Then all further communication browser send the cookie attached with request to that server.

HTTP is stateless protocol. so server can't find the client session usual way. so server reads the header of the request and extract the cookie value and server got the Random session - id. then it search through group of session maintained by the tomcat. (It usually hash-map). then tomcat got perfect Session of that particular client (Web Browser).

If client cookie value doesn't match with group of sessions then tomcat create the completely new session and send the new cookie to browser. then browser update it. this index.jsp code to deploy all tomcat instances

```
<%@page import="java.util.ArrayList"%>
<%@page import="java.util.Date"%>

<%@page import="java.util.List"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>



 <span style="font-size:12px;background-color:transparent;line-height:16px;">
                    Instance 1


</span>

 <hr style="background-color:transparent;border-bottom-style:none;border-bottom-
width:0px;border-left-style:none;border-left-width:0px;border-right-style:none;border-right-
width:0px;border-top-style:none;border-top-
width:0px;margin:1px 1px 1px 1px;"><span style="font-size:12px;background-
color:transparent;line-height:16px; word-break:break-word; ">


Session Id : <%=request.getSession().getId()%>

Is it New Session : <%=request.getSession().isNew()%>

Session Creation Date : <%=new Date(request.getSession().getCreationTime())%>

Session Access Date : <%=new Date(request.getSession().getLastAccessedTime())%>


</span>
<b style="background-color:transparent;font-weight:bold;">Cart List </b>
```

<hr style="background-color:transparent;border-bottom-style:none;border-bottom-width:0px;border-left-style:none;border-left-width:0px;border-right-style:none;border-right-width:0px;border-top-style:none;border-top-width:0px;margin:1px 1px 1px 1px;">

Details:

1. User request one web page, in that web page its used sessions (like shopping cart).
2. Load balancer intercept the request and use the round robin fashion its send to one of the tomcat. suppose this time its send to tomcat1.
3. tomcat1 create the session and respond with cookie header to client
4. load balancer just act as relay. its send back to client
5. next time user request again the shopping cart to server. this time user send the cookie header also

6. .Load balancer intercept the request and use the round robin fashion its send to one of the tomcat. this time its send to tomcat2.
7. Tomcat 2 receive the request and extract the session-id. and this session id is doesn't match with their managed session. because this session is available only in tomcat1. so tomcat 2 is create the new session and send new cookie to client
8. Client receive the response and update the cookie(Its overwrite the old cookie).
9. Client send one more time to request that page and send the cookie to server.
10. Load balancer intercept the request and use the round robin fashion its send to one of the tomcat. this time its send to tomcat3.
11. Tomcat 3 receive the request and extract the session-id. and this session id is doesn't match with their managed session. because this session is available only in tomcat2. so tomcat3 is create the new session and send new cookie to client
12. Client receive the response and update the cookie. (Its overwrite the old cookie).
13. Client send one more time to request that page and send the cookie to server.
14. Load balancer intercept the request and use the round robin fashion its send to one of the tomcat. this time its send to tomcat1.
15. Tomcat 1 receive the request and extract the session-id. and this session id is doesn't match with their managed session. because client session id is updated by tomcat 3 last time. so even though tomcat 1 have one session object created by this client. but client session id is wrong. So tomcat1 is create the new session and send new cookie to client
16. Client receive the response and update the cookie.

this sequence is continue ...

as the result every request one session is created. instead of continue with old one.

Here root cause is Load balancer . If load balancer redirect the request correctly then this problem is fixed. but how load balancer know in advance about this client before is processed by particular tomcat.
HTTP is stateless protocol. so HTTP doesn't help this situation. and other information is jsessionid cookie. its good but its just random value. so we can't take decision based on this random value.
ex:

Cookie: JSESSIONID=40025608F7B50E42DFA2785329079227

**Session affinity/Sticky Session**

Session affinity overrides the load-balancing algorithm by directing all requests in a session to a specific tomcat server. so when we setup the session affinity our problem is solved. but how to setup because session values are random value. so we need to generate the session value some how identify the which tomcat generate response.

jvmRoute

- Tomcat configuration file (server.xml) caontain <Engine> tag have jvmRoute property for this purpose. so edit the config file and update the <Engine > tag like this

  <Engine name="Catalina" defaultHost="localhost“ jvmRoute=“tomcat1” >

- Here we mention jvmRoute="tomcat1" here tomcat1 is worker name of this tomcat. check the workers.properties file in last post.

- Add this line to all tomcat instances conf/server.xml file and change the jvmRoute value according to workers name and restart the tomcat instances.

- Now all tomcat generate the session-id pattern like this
  <Random Value like before>.<jvmRoute value>

  ex: tomcat1 generate the session id like
  Cookie:JSESSIONID=40025608F7B50E42DFA2785329079227.tomcat1

here tail have which tomcat generate the session. so load balancer easily find out to where we need to delegate the request. in this case its tomcat1.

so update all tomcat instances conf/server.xml file to add the jvmRoute property to appropriate worker name values. and restart the instances. all problem is fixed and entire load balance works fine even session based application.

but there is still one drawback

if 5 user acessing the website. In session affinity is setup. here

tomcat 1 serves 2 user,
tomcat 2 serves 2 user,
tomcat 3 serves 1 user, then suddenly one of instance is failed, then what happen?

suppose instance 1 (tomcat1) is failed, then those 2 users lost their session. but their request are redirect to one of the remaining tomcat instances (tomcat2,tomcat3). so they still access the web page. but they lost previous sessions. this is one of the draw back. but its compare to last post load balancer. its works in session based web applicationalso.
Next post we will see how to set up the session replication in load balancer.

**Session Replication**

Hi this is my third part of the Tomcat Clustering Series . In this post we are going to discuss the how to setup session replication in tomcat clustering environment. Session replication makes High availability and full fail-over capability to our clustering environment.**[**

In my previous post we discussed about setup simple load balancer and how to makesession affinity concepts.

How to setup Session Replication in tomcat
before going to session replication we need to understand 2 important concepts

- Multicast
- Session Manager in Tomcat

**Multicast**
Multicast is To transmit a single message to a select group of recipients.
here multicast used by tomcat cluster to identify the instances those part of cluster.

There is 2 types of cluster
- Static Tomcat Cluster
- Dynamic Tomcat Cluster

In static cluster there is no need multicast, because each tomcat we statically defined/configured the other instances. But dynamic Cluster we are
not defined anything. so each tomcat in that cluster some how to identify the other tomcat instances.

so here multicast concepts is used. each and every tomcat first joining to single multicast group. and send the heartbeat signals in periodic interval. so other tomcat instances received these signal and add the member to the cluster.

**Session Manager in Tomcat**

Session Manager is used to create and manage the session behalf the application. In Servlet Specification request.getSession(); line is mention that container (tomcat) is responsible for create the session. here tomcat use the Session Manager for this purpose.

there is 4 types of Session Manager

- Standard Manager
- Persistent Manager
- Delta Manager
- Backup Manager

**Standard Manager**

It is the default manager used by tomcat. Even though we are not mention in our web application tomcat use this manager for managing our session. If u want to customize the this standard manager then add <Manager> tag in context.xml file.

<Manager className="org.apache.catalina.session.StandardManager" />

here org.apache.catalina.session.StandardManager is fully qualified class name of the Standard Manager.

**Persistent Manger**

This manger is to sote the session information into persistent place after some interval. here two types of store is available.

- File Store
- JDBC Store

**File Store**

It helps to store all session information in separate files in underlying file system (local HDD or shared file-system like NFS,..)

**JDBC Store**

It helps to store the session information to relational database. so using the Persistent Manager we can achieve the tomcat cluster. but its not swapped out in real time. its pushes the information after certain interval. so if anything badly happen(crash) before that interval then in-memory session data is gone.

**Delta Manger**

In this post we are going to use this manager. Its replicate the session to all other instances. so this manager usually used clustered environment. but not good for large cluster.

**Backup Manager**

This manager usually used clustered environment. It's like delta manger. but it will replicate to exactly one other instance(backup instance). Its acted like one instance is Primary and another instance as backup

Steps to make Session Replication in Tomcat Clustering

1. Enable Multicast routing
2. Add <Cluster> Entries in conf/server.xml file for all instances.
3. Enable the Web Application as distributable

**1. Enable Multicast routing**

In Linux Environment most of the system kernel is capable to process the multicast address. but we need to add route entry in kernel routing table.

sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0

Here eth0 is my Ethernet interface. so change according to your interface
In multicast address is belong to Class D address Range (224.0.0.0 to 239.255.255.255).so we inform to kernel if any one access these address then it goes through eth0 interface.

**2. Add <Cluster> Entries in conf/server.xml file for all instances.**

This very important part for tomcat clustering. We need to Add <Cluster> tag inconf/server.xml file in all tomcat instances.

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

We can add this <Cluster> tag in either inside the<Engine> tag or <Host> tag.
here SimpleTcpCluster is Tomcat Cluster implementation

This tag is looks like simple but its has many inner tags. if we omitted then its takes the default values. if we want do any cutomization (like change multicat address, receving address port) we need to use complete <Cluster> tag

this is complete <Cluster>

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
                channelSendOptions="8">


 <Manager className="org.apache.catalina.ha.session.DeltaManager"
 expireSessionsOnShutdown="false"
 notifyListenersOnReplication="true"/>
 <Channel className="org.apache.catalina.tribes.group.GroupChannel">
 <Membership className="org.apache.catalina.tribes.membership.McastService"
 address="228.0.0.4"
 port="45564"
 frequency="500"
 dropTime="3000"/>

 <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
 <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
 </Sender>
 <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
 address="auto"
 port="4000"
 autoBind="100"
 selectorTimeout="5000"
 maxThreads="6"/>
 <Interceptor

className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
 <Interceptor

className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"/>
 </Channel>
```

```
 <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>

 <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

 <ClusterListener
className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>

 <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>

</Cluster>
```

here most of the code are  boiler plate code. just copy and paste. if we need we can customize. for example we can change the multicat address and port number.

<Manager className="org.apache.catalina.ha.session.DeltaManager"/>

here Manager tag define the delta manager. Delta manager means replicate to all instances.

<Channel className="org.apache.catalina.tribes.group.GroupChannel">
Tomcat Clustering use the Apache Tribes communication framework .  This group commnication framework is responsible for dynamic membership (using multicast) , send and receive the session delta information using uni-cast (normal TCP connection).

```
<Membership className="org.apache.catalina.tribes.membership.McastService"
 address="228.0.0.4"
 port="45564"
 frequency="500"
 dropTime="3000"/>
```

This is Membership definition. here address is multicast address. we can pick any address from Class D address range (224.0.0.0 to 239.255.255.255)and any port number.

Each and every tomcat send the heart beat signal to multicast address in periodic (frequency ) interval. all other tomcat whose joined the multicast address they can receive these signals and add the membership to the cluster. if heat beat signal is not revive some particular interval ( dropTime ) from any one of the tomcat, then we need to consider that tomcat is failed.

Note:-
All tomcat instances which is part of the clustering, should have same multicast address and port number.

```
<Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
 <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
 </Sender>
```

here sender use the PooledParallelSender have pooled connections to use the send the session information concurrently. so its speedup the session replication process.

```
 <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
address="auto"
port=" 4000 "
autoBind="100"
selectorTimeout="5000"
maxThreads="6"/>
```

here we define which port Receiver can bind and used for receiving the session replicate information. here two properties are important. address and port. here address is ur system IP address and port is any unused port. here address="auto" its automatically pick the system IP address.

We have some interceptor
TcpFailureDetector -Its ensure that instance are dead. In some case multicast messages are delayed, all tomcat instances are think about that tomcat is dead. but this interceptor to make tcp unicast to failed tomcat and ensure that instances is actually failed or not

another important listener is JvmRouteSessionIDBinderListener, we talk about later

**3. Enable the Web Application as distributable**

We need to make the our web application distribuable. its simple add <distributable/> tag in web.xml file. In according to servlet specification <distributable/> tag in web.xml mention that any container to consider this application can work in distributed environment.

Note:
All session in our web application must me serializable.
Do these steps to all tomcat instances and start the tomcat and httpd server. check my configuration in my github repo or get as ZIP

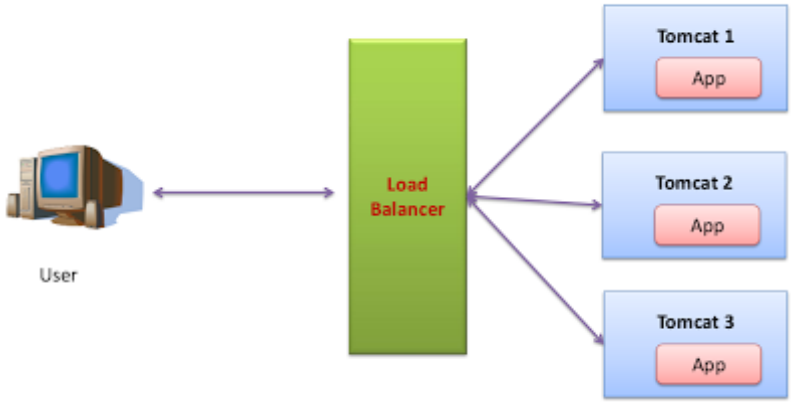



**Figure 19**: Configured Tomcat instances

This is my configuration. all 3 tomcat instances are configured in delta manager and i deployed the distributed web application. all tomcat use the multicast to maintain the membership.

now client make the request and first tomcat process and create the session, then looks like this

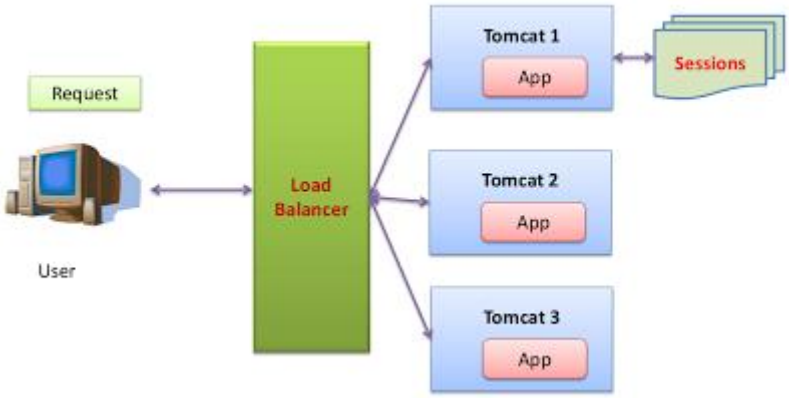



**Figure 20:** Use of Tomcat 1

then tomcat 1 is responsible to replicate the session using Apache tribes group communication framework to replicate the session to all instances.

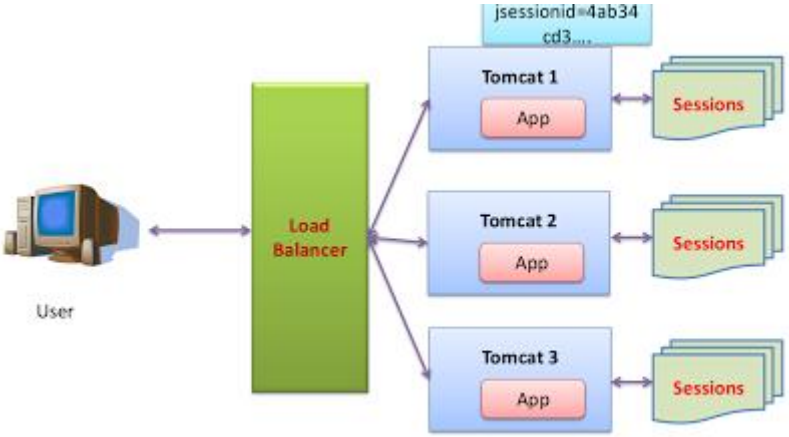



**Figure 21:** Working of all tomcat servers

Now all tomcat instance have exact copy of the session. so if tomcat 1 crashed or shutdown, then any other tomcat still can process the request [see the video below]

We used session affinity like previous post . based on that cookie id contain the tomcat name (worker name). so when first tomcat1 return the session id end with tomcat1. but when tomcat 1 is failed and tomcat 2 take the responsible for all further request. but session id still contain the tomcat1. so its makes the load balancer difficult. because tomcat1 is down. and load balancer pick any other tomcat. but actually tomcat2 takes the responsible. so we need to reflect these changes in session id.
JvmRouteSessionIDBinderListener take care to change the client session id to tomcat2 when failure is occurred so load balancer redirect to tomcat2 without confusing.

Check the git hub for all configuration files and tomcat clustering setup is available.

**Session Replication using Backup Manager**

Hi this is my fourth part of the Tomcat Clustering Series . In this post we are going to discuss the how to setup session replication using Backup Manager in tomcat clustering environment. Session replication makes High availability and full fail-over capability to our clustering environment.
Its continuation of the last post (session replication using Delta Manager) . In delta manager each tomcat instance need to replicate the session information to all other tomcat instances. Its take more time and replication if our cluster size is increased. so there is alternative manager is there. Its Backup Manager.

Backup Manager is replicate the copy of session data to exactly one other tomcat instances. This big difference between both managers. here which tomcat creates that is primary copy of the session. and another tomcat whose hold the replicate session is backup copy . If any one of the tomcat is down. back up tomcat serve the session.
Its achieve the fail over capability.

The setup process of backup manager is same as Delta manager. except we need to mention the Manager as BacupManager
( org.apache.catalina.ha.session.DeltaManager ) inside <Cluster> element.
Suppose we have 3 tomcat instances like previous post . and i configured into backup manager.
Now user try access the page. User request comes to load balancer, and load balancer redirect the rquest to suppose tomcat1. Now tomcat one create the session, now tomcat1 is responsible to replicate exactly one copy to any one of the tomcat.
so tomcat1 picks any tomcat which is part of the cluster (multicast). here tomcat1 picks tomcat3 as a backup. so tomcat3 hold the backup copy of the session.

we run the load balancer in sticky session mode. so all further request from that particular user is redirect to tomcat1 only. all modification in tomcat1 is replicate to tomcat3.

now tomcat1 is crashed/shutdown for some reason
Now same user try to access the page. this time load balancer try to redirect to tomcat1. but tomcat1 is down. so load-balancer pick one tomcat from the remaining tomcats. here interestingly 2 case are there.

**Case 1:**
Suppose Load balancer pick the tomcat3 then tomcat3 receive the request and tomcat3 itself hold the backup copy of the session. so tomcat3 make that session as primary copy and tomcat3 pick any one tomcat as backup copy. so here remaining only one tomcat is there. so tomcat3 replicate the session to tomcat2. so now tomcat3 hold primary copy and tomcat2 hold the backup copy. now tomcat3 give the response to user. all further request is handled by tomcat3 (sticky session).

**case 2:**
Suppose Load balancer pick the tomcat2 then tomcat2 receive the request and tomcat2 don't have the session. so tomcat2 session manager (Backup Manager) ask to all other tomcat manager "hi anybody hold the session for this user (based on session id [cookie])". Actually tomcat 3 have the backup session. so tomcat3 inform to tomcat2. and replicate the session to tomcat2. now tomcat2 make that session as primary copy and tomcat3 whose already have copy of session as remains as a backup copy of that session. so now tomcat2 hold primary copy and tomcat3 hold the backup copy. now tomcat2 give the response to user. all further request is handled by tomcat2 (sticky session).so in either case our session is replicate and maintained by backup manager. It's good for large cluster.

Note:

- Load balancer also faces single point failure . to resolve this we need to put another load balancer with public address and update the new IP to DNS server with same URL. so our URL like example.com query resolves the 2 IP address for 2 load balancer.

- How its work:

- If browser want to access http://example.com then it first ask DNS server.
- DNS server gives 2 IP address to browser
- Browser take the first IP address and try to connect.
- If in the case that server is failed to respond then browser side make timeout

- Then browser contact second IP address, now second load balancer is works fine.

  This kind of adding more load balancer makes to our website more scalable and reliable in case of tragedy

It gives 11 IP address. these all are google load balancer's located in various geographic locations

Another thing this DNS servers not return same order of IP list to browser. each an every time its rotate(round robin) the IP list. so in 2 different machine ask google.com get different order of IP list. so these 2 different machine connect different Google load balancer. so here DNS server also play little role for distribute the requests(loads).to verify use same command twice. and verify the IP order.

# Chapter 5: Experiments and Results

I have created three tomcat servers by the name of instance 1 , instance 2 and instance 3.Each server has its own session id and session time out. All the three servers can work on the single system at the same time ,without affecting the performance of other server. Each server has a session time out of one minute.I have created the web page for each server to enter the data, as you can see in these three figures

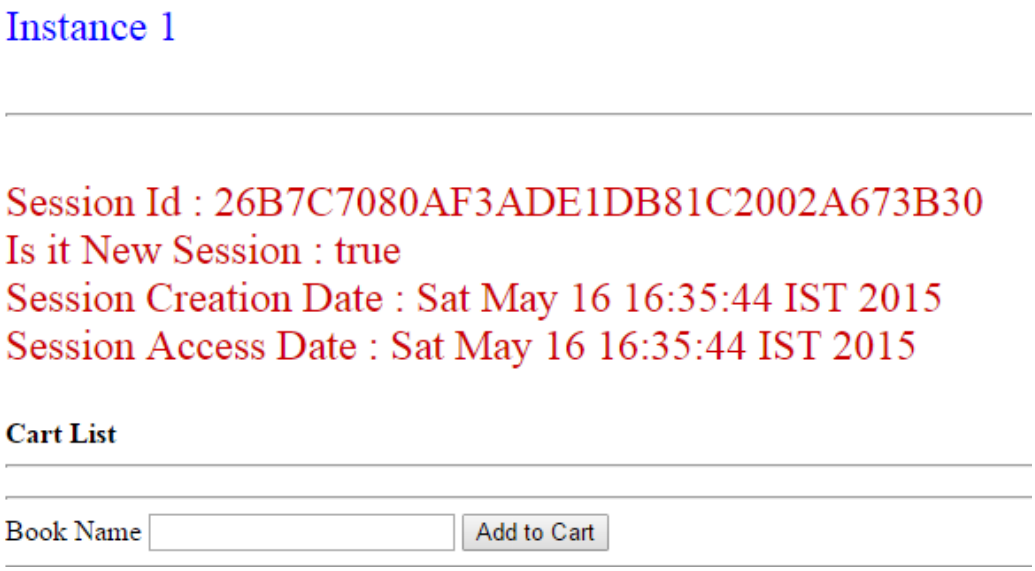



**Figure 22 :** Web page of instance 1

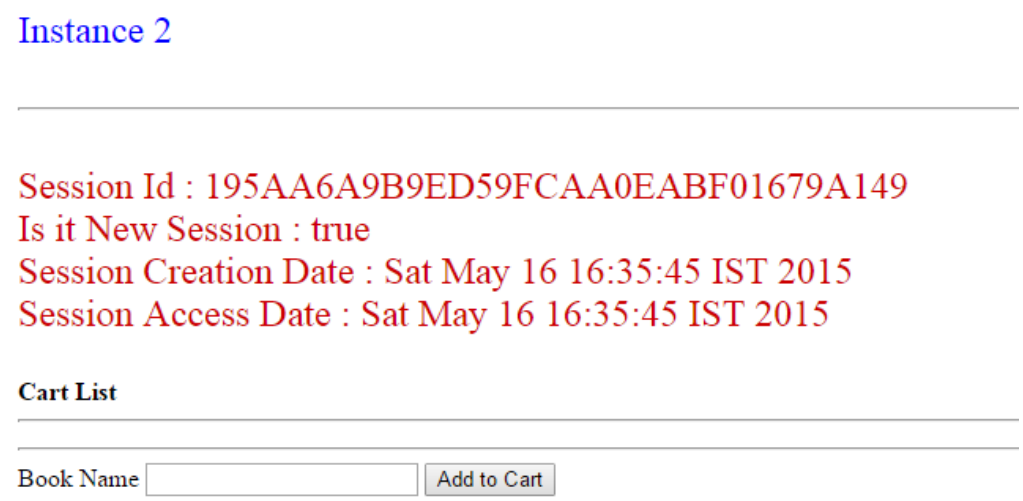



**Figure 23:** Web page for instance 2

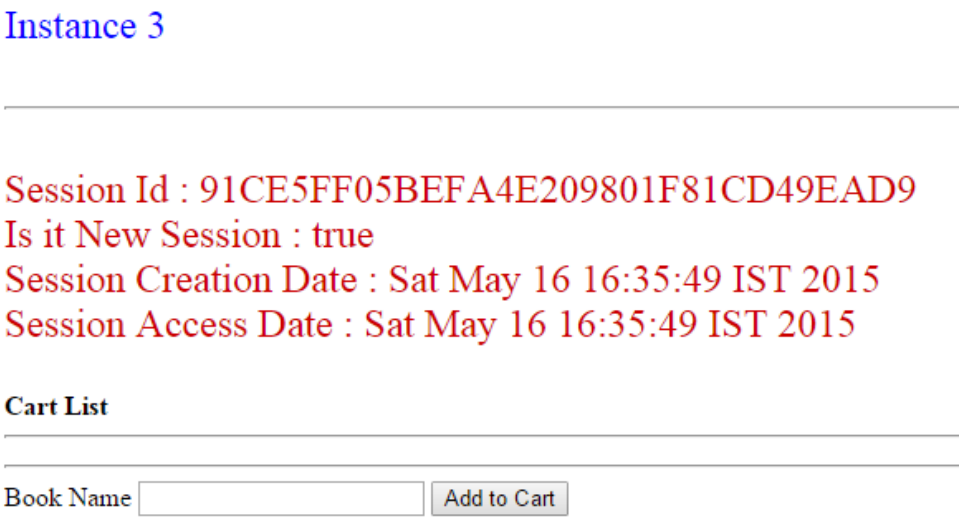



**Figure 24:**Web page for instance 3

Here you can see that each server has its own session id . Session access date will be updated , each time we enter the data

# Chapter 6 :
# Conclusion

With the proliferation of the Internet it is clear that CDN is playing a verge role, but due to huge cost it is still far away from third world countries. In this regards VO based CDN could be a desired solution considering its better QoS in service delivery to end-users and also cost cutting efficiency.

Physics determines how fast one computer can contact another over physical connections, and so attempting to access a server in China from a computer in the United States will take longer than trying to access a U.S. server from within the U.S. To improve user experience and lower transmission costs, large companies set up servers with copies of data in strategic geographic locations around the world. This is called a CDN, and these servers are called edge servers, as they are closest on the company's network to the end-user.

This is a good time to be a site owner. A few short years ago, content delivery solutions were a luxury that only deep-pocketed mega-sites could afford. Now there's a growing selection of competitive products backed by innovative companies offering newer technology that plays well with other technologies such as real user monitoring (RUM)..In order to decrease the traffic load, CDN can be a optimal path

# 6. References


**1.** http://www.ramkitech.com/
**2.** http://www.rackspace.com/
**3.** en.wikipedia.org
**4.** http://www.tutorialspoint.com/
**5.** http://www.javacodegeeks.com/
**6.** http://code.tutsplus.com/
**7.** http://www.mulesoft.com/
**8.** serverfault.com