International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7, Issue-6, March 2019

Building Scalable Framework for web Applications using Go Lang and No Sql

S.Arun Kumar, Vanishree P Iyer, Abhinav Parhad

Abstract*: *Now-a-days with the increase in the number of internet users, the requests sent to a particular server/website per second is increasing rapidly. There is need for service providers to take this into consideration and build their service accordingly. Many times the application works fine but it is unable to handle the rate at which requests are sent to it, due to which it crashes causing inconvenience to the user. If all these requests are processed simultaneously and asynchronously instead of serially, the application performance will improve drastically which intern will help to serve huge number of requests at any given point in time.

Keywords: HTTP, Asynchronously, Server, Client

I. INTRODUCTION

Internet, the worldwide arrangement of interconnected PC utilizes the internet protocol suite (TCP/IP)[4] to interface gadgets around the world, is used widely throughout the world. The number of internet users are increasing very rapidly day by day. With the surge of internet users it is extremely important to provide users with fast and reliable services. This is especially important for websites which receive a lot of requests per second i.e. are used by huge number of users.

One of the main protocol used widely is TCP/IP (Transmission Control Protocol/Internet Protocol). TCP/IP has mainly four layers. This protocol is widely used because it is highly reliable, no duplicate data and sends notification if data is lost/corrupted [4].

Handling thousands or millions of requests per second isn't very trivial and it also painful for the developer to think of such a design every time he designs an application. Instead a generic scale framework can be built to overcome this issue. In this paper we intend to build such a generic scalable framework which can be used by developers in their application just like a plug in.

The Hypertext Transfer Protocol (HTTP) is widely used for exchanging information between clients and servers. Hypertext Transfer Protocol (HTTP) was at first intended to be conventional and stateless. By and large, a web program usually closes its HTTP connection once it gets a request from a server. Usually, an internet browser initiates a HTTP request. A response is returned which contains the address and body related to the request. In certain cases, server sends information without being requested for [5]. HTTP uses REST (REpresentational State Transfer). The status codes like 404 for resource not found, 200 for no error , etc [7].

The project aims at building a framework that can be used to scale services as when required mainly based on requests received per second. The proposed project can handle multiple request (HTTP Post request) and write the information to database asynchronously. Asynchronous programming is a method for parallel programming in which a unit of work runs independently from the principle application thread and informs the calling thread on its finish, be it success or failure. It will improve application performance and responsiveness.

II. RELATED WORK

Different methods and principles have been applied to make applications scalable:

A scalable system with dominated reads and set of query and update templates was proposed in A scalability service for dynamic web applications. In Proc. Conf. on Innovative Data Systems Research (Olston, C., Manjhi, A., Garrod, C., Ailamaki, A., Maggs, C., and Mowry, T., (2005)) [10]. This also includes distributed consistency management and cache invalidation. It has a distributed architecture with proxy server. This paper uses JDBC framework for achieving dominated reads and writes using a well defined template/framework.

A database model using load balancer for scalable system was proposed in Extending enterprise applications to the edge of the internet (Davis, A., Parikh, J., and Weihl, W., (2004)) [11]. It uses data exchange for data collection and data aggregation. A monitoring system is placed on top of it. Load balancer is used to distribute the requests arriving at each instance.

Caching logic used to build scalable systems was proposed in A dynamic data cache for Web applications (K. Amiri, S. Park, R. Tewari, and S. Padmanabhan (2003)) [14]. The main logic used is caching using database proxy layer. Various caching algorithms are used to improve caching. Cache replacement methodology is also discussed. There has always been issues with the ability of CPU processing just one job at any given point in time. This problem was addressed using multi-threading technique.

Revised Manuscript Received on 30 March 2019.
***** Correspondence Author
S.Arun Kumar*, (Assistant Professor Senior Grade), Department of Computer Science, SRM IST, Chennai, India.
Vanishree P Iyer, Department of Computer Science, SRM IST, Chennai, India.
Abhinav Parhad, Department of Computer Science, SRM IST, Chennai, India.

The main disadvantage of this is context switching. Context switching produces a lot of overhead. The main problems with current system are: Advanced Web applications require that, their data is not inconsistent across systems.

It is quite evident that maintaining strong consistency among replicas especially in a distributed system poses significant scalability challenges [6][13].

Administrative managers are usually hesitant to transfer the ownership of data/information and also enforce strict access control over modifying the data outside organization. This hesitance is mainly because of the security issues, information debasement dangers and cross authoritative administration troubles involved. This mainly prevents the use of caching techniques which are mainly based on time-to-live (TTL) protocols which are predominantly used for the development of static content usually in web applications. Now-a-days, web applications deliver dynamic content to its users which may have delicate information requires a high level of information loyalty and must also have a model of consistency for the data to be regular and dependable [6].

There are various factors that affect scalability of an application. Few of them are;

1) High availability
2) Low processing latency
3) Less context switching
4) Efficient garbage collection
5) Fault tolerant system
6) Optimal resource utilization

Internet applications in real-time regularly require extremely high scale. Also, they need to settle on choices inside a strict SLA [19]. This commonly mandates the applications to read/write from/to a database.

The database usually contains huge amount of information (millions or even billions of sets). The read/write operation rate goes upto million operation per second [20][21]. The latency must be as less as possible [2].

Latency is very critical in most of the web applications. The best example would be that of an online payment portal. The portal gets huge number of requests per second. Even if there is little more latency than unusual then unexpected failure may occurs.

III. SYSTEM OVERVIEW

The proposed system provides solution to the problem of frequently scaling an application. It provides a framework which can automatically scale based on the demand (requests). This way the developer of the application need not worry about scaling the application and thus give the users a better feel of using the application with minimal request latency.

The framework is mainly a software based which enables application to scale dynamically. The system processes requests asynchronously .i.e parallel processing. Parallel programming is a way of processing multiple tasks or information concurrently which means a lot of processing can be done simultaneously.

A NoSql database is utilized as the brief database to compose data. NoSql is utilized and explicitly Aerospike which is a key-esteem store, to expand the application execution. Each of the requests are pushed to a queue. This aides in non concurrently preparing the requests.

There are various ways to integrate a scalable system with cloud. Few of them are:-using AWS ASG (Amazon Web Services Auto Scale Group) ,using AWS EC2 (Amazon Elastic Compute Cloud) to host the service [1][18]. Integrating the frame work with cloud will be more productive.

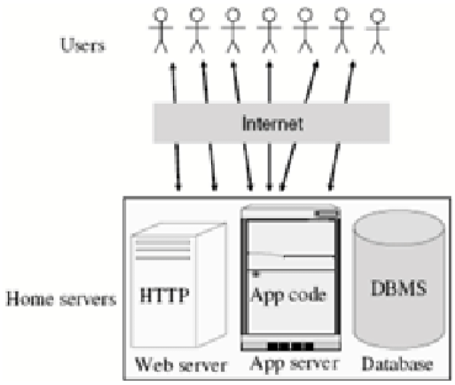

Fig 1 Web Architecture

IV. SYSTEM DESIGN

A. SCALABILITY

Scalability is the ability of an application, service, network or software as a whole to fulfill the increase in demand. Such systems have more advantage over the traditional once since they can dynamically adapt to the increasing requests [10].

Scalable system throughput X(N) can be defined with the help of Universal Scalability Law (USL) [3].

The equational model to compute the relative capacity can be defined as:

$$X(N) = \gamma N/(1+\alpha(N-1) +\beta N(N-1))$$

α, β, γ in equation are three coefficients which can be identified as follows:

1) Concurrency

Concurrency is described as the slope of scalability which rises linearly or the efficient output achievable.

2) Contention

The time taken for obtaining shared resources which may be either by waiting for them randomly or in an organized fashion similar to a queue is called as contention. This value is proportional to α.

3) Coherency

For the data to be used or shared among resources, it must be consistent among the distributed resources. This consistency is called as coherency, which is proportional to β.

Universal Scalability Law (USL) is mainly used for inter process communication [3]. Inter process communication happens at various levels within the system: application software level, middleware level, operating system level, and hardware level.

Scalability of the system can be measured using numerous parameters. Few of them are:

1) The length of the program
2) The quantity of processor tasks per unit time
3) The quantity of current clients

The way the system performs is usually evaluated for various values of these parameters.

$SR = P_i/P_o$

where

SR= Scalability Ratio. The closer this proportion is to 1, the better the Scalability

P_i= The i_{th} level of parameter

P_o = The initial level

Most important factors in determining a system's overall scalability, include:

1) Processor's throughput and it's design
2) Processor's memory limit
3) The input output data transmission rates of the network
4) Features of operating system
5) The design of application software

B. DATA DISTRIBUTION

Data Distribution plays a key role in scalability.

Fig 2 Data Distribution Overview

This sort of engineering is intended to dependably store gigabytes or even terabytes of information with automatic recovery mechanism, replication, etc. This system is a linearly scalable system.

This layer is intended to wipe out the manual tasks with the methodical robotization of all the clusters [2][8]. It incorporates three modules:

1) Management of clusters

A cluster is a combination of two or more nodes. These nodes are put together because of the similarity between them. The algorithm used for managing is based on consensus model called as Paxos algorithm. Paxos algorithm is used for node to node communication mainly in distributed systems. Nodes interact with each other by sending heartbeat messages to indicate that they are alive. A leader is elected to minimize the contention and is responsible for sending messages to other nodes. If the leader becomes inactive then another leader is elected from the existing nodes.

2) Migration of data across nodes

The addition or deletion of nodes determines to which cluster of database does the node belong. The algorithm used to retrieve information from the nodes is based on distributed hashing. Distributed hashing search is based on the keys. All the keys are unique. Values stored in the nodes are identified by their keys. The load of data is equally distributed across all the nodes in a cluster due to distributed hashing technique. All this makes data transfer much simpler.

3) Processing transaction

This module is responsible for reading as well as writing data whenever requested. It also ensures that the data written/read is both consistent across nodes and also isolated between them. This module is in charge of Synchronous or asynchronous replication, proxy and Duplicate Resolution [13].

Several servers or nodes connected to a database is called as database clustering [2]. Clustering has it's own advantages. The main advantage is searching becomes faster since internally cluster tables are maintained [12].

Caching is the mechanism of storing data which is most often used. This minimises the need to fetch the data repeatedly [15]. Also, the data can be retrieved faster. Caching mechanism can be used to increase the speed of data distribution. Advantages of caching are [9]:

1) Reduced latency
2) Increased throughput

Caching plays a vital role in web applications as well. Usually web applications use two type of caches:- server side cache and client side cache. Using two caches increase the performance by manifolds.[9][14]

Data distribution follows a partition assignment algorithm which can be deterministic so that all the nodes in distributes system can independently do computations. All the data in master nodes as well as replica nodes are distributed uniformly so that the load is also equal among them. It also minimizes the movement of partitions.

Partition Algorithm :

```
func ReplicationToPartition(partitionID){
nodeHash=new map()
for each nodeID in nodeList{
  nodeHash[nodeID]=
  ComputeNodeHash(nodeID,partitionID)
 }
 replicationList=sort(nodeHash)
 return replicationList
}
func ComputeNodeHash(nodeID,partitionID){
 nodeIDHash=nodeHash(nodeID)
 partitionIDHash=partitionHash(partitionID)
 hash=merge(nodeIDHash,partitionIDHash)
 return hash
 }
```

Data distribution is well depicted by the following picture:

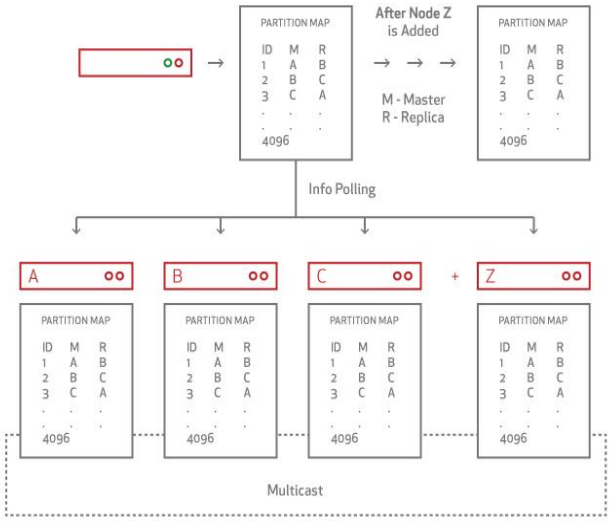

Fig 3 Data distribution amongst nodes

Consider the case when a node goes down. Such node will be deleted from the replication list. If the deleted node is not having a copy of the partition, then migration of data is not necessary in that partition. The old node will be replaced by a new node. When the original node is up then it joins other existing nodes in that cluster. Also, after it joins the cluster it will recapture its position in the partition list. Whenever a new node is brought up, it is becomes a part of the cluster as well as the partition list.

Distributed data structure (DDS) is a storage layer that runs on the cluster [17]. DDS mainly uses conventional DS (Data Structure) techniques like maps, graphs, trees, etc. DDS doesn't expose the methods that it uses internally. Since all the complexity is hidden, DDS becomes simple and easy to use. All DDS transactions preserve ACID (Atomicity Consistency Isolation Durability) properties. Data stored in DDS clusters are consistent. The DDS storage layer is much durable and reliable than [17].

C. CONCURRENCY

Concurrency can be defined as the process in which two processes or tasks are being performed at the same time. This implies different computations are occurring simultaneously. This forms the basis of multi-programming model. There are two models used for concurrent programming:

1) Memory Sharing

In this model, reads and writes occur by using shared objects present in the memory.

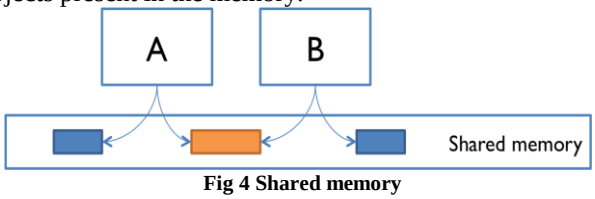

Fig 4 Shared memory

2) Message Passing

In this model, a communication channel is used for sending messages across modules. There is a bidirectional communication among modules i.e they send and receive messages.

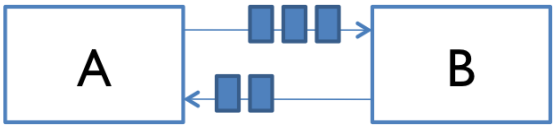

Fig 5 Message Passing

Message passing concurrency is simultaneousness among at least two procedures (a procedure is a stream of control; as opposed to a specific kind of part object) where there is no mutual locale between the two procedures. Rather they convey by passing messages.

Actors Model is a case of a message passing simultaneousness display, with solid messages and supporting both non concurrent and synchronous calls.

While building scalable framework, the input and output must be taken into account. Golang's input/output handling is extremely suitable for building scalable systems. Go's I/O model can be represented as:

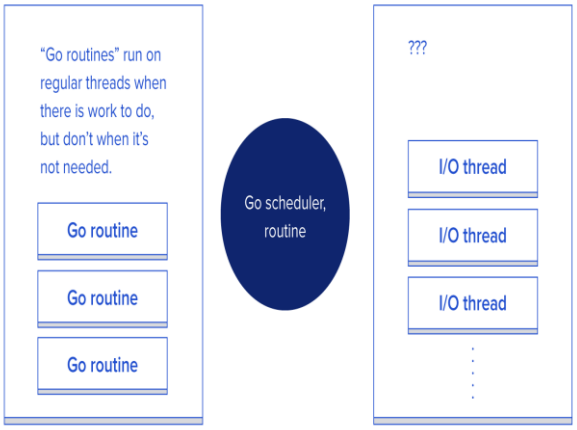

Fig 6 I/O Model of Golang

Golang's concurrency model allows users to execute multiple go routines at a given point in time. All the go routines are non blocking in nature. Consider an application with N go routines (g_0,g_1,g_2.....g_n). Each of these go routines will execute separately and independent of each other. This speeds up the process and yields better results than any other programming language.

Go routines are light weight threads meaning they consume very less memory. All the go routines must be independent. If there is dependency between any two routines say g_0 and g_1, then there is no point in making them go routines since they are blocking in nature i.e g_0 waits for g_1 and vice-versa. In this case the program waits indefinitely and enter deadlock which is not preferred for any programmer. Hence while choosing go routines one must take care of the dependency of routines. To achieve concurrency one must make use of go routines efficiently.

V. METHODOLOGY

Aerospike is utilized for reserving demands as it is a distributed, scalable NoSQL database. The requests are reserved to Aerospike. Aerospike is utilized as a storing component where a TTL (time to live) can be indicated. The records are erased after the predetermined TTL. When records are erased from reserving store,

they can be recovered from persistent datastore which reduces the whole traffic load for active users. Aerospike and mysql will have the indistinguishable information after a delay as it's a queue based asynchronous service.

The partition algorithm is used for caching mechanism. Each node is assigned a P_{id} i.e Partition Id. Data is equally stored among all the nodes. Internally a Hash Id i.e H_{id} is associated with each P_{id}. The H_{id}s help in faster searching. Due to it's various advantages this algorithm is widely used especially for faster searching.

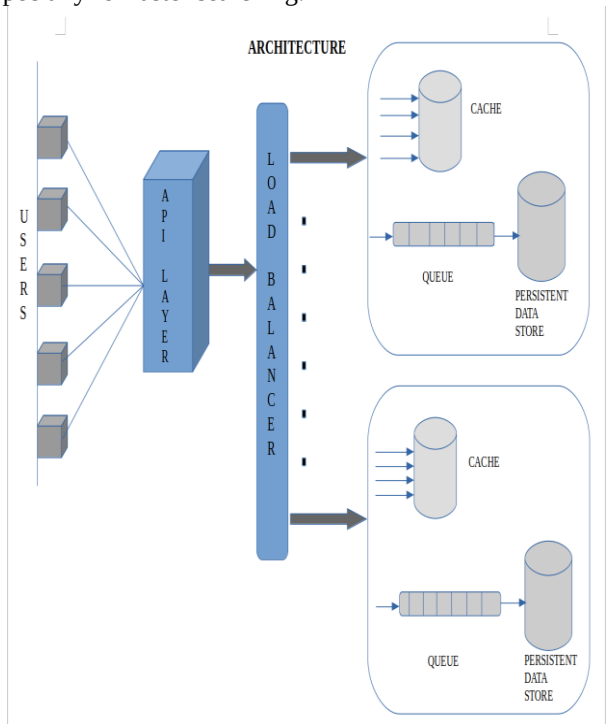

Fig 7 System Architecture

Consider a model of N clients attempting to enlist through a gateway, the entire traffic will be distributed by the load balancer to the instances as indicated by the policy. The policy may be time based or traffic based. Let the clients be $c_0,c_1,c_2...c_N$. Assume X (X=N/10) requests arrives on one of the instance, our service will create workers which will process the request utilizing a few Go routines. The Go routines are primarily like threads. Worker creates X/10 routines (the number of routines is computed by dividing the number of requests by the number of cores of the CPU) and channels read through the routines to dequeue and process it.

Let the number of clients i.e. N be 10000 ($c_0,c_1,c_2...c_{9999}$). Then request on one machine X will become 1000 (10000/10). Since there are 1000 requests, the worker will create 100 (X/10) go routines to process these requests. Thus the workers will spawn go routines dynamically based on the requests. Since this is done dynamically, the frame scales on it's own by spawning go routines as required. All these routines are independent of each other and this increases the processing speed a lot. Although using go routines increases the speed, they must be used carefully since they may result in deadlocks if not properly used

VI. PERFORMANCE ANALYSIS

The main aim of this framework is to architect and develop numerous web applications for a faster user response within the minimal infrastructure possible to support millions of request per minute. The system must be as simple as possible since all the existing systems are very complex and not efficient to handle huge amount of requests at one go.

The plot against time for caching v/s number of concurrent requests and processing time v/s number of concurrent requests is as follows:

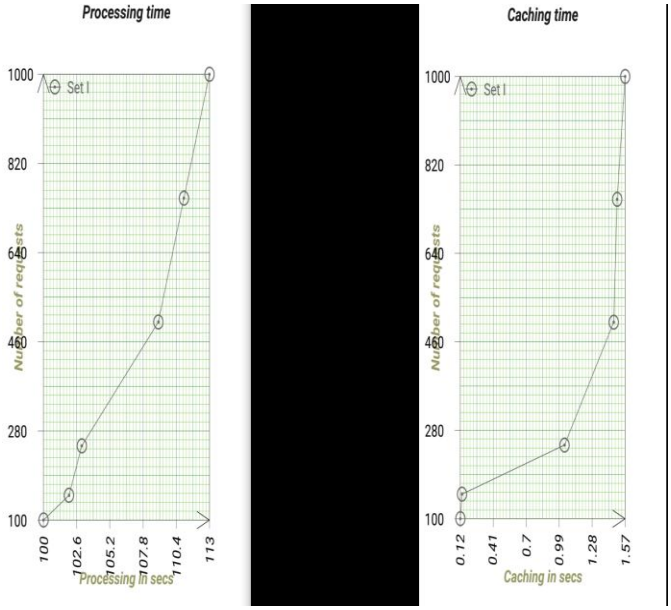

Fig 8 Result graphs

It is clearly seen from the graph that the time taken to process the requests doesn't increase much as the number of requests are increased. This is due to the use of go routines. Using go routines, the requests are processed concurrently and hence they are processed faster. The proposed system is much faster than the existing systems. The performance of Golang is much better than any programming language. GoLang can handle 54502 requests per second, which is much higher than any other language or framework. This speed is mainly due GoLang's concurrency model. The concept of go routines and channels enhance the ability to do concurrent tasks.

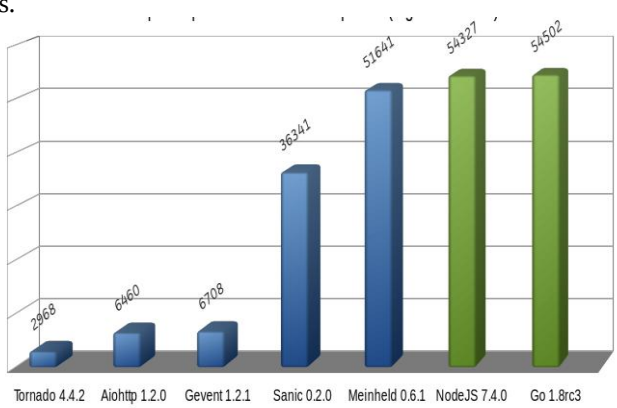

Fig 9 Performance graph

VII. CONCLUSION

In this paper, we have proposed a new system for developing scalable system framework. This will enhance the ability of the applications to scale on demand based on the number of requests.

The proposed system endeavors to introduce a scalable framework model which can be embraced by web application developers so that they can make their system scale according to requests. This ensures better user experience. Also, the information is accessible amid critical occasions independent of unexpected burden increment. This results in better UX (User Experience).

The proposed system framework can be integrated with any web application which wants to scale dynamically. Such a framework reduces the effort of developers considerably.

ACKNOWLEDGMENT

We wish to convey our sincere thanks and gratitude to Mr. S.Arun Kumar (SRM University Ramapuram) for his continuous counselling. His invaluable inputs during the development of project and paper were of great help.

REFERENCES

1. Hussachai Puripunpinyo,M.H. Samadzadeh (2017). Design, Prototype Implementation, and Comparison of Scalable Web-Push Architectures on Amazon Web Services Using the Actor Model.
2. V. Srinivasan, Brian Bulkowski,Wei-Ling Chu (2016). Aerospike: Architecture of a Real-Time Operational DBMS
3. Baron Schwartz (2015). Practical Scalability Analysis With The Universal Scalability Law.
4. Pranab Bandhu Nath, Md.Mofiz Uddin (2015).TCP-IP Model in Data Communication and Networking.
5. V. Wang, F. Salim, and P. Moskovits, The definitive guide to HTML5 WebSocket. New York, NY: Apress, 2013.
6. Oluwagbemi Oluwatolani , Afolabi Babajide , Achimugu Philip (2012). Development of a Scalable Architecture for Dynamic Web-Based Applications.
7. David Alfred Ostrowski (2012).A Scalable, Lightweight WebOS Application Framework.
8. Srinivasan, V. Bulkowski, B., Citrusleaf: A Real-Time NoSQL DB which Preserves ACID., PVLDB 4, (2012).
9. Groothuyse, T., Sivasubramanian, S., and Pierre, G., (2007). GlobeTP: Template-based database replication for scalable web applications.
10. Olston, C., Manjhi, A., Garrod, C., Ailamaki, A., Maggs, C., and Mowry, T., (2005). A scalability service for dynamic web applications. In Proc. Conf. on Innovative Data Systems Research.
11. Davis, A., Parikh, J., and Weihl, W., (2004). Edge computing: Extending enterprise applications to the edge of the internet
12. Cecchet, E., (2004). C-JDBC: a middleware framework for database clustering.
13. Plattner, C., and Alonso, G., (2004). Scalable replication for transactional web applications.
14. K. Amiri, S. Park, R. Tewari, and S. Padmanabhan.,(2003). A dynamic data cache for Web applications.
15. M. Altinel, C. Bornhvd, S. Krishnamurthy, C. Mohan,H. Pirahesh, and B. Reinwald.(2003). Cache tables:Paving the way for an adaptive database cache.
16. Gao, L., Dahlin, M ., Nayate, A., Zheng, J., and Iyengar, A., (2003). Application specific data replication for edge services.
17. Gribble, S., Brewer, E., Hellerstein, J., and Culler, D., (2000). Scalable, distributed data structures for internet service construction.
18. The Cloud Does Equal High Performance, http://highscalability.com/blog/2014/8/20/part-2-the-clouddoes-equal-high-performance.html
19. Real-time bidding, https://en.wikipedia.org/wiki/Realtime_bidding
20. Aerospike Hits One Million Writes Per Second with just 50 Nodes on Google Compute Engine, http://googlecloudplatform.blogspot.in/2014/12/aerospikehits-one-million-writes-Per-Second-with-just-50-Nodes-onGoogle-Compute-Engine.html
21. Aerospike 1 Million TPS, http://highscalability.com/blog/2014/8/18/1-aerospikeserver-x-1-amazon-ec2-instance-1-million-tps-for.html