ISSN:0975-9646

A Generalized Framework for Building Scalable Load Balancing Architectures in the Cloud

Illa Pavan Kumar [1], Subrahmanyam Kodukula [2]

[1] M.Tech, Department of CSE, K L University.

[2] Professor, Department of CSE, K L University.

***Abstract*— Load Balancing in cloud computing is dissimilar from conventional load balancing techniques, because of the limitations in conventional load balancing techniques if they are deployed in the cloud; hence they are unsuitable for the cloud environment. Since the cloud is designed to provide scalable infrastructure, it is difficult to build a scalable architecture in order to take the advantage of scalable infrastructure, because of several issues that differentiate cloud from traditional organizational infrastructure and we can't leverage all that scalability in infrastructure if our architecture is not scalable, both have to work together. In this paper we discuss those issues and limitations of conventional load balancing techniques if deployed in cloud, importance of building scalable architecture for cloud by exploring scalability solutions provided by cloud vendors. Finally, we propose a generalized framework for building scalable load balancing architecture for cloud environment.**

***Keywords*— Cloud- computing, Load balancing, scalable cloud-architecture.**

1. INTRODUCTION

It has become difficult for most of the IT industries and business organizations to buy computing infrastructure (e.g., computing servers, storage or network) due to the tremendous increase in the cost and maintaining such infrastructure, deploying and running business applications on their own becomes an overhead. In order to break all these overheads, organizations are migrating to cloud computing.

1.1. Cloud Computing

Cloud computing is a model that allows us to access software, servers and storage resources over the internet, in a self-service manner. Instead of having to buy, install, maintain and manage these resources on our own computer, we can access and use them through a web browser. Instead laying out capital to buy hardware, software, we rent what we need, usually on a subscription basis. The **NIST** (National Institute of Standards and Technology) Definition [1] of cloud computing is **"**Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction**"**. It provides pay-per-use capabilities. As a user we see only the self-service interface to the computing resources we need. All these resources are maintained by cloud providers at some geographic location. Behind the scenes cloud computing providers have to do a lot of work to manage the entire infrastructure, technology, people that make this possible. To provide services easily, flexibly and profitably to thousands or even millions of user's, they invest heavily in hardware, virtualization techniques, networking infrastructure and automation capabilities. There are thousands of cloud providers and have solutions out. Among them Amazon web services (AWS) and Rackspace are the top most cloud computing service providers.

1.2. Load balancing in the cloud

Main goal of load balancing technique is to achieve maximum throughput with minimum response time [2]. Incoming traffic will be distributed among different servers. Client-server communication must take place without major delay. This delay will lead to the customer dissatisfaction.

Load balancing in the cloud differs from classical thinking on load-balancing architecture and Implementation by using commodity servers to perform the load balancing. It has its own set of challenges like scalability and changes the economics of computing. An infrastructure cloud, such as Amazon's EC2/S3 services [3], promises to fundamentally change the economics of computing. First, it provides a practically unlimited infrastructure capacity (e.g., computing servers, storage or network) on demand. Instead of grossly over-provisioning upfront due to uncertain demands, users can elastically provision their infrastructure resources from the provider's pool only when needed.

Second, the pay-per-use model allows users to pay for the actual consumption instead of for the peak capacity. Third, a cloud infrastructure is much larger than most enterprise data centers. The economy of scale, both in terms of hardware procurement and infrastructure management and maintenance, helps to drive down the infrastructure cost further.

These characteristics make cloud an attractive infrastructure solution especially for web applications due to their variable loads. Because web applications could have a dramatic difference between their peak load and their normal load, a traditional infrastructure is not suitable for them. The common practice of organizations it to provision a fixed capacity. This practice would either result in dissatisfied customers or waste of capital investment. Using the elastic provisioning capability of a cloud, a web application can ideally provision its infrastructure tracking the load in real time and pay only for the capacity needed to serve the real application demand.

Since the cloud is designed to provide large- scalable infrastructure, generally we might have thought that an application can scale up unlimitedly and automatically when demand increases. But when it comes to reality the performance study of existing cloud components [4] shows that scaling an application in cloud is more difficult, because of the several issues that differentiate cloud from traditional organization infrastructure.

First, in traditional organization, application owners have the freedom to customize their infrastructure for optimal results. When it comes to cloud, a cloud infrastructure is owned and maintained by the cloud providers. Hence they offer a limited set of components. For example Amazon EC2 [2] only offers five types of EC2 instances (virtual servers) and application owners cannot customize the specification of them.

Second, again due to the reason that the cloud is maintained by cloud providers, the network bandwidth and computational power of Virtual Machines (VM) of cloud is less than commercial web servers. For example performance study on Amazon VMs [4] shows that they are capable of at most transmitting at roughly 800Mbps, whereas commercial web servers routinely have several network interface cards, each capable of at least 1Gbps.

Third, unlike in traditional organization, application owners have little or no control of the underlying cloud infrastructure. For example, for security reasons, Amazon EC2 disabled many networking layer features, such as ARP, promiscuous mode, IP spoofing, and IP multicast. Application owners have no ability to change these infrastructure features. Many performance optimization techniques rely on the infrastructure choice and control. For example, to scale a web application, application owners either ask for a hardware load balancer or ask for the ability to assign the same IP address to all web servers to achieve load balancing. Unfortunately, neither option is available in Amazon EC2.

Last, commodity machines are likely to fail more frequently. Any architecture design based on cloud must handle machine failures quickly, ideally in a few milli-seconds or faster, in order not to frequently disrupt service.

Because of these characteristics, cloud-hosted web applications tend to run on a cluster with many standard commodity web servers, thus requiring a scalable load balancing architecture.

In next section, we show the usual load balancing techniques.

2. CONVENTIONAL LOAD BALANCING

There are several existing load balancing techniques to achieve scalability in an owned infrastructure. We will study them here and point how they restrict the scalability if deployed in cloud.

2.1 Load Balancer

There are two types of load balancers in existence.

1. Hardware load balancer
2. Software load balancer

A regular way to scale web applications is by using hardware load balancer [5]. The fundamental working rule is that network traffic is sent to a common IP in many cases called a virtual IP (VIP). This VIP is an address that it attached to the load balancer. Once the load balancer receives a request on this VIP it will need to make a decision on where to send it. This decision is usually made by a load balancing algorithm. The client request is then sent to the right server and the server will generate a response. Depending on the type of device, the response will be sent either back to the load balancer, in the case of a Layer 7 device, or more naturally with a layer 4 device straight back to the customer. The hardware load balancer is intended to handle high level of load, so it can simply scale.

However, a hardware-based load balancer uses application specific hardware-based components, thus it is naturally expensive. Because of cloud's commodity business model, a hardware load balancer is rarely offered by cloud providers as a service. As an alternative, one has to use a software load balancer running on a generic server.

A software load balancer [6, 7] is not scalable solution. Since it is run on a generic server, the scalability is generally restricted by the CPU and network bandwidth capacity of the generic server. The generic server's capability is much smaller than that of hardware load balancer. The performance study [4] existing cloud components shows that Amazon EC2 instance can handle at most 400Mbps combined incoming and outgoing traffic. Hence it is not suitable for cloud environment.

2.2 DNS Load Balancing

Another existing technique is DNS load balancing [8]. When a user attempts to access a particular domain via a text-based URL address, that domain request needs to be resolute into an IP address that a local DNS server can interpret. DNS load balancing performs this essential function, translating readable

text domain names into IP addresses and afterward redirecting requests to appropriate servers. If local DNS server does not contain the IP address information, it contacts other DNS servers that have the information, which will be the global DNS server. The global DNS server can handout different IP addresses to different local DNS servers so that the load could be scattered among the servers sitting at each IP address.DNS also specifies a validity period, known as Time-To-Live (TTL) for caching the IP address. After the expiration of TTL, that IP address will be assigned to other local DNS server.

DNS load balancing has its drawbacks- load balancing granularity and adaptiveness [9] those are not specific to the cloud. Since a local DNS server caches the IP address information, all browsers contacting the same DNS server would get the same IP address. Moreover the DNS server could be responsible for a large number of hosts; the load could not be effectively smoothed out.

Second, the local DNS server caches IP address for a set period of time (TTL), e.g., for some days. Until the cache expires, the local DNS server guides requests from browsers to the same web server. When traffic fluctuates at a time scale much smaller than days, tweaking DNS server settings has little effect. Traditionally, this drawback has not been as marked because the number of back-end web servers and their IP addresses are static anyway. However, it seriously affects the scalability of a cloud-based architecture. A cloud-based architecture elastically changes the number of web servers tracking the volume of traffic in minute's granularity. Days of DNS caching significantly reduces this elasticity.

More specifically, even though the architecture increases the number of web servers to serve the peak load, IP addresses for new web servers will not be propagated to DNS servers that already have a cached IP address. Therefore, the requests relying on those DNS servers will be kept and being sent to the old web servers which overloads them while the new web servers remain idle. In addition, when a web server fails, the DNS entry could not be immediately updated. While the DNS changes propagate, users are not able to access the service even though there are other live web servers. Hence it is ill-suited for the cloud.

2.3 LAYER 2 Optimization

Layer 2 refers to the Data Link layer of the Open Systems Interconnection (OSI) model. The Data Link layer is concerned with moving data across the physical links in the network. In a network, the switch is a device that redirects data messages at the layer 2 level, using the destination Media Access Control (MAC) address to determine where to direct the message. We can apply layer 2 optimization techniques to build a scalable web architecture that does not impose all the limitations discussed above.

There are different variations in layer 2 optimization. One of the variations is referred as direct server return [10]. Direct Server Return is a load balancing configuration which facilitates responses to server requests being delivered directly to the network rather than being routed back through the load balancer.

Another variant is TCP handoff [11]. Here we have a sender at front-end. Client request first establishes a TCP bond with this sender. Before any data transfer occurs the sender sends the TCP state to one of the servers at back-end.

Unfortunately, layer 2 optimization leads to security issues. One can capture all packets targeting for a host by inducting another host with the same IP address. Because of these issues, Amazon EC2 disables all layer 2 optimization techniques. Hence it is out of race.

2.4 Client Load Balancing

In client load balancing, client browser decides on the mechanism of request routing. One technique is NetScape's approach [8]. When the user accesses the Netscape home page (located at the URL www.netscape.com), Navigator selects a random number between 1 and the Number of servers and directs the user request to that corresponding server. This technique is generally not applicable, because this architecture is not scalable unless client browser is re-installed. Hence it is not suitable for the cloud.

Other approach is to use Smart Client [12]. It aims at migrates some server functionality to the client machine, in contrast with the traditional approach in which the Web client is not involved. The Java Applet executed at the client side will decide the request routing. Unfortunately, it has several weaknesses. First, Java Applets need the Java Virtual Machine, which is not available by default on most browsers. This is especially true in the mobile environment. Second, if the user accidentally agrees, a Java Applet could have full access to the client machine, leaving open a big security weakness.

Third, many organizations only allow administrators to install software, so users cannot view applets by default. Fourth, a Java Applet is an application; the HTML page routing structure is lost if navigating within the applet. Lastly, Smart Client still relies on a central server to download the Java Applet and the server list, which still presents a single point of failure and scalability jam.

In next section we discuss, best practices given by leading cloud providers to build scalable architectures for cloud.

3. BUILIDING SCALABLE ARCHITECTURES

Since the cloud is designed to provide scalable infrastructure, an ideal scalable load balancing architecture is the one which can handle those entire infrastructure. In general it is difficult to build such architecture. We cannot pull all that scalability in

infrastructure if our architecture is not scalable. Both have to work as one. An ideal scalable architecture should have the following characteristics.

- Growing resources results in a proportional growth in performance
- A scalable service must be able to handle heterogeneity
- A scalable service is operationally efficient
- A scalable service is durable
- A scalable service should become more cost effective when it grows.

3.1 Amazon Guidelines for Cloud Architects

AWS providing best practices [13] to build such architecture. We discuss some of those cloud best practices here.

One such practice is to Design for failure, in this we assume that things will fail. We always design, implement and deploy for automatic recovery from failure. If we realize that the things will fail over time and add in that thinking into our architecture, build mechanisms to handle that failure before disaster strikes to deal with a scalable infrastructure, we will end up creating a fault-tolerant architecture that is well suited for the cloud. This practice is all about the effective utilization and configuring of cloud components like Amazon RDS, Amazon EC2, and Amazon EBS, so that we can build a scalable architecture.

Another practice is, decoupling our components. The main goal of this practice is to build components that do not have tight dependencies on each other, so that if one component were to fail, or not respond, or slow to respond for some reason, the other component will continue the work, so that we can handle the failure. Hence it is all about isolating the various layers and components of our application. One can build such a system with Amazon SQS to isolate components. GrepTheWeb Architecture [14] uses heavy number of Amazon SQS queues. In GrepTheWeb, if lots of requests suddenly reach the server or the processing of regular expressions takes a longer time than the slow response rate of a component, the Amazon SQS queues buffer the requests in a robust manner so that those delays do not affect other components.

The cloud brings a new conception of elasticity in our applications. By implementing elasticity one can build a scalable architecture. To implement “Elasticity”, one has to first automate the deployment process and simplify the design and build process. This will ensure that the system can scale without any human involvement. This will result in immediate cost profit as the overall utilization is increased by ensuring your resources are closely associated with demand rather than potentially running servers that are under-utilized. We can automate the deployment process by utilizing Amazon S3, Amazon EC2, and Amazon EBS.

Other practice is parallelization. Whether it is requesting data from the cloud, storing data to the cloud, or processing data in the cloud, we need to internalize the concept of parallelization when designing architectures in the cloud. It is suitable to not only implement parallelization wherever possible but also automate it because the cloud allows you to create a repeatable process very simply. This can be achieved by Multi-threading our Amazon S3 and Amazon Simple DB requests. We can build ideal architecture by combining elasticity and parallelization practices.

In order to deal with internet latencies, it’s a good practice to keep our data as close as possible to compute or processing elements, i.e. keep dynamic data closer to the compute and static data closer to the end-user. With this practice we can reduce internet latencies in our architecture. This is all about ideal utilization of Amazon S3 component.

Now we analyze a scalable web server farm [9] developed by effective utilization of storage component Amazon S3 for web applications. It comes under client side load balancing technique. Since Amazon S3 supports only static web content, the processing of dynamic content is done by server side scripting on Amazon EC2.For each dynamic page an anchor page is created with the information of list of web servers and their load information, load balancing logic. All these anchor pages are hosted in S3 to achieve scalability. It is a system that delivers the java script as well as the static content becomes a single point of contact for all clients. In this architecture, scalability is achieved by efficient utilization of Amazon S3 and Amazon EC2. We can use reference architectures [15] offered by AWS while building a scalable architecture.

3.2 Rightscale cloud management service

Until now in our discussion, the end-users have no control on underling cloud infrastructures, i.e. we cannot customize the specifications of existing cloud components. RIGHTSCALE [16] is a cloud management service provider, allowing end-users to have a control on our cloud infrastructures. They have the software, tools that allow us to control underlying cloud infrastructure. RIGHTSCALE is having the deal with leading cloud providers like Amazon web services, Rackspace etc.

For this RIGHTSCALE is providing a dashboard, which is a web-based portal that we use to control existing cloud components. For example if we are working with AWS components, if we need a modification to those components, we use RIGHTSCALE dashboard to specify those modifications. They templatize and automate many of those modifications, to make the whole process easier than doing it natively on AWS's console. If we require other than that templates, then we can modify one of their templates, or build one from scratch similar to how we would bundle an image directly on AWS. After specifying the modifications, RIGHTSCALE Application

Programming Interface (API) which is abstracted away from AWS, interacts with their API. If we launch a server through dashboard, RIGHTSCALE API makes a call with AWS to launch that server instance.

Moreover RIGHTSCALE is providing tools, tips and techniques [17] for building scalable load balancing architecture in the cloud.

Now we discuss a case study of “Woot.com” [18], which is a well known online e-commerce site, famous for creating the "one day one item sale" business model. Woot.com's main website offers one discounted product each day, often a piece of computer hardware or an electronic gadget.Woot.com turned to the cloud because of the attractive pay-as-you-go, resources on-demand model. This model provided Woot.com a solution for handling massive traffic spikes without the up-front capital investment. Woot.com chose Amazon Web Services (AWS) cloud as its base cloud infrastructure. Periodically Woot.com announces a "Woot-Off", a short term frenzied alteration of the product posting procedure. In Woot-Off mode, a new product is launched immediately after the sellout of the previous deal. They configured aiCache load balancer offered by AWS for use during woot-off’s through RIGHTSCALE dashboard. Using the RIGHTSCALE platform, Woot.com deployed instances on the Amazon Web Services Elastic Compute Cloud (EC2) and directed all web traffic to the aiCache accelerators.

During their Woot-off traffic events, Woot.com harnesses the power of RightScale's monitoring, and now uses their load balancer to move all Woot-off traffic from their datacenter to the aiCache instances in the cloud. Once complete, they use the Rightscale dashboard to scale back the instances on Amazon EC2 to reduce costs. Hence Woot.com achieved scalability. This is all about using Amazon EC2 according to user needs. We can use the reference architectures [19] made by RIGHTSCALE while building a scalable load balancing architecture.

From section 3 we conclude that building a scalable load balancing architecture for a cloud is all about ideal grouping of existing cloud components like Amazon EC2/S3, Amazon SQS etc. if this grouping is associated with appropriate cloud management like RIGHTSCALE we can achieve scalable architecture for the cloud. Now from above study, we propose a generalized framework for building a scalable load balancing architectures with Amazon cloud components under the management of RIGHTSCALE cloud platform.

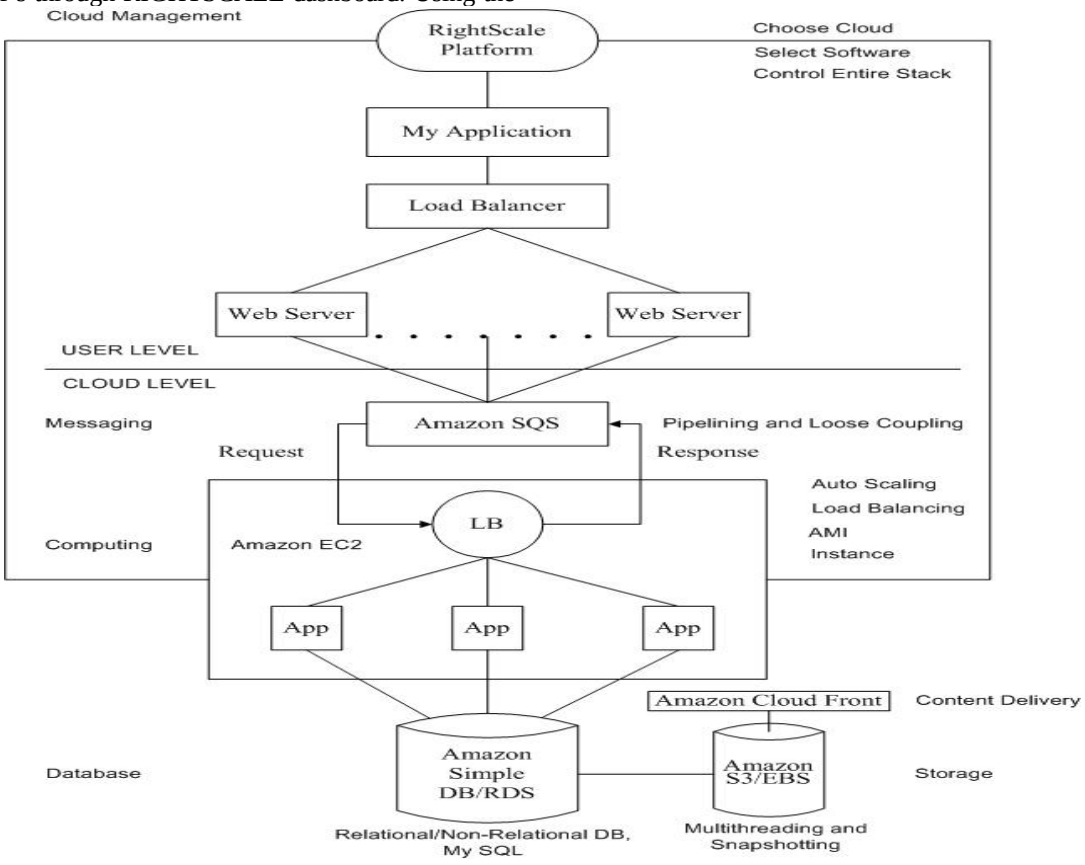

Fig 1: A Generalized framework for Scalable Architectures

4. GENERALIZED FRAMEWORK

Since most of the organizations moving their data centre to Amazon Web Services [2] cloud, we proposed the framework shown in Fig 1. for building scalable load balancing architecture with Amazon computing component like Amazon EC2[21] and storage options like Amazon /S3/EBS/SimpleDB/RDS/SQS[20] according to the cloud best practices provided by AWS[13]. This framework is a real or conceptual structure intended to serve as a support or guide for the building of scalable architectures. Now we explain the components of the framework and how they lead to scalability of our application, My Application in the Fig 1 with the best usage scenario.

Amazon EC2: Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. You can bundle the operating system, application software and associated configuration settings into an Amazon Machine Image (AMI), through which we launch our instances or we can launch multiple instances. With Amazon EC2 we can build applications that start small but can scale up rapidly as demand increases. However, for the efficient usage of Amazon EC2, it is important to architect your application correctly, otherwise we cannot rebuild after the occurrence of a failure. To ensure that our application scales with increasing load, you must ensure that it has no bottlenecks or single points of failure, and that it can continue to run even when one or more components fail. It is associated with feature like auto scaling; Load Balancing (LB among App servers) etc.It is an Amazon cloud computing platform.

Amazon S3: Amazon S3 is a highly scalable, robust and available distributed object store designed for mission-critical and primary data storage with an easy to use web service interface. We can store unlimited number of objects in Amazon S3 bucket, hence it is highly scalable. Common use for Amazon S3 is storage of static web content. This content can be delivered directly out of Amazon S3 via a web server, since each object in Amazon S3 has exclusive HTTP URL address, or delivered through Amazon Cloud Front. Multithreading the requests to Amazon S3 is the best practice to achieve scalability.

Amazon EBS: Amazon Elastic Block Storage (EBS) is meant for data that changes frequently and requires durable persistence. EBS provides persistent virtual block mode storage for Amazon EC2virtual servers, so that we can use it just as a hard drive on a physical server. Amazon EBS is particularly well-suited for use as the primary storage for a file system, database or for any applications that require fine granular updates and access to raw, unformatted block-level storage. In order to maximize both durability and availability of their EBS data, users should snapshot their EBS volumes frequently and store those images in AmazonS3.

Amazon SQS: Amazon Simple Queue Service (Amazon SQS) provides a reliable, highly scalable, hosted message queuing service for temporary storage and delivery of short(up to 64kB) text-based data messages. An Amazon SQS queue is a temporary data repository for messages that are waiting for processing. While Amazon SQS and other message queuing services are usually thought of as an asynchronous communication protocols, Amazon SQS can also be viewed as a store for providing temporary but durable data storage for many classes of applications.

Use of Amazon SQS as temporary storage can minimize the use of other storage mechanisms, such as temporary disk files. It is the main component to build scalable architectures. It enables us to build scalable EC2 applications by providing the features like loose coupling and pipe lining. It isolates the failure of a component from the rest of the system. The failure will be transparent to the end user. We can implement a high-availability solution with Amazon SQS, because it makes it possible to drop in a replacement server without impacting the rest of the system. A single Amazon SQS queue can handle multiple server instances simultaneously. So that load can be balanced. Some of the processing operations may take longer time than the rest; we have to implement such long-running operations in a separate, dedicating server. This can be implemented with minimal interruption to the rest of the system using Amazon SQS, by pipelining.

Amazon SimpleDB: This storage alternative is well-suited for situations where we have structured, fine-grained data that we need to persist and then query with high availability and durability. Many organizations use SimpleDB as part of a larger cloud-based storage architecture. A common pattern is to use SimpleDB to keep track of metadata about information that is stored in other AWS offerings, SimpleDB is a highly available, scalable, and flexible non relational data store that offloads much of the work of database administration and associated systems management.

Amazon RDS: Amazon Relational Database Service (RDS) is a fully functional, MySQL relational database provided as a managed, cloud-based service. If our application requires relational storage, but we want to reduce the time we spend on database management, Amazon RDS automates common administrative tasks to reduce the complexity and total cost of ownership, allowing us to spend more time on application development. Amazon RDS is a great choice for any application that relies on MySQL as its information repository and you want to take advantage of a highly scalable, low-maintenance, cost-effective, cloud-based database without the need to make any code changes.

Amazon CloudFront: Amazon CloudFront is a web service for content delivery. It integrates with other components to give developers an easy way to distribute content end users with low latency and high data transfer speeds.

Rightscale Platform: With this cloud management service we can select the underlying cloud, chose the software, control the entire infrastructure through RIGHTSCALE dashboard. It is optional to use this platform if user does not need any modifications to cloud components.

Most of the organizations are developing their own architectures according to their needs. It is not necessary to use all the cloud components, organizations will selects the components that suits with their usage scenarios. The main thing is the grouping the selected components ideally Hence there is no standard architecture for load balancing in the cloud, it depends on organization needs, they may use the reference architectures offered by the cloud provider or they modify the components through a cloud management service.

5. CONCLUSION

This paper has provided the importance of building scalable architectures in the cloud. By following the best practices of AWS - like designing for failure, decoupling the application components, understanding and implementing elasticity, combining it with parallelization, we can understand the design considerations required for building highly scalable cloud applications and the proposed framework is intended to serve as a support or guide for building scalable load balancing architectures.

ACKNOWLEDGEMENTS

We like to express our gratitude to all those who gave us the possibility to carry out the paper. We would like to thank Mr.K.Satyanarayana, chancellor of K.L.University, Dr.K.Raja Sekhara Rao, Dean, and K.L.University for stimulating suggestions and encouragement. We have further more to thank Prof.S.Venkateswarlu, Dr.K.Subrahmanyam, who encouraged us to go ahead with this paper.

REFERENCES

[1]. Peter Mell, Timothy Grance, “National Institute of Standards and Technology Special Publication 800-145: The NIST Definition of cloud computing”, September 2011.

[2]. Shimon ski. R, “Windows 2000 & Windows Server 2003 Clustering and Load Balancing Emeryville,” McGraw-Hill Professional Publishing, CA, USA (2003), p 2, 2003.

[3]. Amazon Web Services, “Amazon Web Services (AWS)”, http://aws.amazon.com.

[4]. H. Liu and S. Wee. Web Server Farm in the Cloud: "Performance Evaluation and Dynamic Architecture”, In Proc. of the 1st International Conference on Cloud Computing (CloudCom 2009), Dec 2009.

[5]. F5 Networks,"F5 Networks”, http://www.f5.com.

[6]. HaProxy, “HaProxy load balancer”, http://haproxy.1wt.eu/.

[7]. Nginx,"Nginx web server and load balancer", http://nginx.net/.

[8]. Valeria Cardellini, Michele Colajanni, “Dynamic Load Balancing on web-server systems” IEEE Internet Computing, vol. 3, no. 3, pp. 28-39, May-June 1999.

[9]. H. Liu and S. Wee, "Client-Side Load Balancer Using Cloud", SAC’10 March 22-26,ACM 2010, Sierra, Switzerland.

[10]. L.Cherkasova, "FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service", IEEE Symposium on Computers and Communications, 0:8, 2000.

[11]. G. Hunt, E. Nahum, and J. Tracey, "Enabling content-based load distribution for scalable Services”, Technical report, 1997.

[12]. C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler," Using Smart Clients to Build Scalable Services, In Proc. USENIX 1997Annual Technical Conference, Jan 1997.

[13]. J.Varia, “Architecting for the cloud: Best practices”. http://d36cz9buwru1tt.cloudfront.net/AWS_Cloud_Best_Practices.pdf.

[14]. J.Varia, "Cloud Architectures". http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf,2007-07-01

[15]. AWS. http://aws.amazon.com/architecture/.

[16]. RightScale Cloud Management. http://www.rightscale.com/.

[17]. Brian Adler,” Load Balancing in the Cloud: Tools, Tips, and techniques”. http://www.rightscale.com/info_center/white-papers/load-balancing-in-the-cloud.php

[18]. RightScale Cloud Management. http://www.rightscale.com/customers/woot-scales-usingrightscale-and-aicache.php.

[19]. RightScale Cloud Management. http://support.rightscale.com/12Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams. http://support.rightscale.com/12Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams.

[20]. Joseph Baron, Amazon Web Services Robert Schneider, Think88."Storage Options in theAWS Cloud”. http://d36cz9buwru1tt.cloudfront.net/AWS_Storage_Options.pdf.

[21]. AWS.http://aws.amazon.com/ec2/