I.J. Information Technology and Computer Science, **2012, 5, 8-15**
Published Online May 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2012.05.02

Solving Web-based Applications Architectural Problems in the Cloud: The Way Forward

Philip Achimugu[a], Oluwatolani Oluwagbemi[b], Ishaya Gambo [c]
[a,b,]*Department of Computer and Information Science, Lead City University, Ibadan. Nigeria*
[a]*Email:check4philo@yahoo.com*

[c]*Department of Computer Science and Engineering, Obafemi Awolowo University, Ife-Ife.*

***Abstract*—Highly-available and scalable software systems can be a complex and expensive proposition. Traditional scalable software architectures have not only needed to implement complex solutions to ensure high levels of reliability, but have also required an accurate forecast of traffic to provide a high level of customer service. This traditional software architecture is built around a common three-tier web application model that separates the architecture into presentation, business logic and database layers. This architecture has already been designed to scale out by adding additional hosts at these layers and has built-in performance, failover and availability features. Even with all these developments in architectural designs, some software still lacks in scalability, reliability and efficiency. This paper therefore examines the shortfalls of traditional software architectural problems with a view to addressing them using the cloud computing approach.**

***Index Terms*—Scalability, Software, Systems, Architecture, Service**

1. Introduction

Cloud computing is a model for enabling 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 [4]. For several years however, software architects have discovered and implemented several concepts and best practices to build highly scalable applications. In today's "era of tera", these concepts are even more applicable because of ever-growing datasets, unpredictable traffic patterns, and the demand for faster response times.

As companies move computing resources from premises-based data centers to private and public cloud computing facilities, they should make certain their applications and data a safe and smooth transition to the cloud. In particular, businesses should ensure that cloud-based facilities will deliver necessary application and transaction performance now, and in the future. Much depends on this migration and preparation for the transition and final cutoff. Rather than simply moving applications from the traditional data center servers to a cloud computing environment and flick the "on" switch, companies should examine performance issues, potential reprogramming of applications, and capacity planning for the new cloud target to completely optimize application performance. Applications that performed one way in the data center may not perform identically on a cloud platform. Companies need to isolate the areas of an application or its deployment that may cause performance changes and address each separately to guarantee optimal transition [3]. In many cases, however, the underlying infrastructure of the cloud platform may directly affect application performance.

Therefore, businesses should also thoroughly test applications developed and deployed specifically for cloud computing platforms. Ideally, businesses should test the scalability of the application under a variety of network and application conditions to make sure the new application handles not only the current business

demands but also is able to seamlessly scale to handle planned or unplanned spikes in demand.

Finally, this paper is divided into about eight sections. The first section introduces the topic of discussion which is cloud computing, the second section discusses software architecture practice as a discipline, the third section deals with the essentials of cloud computing, the fourth section enumerates cloud computing performance, the fifth section has to do with understanding performance, scale and throughput in the context of cloud computing infrastructure while the sixth section describes the scalable architectural considerations in the cloud and the seventh section presents an envisioned reference architecture for the cloud. Section eight brings the research to a logical conclusion.

2. Software Architecture Practice

Today, software architecture practice is one sub-discipline within software engineering that is concerned with the high-level (abstract) design of the software of one or more systems [1]. Software architecture are created, evolved, and maintained in a complex environment. The architecture business cycle of **Figure 1** illustrates this. On the left hand side, the figure presents different factors that influence software architecture through an architect. It is the responsibility of the architect to manage these factors and take care of the architecture of the system. An important factor is formed by requirements, which come from stakeholders and the developing organization. The architect also has the capacity of influencing opinions of stakeholders, refine user's requirement in a way that it captures all the activities of an organization as well as determine the technicalities of the proposed software in terms of development techniques, architectural considerations, programming language (s) to be used and the extent of scalability of the database [2].

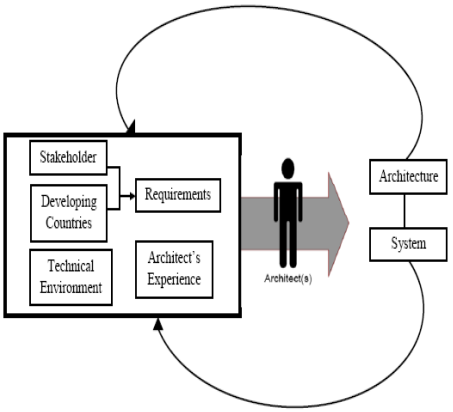

Figure 1: The Architectural Business Circle[1]

3. Essentials of Cloud Computing

Poor application performance causes companies to lose customers, reduce employee productivity or morale, reduces bottom line revenue and trust. Because application performance can vary significantly based on delivery environment, businesses must make certain that application performance is optimized when written for deployment on the cloud or moved from a data center to a cloud computing infrastructure. Applications can be tested in cloud and non-cloud environments for base-level performance comparisons. Aspects of an application, such as disk I/O and RAM access, may cause intermittent spikes in performance. However, as with traditional software architectures, overall traffic patterns and peaks in system use account for the majority of performance issues in cloud computing [3].

Capacity planning and expansion based on multiples of past acceptable performance solves many performance issues when companies grow their cloud environments. However, planning cannot always cover sudden spikes in traffic, and manual provisioning might be required. A more cost-effective pursuit of greater scalability and performance is the use of more efficient application development; this technique breaks code execution into silos serviced by more

easily scaled and provisioned resources. In response to the need for greater performance and scalability in cloud computing environments, this paper offer scalability features and options that aid application performance, including lightweight virtualization, flexible resource provisioning, dynamic load balancing and storage caching, and CPU bursting. This will allow businesses to develop more efficient applications that are easily ported to virtually any open standards environment.

4. Cloud Computing Performance

Poor quality of service in applications, and Web pages frustrates employees and customers alike, and some performance problems and bottlenecks can even cause application crashes and data losses. In these instances, performance or lack of performance is a direct reflection of the company's competency and reliability. Customers are not likely to put their trust in a company whose applications crash and are reluctant to return to business sites that are frustrating to use due to poor performance and low response times. Intranet cloud-based applications should also maintain peak performance. Positive employee productivity relies on solid and reliable application performance to complete work accurately and quickly.

Application crashes due to poor performance cost money and impact morale. Poor performance hampers business expansion as well. If applications cannot adequately perform during an increase in traffic, businesses lose customers and revenue. Adequate current performance does not guarantee future behavior [4]. An application that adequately serves 100 customers per hour may suddenly nose-dive in responsiveness when attempting to serve 125 users. Capacity planning based on predicted traffic and system stress testing can help businesses make informed decisions concerning cost-effective and optimum provisioning of their cloud platforms. In some cases, businesses intentionally over-provision their cloud systems to ensure that no outages or slowdowns occur. Regardless, companies should have plans in place for addressing increased demand on their systems to guarantee they do not lose customers, decrease employee productivity, or diminish their business reputation.

5. Understanding Performance, Scale, and Throughput in the Context of Cloud Computing Infrastructure

Performance is generally tied to an application's capabilities within the cloud infrastructure itself. Limited bandwidth, disk space, memory, CPU cycles, and network connections can all cause poor performance. Often, a combination of lack of resources causes poor application performance. Sometimes poor performance is the result of an application architecture that does not properly distribute its processes across available cloud resources while the effective rate at which data is transferred from point A to point B on the cloud is throughput. In other words, throughput is a measurement of raw speed. While speed of moving or processing data can certainly improve system performance, the system is only as fast as its slowest element. A system that deploys ten gigabit Ethernet yet its server storage can access data at only one gigabit effectively has a one gigabit system and scalability has to do with the search for continually improving system performance through hardware and software throughput gains [3] . This is defeated when a system is swamped by multiple, simultaneous demands. The only way to restore higher effective throughput in such a "swamped resources" scenario is to scale or add more of the resource that is overloaded. For this reason, the ability of a system to easily scale when under stress in a cloud environment is vastly more useful than the overall throughput or aggregate performance of individual components. In cloud environments

however, this scalability is usually handled through either horizontal or vertical scaling.

When increasing resources on the cloud to restore or improve application performance, administrators can scale either horizontally (out) or vertically (up), depending on the nature of the resource constraint. Vertical scaling (up) entails adding more resources to the same computing pool for example, adding more RAM, disk, or virtual CPU to handle an increased application load. Horizontal scaling (out) on the other hand requires the addition of more machines or devices to the computing platform to handle the increased demand.

6. Scalable Architectural Considerations in the Cloud

A scalable architecture can take many forms; but in essence, it is an application and underlying infrastructure that can adapt to dynamically changing conditions to promote the availability and reliability of a service4.

With the proliferation of online communities and the potential for cross-pollination across these environments, the traffic and load patterns encountered by an application have become unpredictable as the potential for viral or flash crowd events can drive unprecedented traffic levels. This dynamic nature drives the need for a massively scalable solution to enable the availability of web-based applications.

Previously, in the traditional hardware model, there were two approaches one could take with regard to the issue of the unpredictability of site traffic and system load, each of which is illustrated in Figure 2.

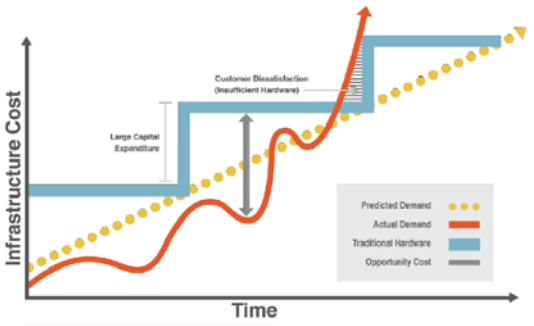

Figure 2: Traditional Hardware Model3

The first approach was to overprovision, that is, to have enough resources in place to handle any spikes in traffic that may occur. While this enables an application to increase its availability in high-traffic situations, it is not an effective use of resources because a portion (and perhaps the majority) of these resources sits idle during non-peak periods which makes it uneconomical while the second approach in the traditional hardware model is to provision for the typical usage pattern of the application and suffer the consequences of lost traffic when peak demands are encountered. Although this is more cost friendly in times of normal usage, it is costly during traffic spikes because lost traffic typically means lost revenue opportunities. This scenario is illustrated in Figure 2 by the shaded region under the demand curve represented by the red line and above the available infrastructure capacity represented by the blue line. In this situation the demand exceeds capacity, and, as a result, traffic is lost and/or the application service is unavailable. Neither of these approaches in the traditional hardware model is ideal, which is why the scalable cloud model is such an excellent fit for this type of dynamic and unpredictable environment. With the scalable cloud model, you can dynamically provision additional resources only when they are needed and then decommission them when they are no longer required. In true utility computing fashion, you incur charges only for the time period in which you use the resources. Figure 3 illustrates the scalable cloud model for application resources. In this figure, the demand curve

is identical to that of Figure 2, but due to the dynamic provisioning of resources, at no time is there excess capacity in which servers sit idle, nor is there insufficient capacity to accommodate the demand for the application. In the following sections, we will describe the preferred methods and techniques for best implementing the scalable cloud model at all levels of an application's multi-tiered architecture. The y-axis represents infrastructure costs (which can be generalized to represent the number of servers in use) while the x-axis represents the time at which user requests are processed, and the red line is an indication of actual user demand for the service provided by the application. The gray arrow illustrates the disparity between the two. This scenario is obviously a costly solution due to the presence of unutilized capacity and therefore generally not a common or recommended approach.

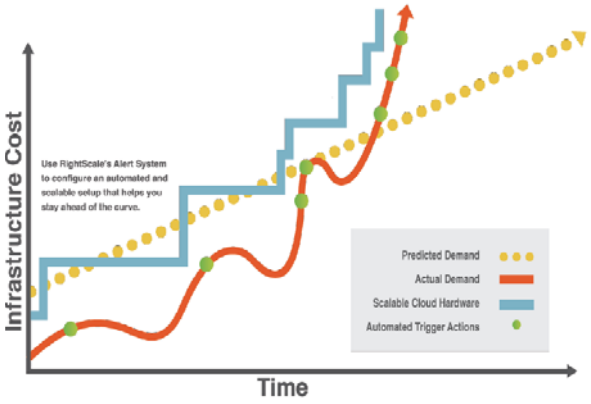

Figure 3: Scalable Cloud Model3

7. Envisioned Scalable References Architecture for the Cloud

Scalable web-based applications enhance organizational performance and also align enterprises. As a result, Figure 4 illustrates a reference architecture that could be adopted by scalable applications that incorporates the best practices for successful implementations of our diverse customer base. Worthy of note is that this architecture looks much like the conventional three-tier web application architecture, which has to do with the presentation, business logic and database layers with the addition of a caching tier between the application servers and the database. However, there are some subtle (and some less subtle) modifications and enhancements that can be made to the architecture to allow it to best handle the unique traffic and load patterns experienced by scalable applications at run time.

The components of the proposed scalable architecture is shown in Figure 4.The first tier shown in the architecture is composed of clients who are responsible for sending HTTP requests. The responses of these requests must be provided at good response time based on user profile. Scaling application-specific computation is relatively easy as requests can be distributed across any number of independent application servers running identical code through the use of two load balancers called request allocator. Regardless of estimated load, we recommend using two front-end load balancers to avoid redundancy in the case of a server failure. Additionally, it is better to ensure that these load balancers be placed in different availability zones to increase the reliability and availability of the application. Similarly, one can reduce the network bottleneck between the application and database servers.

The main challenge, however, is to scale access to application data. The application server are positioned in different availability zones and with alert mechanisms in place to allow automatic scaling (both up and down) of the array based on instance specific metrics. These metrics used for auto-scaling include CPU-idle, free memory, and system load. However, one can use virtually any system metric as an auto-scaling trigger, including application-specific metrics that you can add via custom plug-ins to the system. The core of this architecture lies in the implementation of the proxy servers. A traditional proxy server is sufficient to serve static content on behalf of

application providers. To generate dynamic content however, the system must be designed in such a way that each proxy server has immediate and dynamic access to the contents in the main or home server.

The proxy server also contains a simple database cache designed to reduce CPU utilization and bandwidth consumption of the home server, as well as the impact of database queries on the execution time of a request. Each home server embodies the traditional three-tiered architecture, which enables it to generate Web content dynamically. While it is aimed at offloading the generation of dynamic content to the proxy servers, the prototype is compatible with this well-established home server architecture and hence also allows the application provider to serve directly as much dynamic content as it chooses. The top tier is a standard web server, which manages HTTP interactions with clients.

The web server is augmented with a second tier, the application server, which can execute a program to generate a response to a client's request based on the client profile, the request header, and the information contained in the request body. Finally, the third tier consists of a database server that the application provider uses to manage all of its data. One important issue in any replicated system is consistency. Consistency management has two main aspects: update propagation and concurrency control. In update propagation, the issue is to decide which strategy must be used to propagate updates to replicas. Many strategies have been proposed to address this issue. They can be widely classified into push-based and pull-based strategies. Pull-based strategies are mostly suitable for avoiding unnecessary data update transfers when no data access occurs between two subsequent updates while pushing data updates immediately ensures that replicas are kept consistent and the servers hosting replicas can serve read requests immediately.

The system must also handle concurrent updates to a data unit emerging from multiple servers. Traditional non-replicated DBMSs perform concurrency control using locks or multiversion models. For this, a database requires an explicit definition of transaction, which contains a sequence of read/write operations to a database. When querying a database, each transaction sees a snapshot of consistent data (a database version), regardless of the current state of the underlying data. This protects the transaction from viewing inconsistent data produced by concurrently running update transactions.

However, concurrency control at the database level can serialize updates only to a single database and does not handle concurrent updates to replicated data units at multiple edge servers. Traditional solutions such as two-phase commit are rather expensive as they require global locking of all databases, thereby reducing the performance gains of replication. To handle this scenario, the system must serialize concurrent updates from multiple locations to a single location. The system does guarantee transaction semantics and also enforces consistency for each query individually.

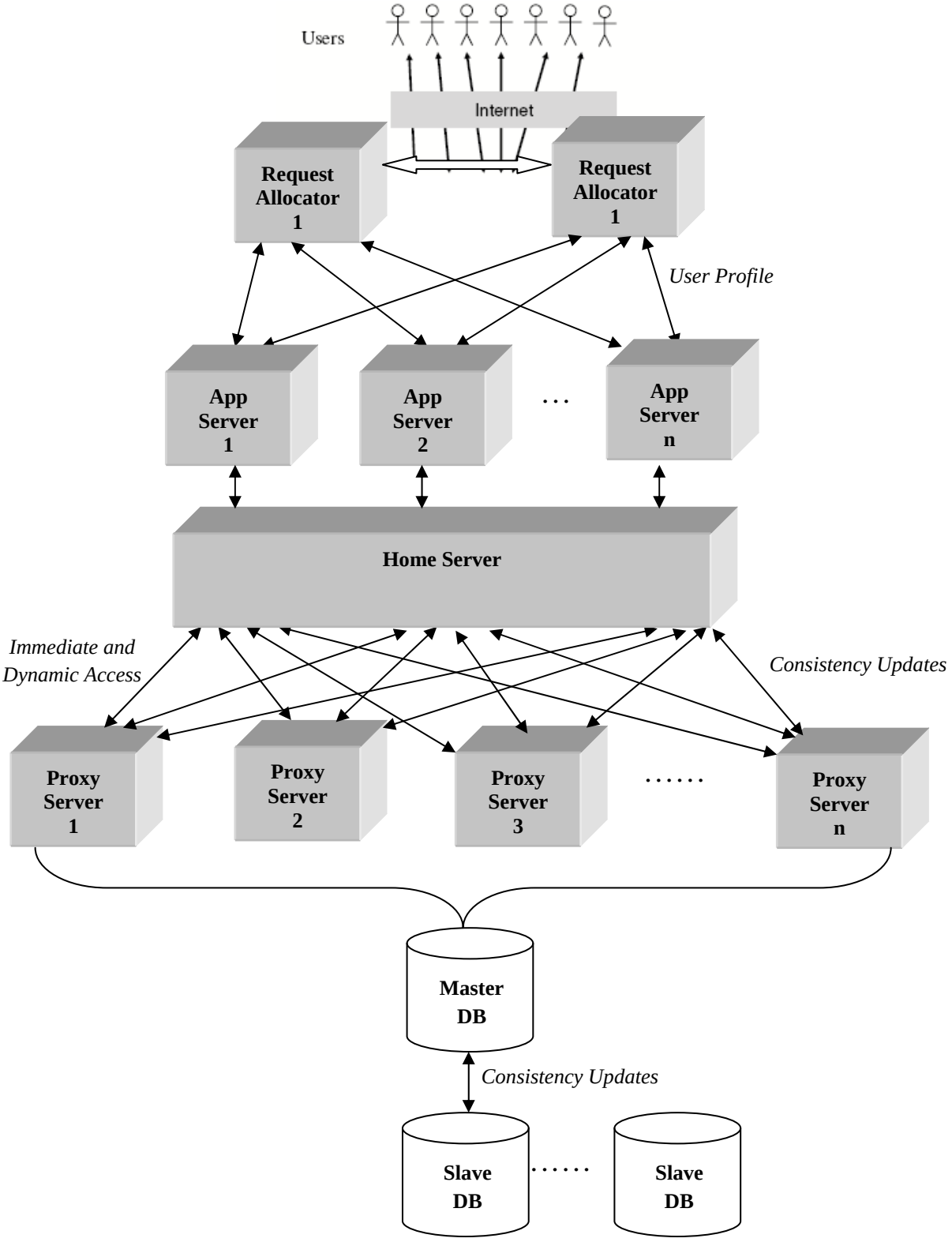

Figure 4: Proposed Scalable Architecture for the Cloud

8. Conclusion

Most approaches toward scalable hosting of Web applications consider the application code and data structure as constants, and propose middleware layers to improve performance transparently to the application. This paper keys into this concept and

demonstrates that major scalability improvements can be gained by allowing one to decentralize an application's data into independent services that are interoperable and allows regular updates using profile of users to respond to queries. While such restructuring introduces extra costs, it considerably simplifies the query access pattern that each service receives, and allows for a much more efficient use of classical scalability techniques.

References

[1] Achimugu et al., "Software Architecture and Methodology as a Tool for Efficient Software Engineering Process: A Critical Appraisal" Journal of Software Engineering & Applications, 2010, 3, 933-938 doi:10.4236/jsea.2010.310110 Published Online October 2010 (http://www.SciRP.org/journal/jsea)

[2] Bass et al., "Software Architecture in Practice," Addison Wesley, New York, 2003.

[3] Alder, B., "Building Scalable Applications in the Cloud: Reference Architecture & Best Practices" 2011 RightScale, Inc.

[4] Nolle, T., "Meeting performance standards and SLAs in the cloud." SearchCloudComputing, 2010. (http://searchcloudcomputing.techtarget.com/tip/0,289483,sid201_gci1357087_mem1,00.html)

Authors Biography

Philip Achimugu is a Lecturer of Computer Science at Lead City University, Ibadan, Nigeria. He holds B.Sc., M.Sc. in Computer Science and also pursuing a PhD degree in the same field. He has got about twenty-five (25) scholarly publications in reputable journals and learned conferences to his credit. His research interest borders on ubiquitous or pervasive computing.

Oluwatolani Oluwagbemi lectures in Computer Science department of Lead City University, Ibadan, Nigeria. She holds B.Sc., M.Sc. in Computer Science and also pursuing a PhD degree in the same field. She has over fifteen (15) reputable journal and learned conference publications. Her research interest is IT project Management and Entrepreneurship.

Ishaya Gambo is a lecturer of Computer Science at the Obafemi Awolowo University Ile-Ife, Nigeria. He holds B.Sc., M.Sc. in Computer Science and also pursuing a PhD degree in the same field and institution. He has got a good number of publications in reputable journals and learned conferences. His research interest is Software Architecture.

