Rapid Scalability of Complex and Dynamic Web-Based Systems: Challenges and Recent Approaches to Mitigation

Mani Malarvannan
Cybelink
www.cybelink.com
manim@ieee.org

S. Ramaswamy
Department of Computer Science
University of Arkansas at Little Rock
AR 72204. srini@ieee.org

Abstract
In this paper, we summarize and outline some of the big challenges in addressing performance, scalability, and availability of applications with large and complex backend database systems. We present some of the limitations imposed by the CAP theorem on distributed systems development and identify some approaches that can be used to effectively address these limitations. We propose the adoption of an effective System-of-Systems approach that pushes adoption of principles supported by loosely-coupled SPU based architecture, which can support incremental development and rapid scalability of such complex and dynamic applicative needs.

1. Introduction

The rapid evolution of Internet has created technology companies such as Google, Amazon, Facebook, Twitter, etc., which are unlike traditional multinational companies. These web-focused companies have one major business requirement that is unique - they need to mange several petabytes of data to efficiently run their daily operations. The existence of these companies depends upon the performance, scalability, and availability of their large and complex backend database systems. Traditional relational database design techniques cannot support these companies manage such massive amounts of data. Furthermore, such voluminous data cannot be stored and managed using a single server, or even a cluster of servers.

The Internet research firm IDC estimates that the digital universe has 281 exabytes of data and it will grow to be 1.8 zetabytes in 2011 [1], most of the exponential growth happening not due to traditional computing and communication equipments, but through mobile and embedded devices. For example, Google is estimated to currently process 20 petabytes [2] of data every day and this is growing rapidly. Amazon with their cloud computing solutions is storing petabytes of customer's data in their systems. Twitter gets more than 50 million tweets per day. For efficient management and service, they require thousands of servers located all over the world, to provide access redundancy, so their customers always experience very efficient service performance. Managing such vastly distributed data across thousands of servers is not a simple task and it needs an effective system of systems engineering principles to identify and fix the issues that arise dynamically in real-time. Of course, while most of the data will sit idle in disks and will only be retrieved when requested, the efficiency of retrieval when needed is of critical importance. Moreover, there are still several petabytes of data like email, blogs, web searches, online business transactions, etc. that are read and regularly updated by the internet community.

Web companies get thousands of concurrent requests either to read or update their data. They cannot store such data in small server farms to effectively serve their customers' needs. They need to store their data across multiply distributed servers that are often geographically separated in multiple locations. For example if Amazon gets a request for a book from a US host it will read it from one of their data center located in US to the customer's browser. Now consider another request comes for the same book from a customer in London it will be retrieved from Amazon data center located in London. Say both the customers add the book into their shopping cart and successfully processed the order. Now Amazon need to reduce the quantity of the book by two and it has to be replicate this update to all their servers. How can Amazon do this seamlessly without any face-losing failures?

One of the hallmarks of such web-based applications is the ability to handle request spikes that come nondeterministically from all over the world. Anytime a major event (like Haiti Earthquake) happens, the number of tweets to a Twitter site spikes drastically by several millions. During these unexpected events, users following other users send a flurry of requests to find what other users are tweeting about. If one can imagine a user following thousand of other users trying to find the tweets posted, the issue then is about how can one manage these sudden upsurges in web traffic?

Accepted for publication in the 5th IEEE International Conference on System of Systems Engineering

Among several popular Google applications is Google Analytics [3, 4], which serves a very important business need, especially for small businesses. This application embeds a small code snippet on web sites, thereby allowing companies to monitor and analyze web traffic on their sites. It is a free service offered by Google and millions of web sites are using it to perform real-time analysis of web site traffic. Such a strong dependency by businesses implies Google should provide these services consistently.

Once data is distributed and stored across multiple servers, companies need to worry about answering all the questions raised above in a cost effective and efficient manner. In such a complex system of multiply distributed server farms, single or multiple server failures are to be assumed as a daily occurrence, and companies need to identify and fix such server failures as they occur without causing any service disruption. This has become a huge issue for cloud computing environments; environments where companies manage data for other companies, subject to honoring specific service-level agreements (SLAs). Traditional processes, tools and technologies cannot help companies to manage their data stored over such a vast array of servers. They need to adopt system of systems engineering principles; specifically with respect to dealing with distributed databases, in order to achieve their goals.

The rest of the paper is organized as follows: Section II briefly presents RDBMS, their properties, various efforts to improve RDBMS for rapid scalability and some of their limitations. Section III presents the CAP theorem, BASE property and its importance to the development of such applications that have rapid 'exponential-scale' upswings in load / user driven activity. Section IV…, Section V ….

2. RDBMS and ACID Properties

RDBMS have been a de-facto standard for managing data for several years now. Based on the ACID properties (Atomicity, Consistency, Isolation, Durability), a RDBMS offered all the benefits needed to manage data in a single server or in a rack of servers. But RDBMS have failed to offer the benefits needed for web companies due to following reasons:

(i) RDBMS is built on a btree index that works well as long as the index can be stored in memory. Any time a request comes, index in the memory is used to find the location in the hard disk to either read or update the data. By avoiding the disk seeks RDBMS gave good performance to the users. But web based companies need to maintain large volume of data, and these indexes often cannot fit in the memory. Hence RDBMS need to do disk seeks to find data, an expensive operation that degrades the scalability of the application.

(ii) RDBMS vendor solutions to the above problem is to buy bigger servers with more RAM capacity, termed vertical scaling. Though this may work for some instances, it cannot completely address huge user spikes that are experienced by the web companies and ultimately it will be a performance bottleneck to support the simultaneous requests from a large number web customers.

While relational database systems are a good fit for applications that follows ACID properties, they are not appropriate for applications that undergo rapid surges in user activities. For example, in a trading application it is a fundamental requirement to inform the customer if a bid is successfully placed or not, an RDBMS may suffice. Any time a transaction fails due to one database instance failure then the entire transaction is abandoned. The data updated at one instance is either available in all the instances or it is not. For web-based applications, ACID properties are too restrictive to satisfy the associated business needs due to surges in user activity; hence a different set of properties must be established to achieve scalability and performance.

2.1. RDBMS Replication

RDBMS has been using data replication techniques to achieve performance and scalability. There are several types of replication available and each has its own set of limitations for web based applications. Different relational databases support different flavors of replication: Master-Slave, Multi-Master, Synchronous, and Asynchronous. For read-intensive applications, replication gives good scalability and performance but for write-intensive applications it cannot provide the desired scalability. Let us examine each of these in more detail in the sequel.

2.1.1. Master-Slave Replication: In this configuration, one master manages several slaves; while reads can happen from any slave, all writes must go through one Master. This restriction degrades scalability and performance. It additionally, leads to a single point of failure for all write operations.

2.1.2. Multi-Master Replication: In this configuration, more than one master manages several slaves, though write operations may scale, it will lead to manual conflict resolutions which are error-prone and problematic as the number of masters' increase.

2.1.3. Synchronous Multi-Master Replication: RDBMS vendors have been using two phase commit

transactions for a long time to guarantee ACID properties to support synchronous replication. In the two phase commit protocol, first a transaction manager asks each participating database instance to check if it can commit the transaction. If all the database instances agree then the transaction manager asks each database instance to commit the transaction. If any database instance vetoes the commit, then all database instances are asked to roll back their transactions. For web-based applications this cannot work as they maintain their data in multiple servers that could go down without warning, causing write failures and degrading availability of the application.

2.1.4. Asynchronous Multi-Master Replication: In asynchronous replication, when more than one master gets a write operation for the same data, both will succeed. In this case, the database has to resolve the conflicts based on timestamps on the data. Current relational databases do not support this feature out of the box. Even if they have the ability to support, such support is not automatic and manual intervention is needed to resolve the conflicts.

2.2. RDBMS and Sharding

Database sharding is a process in which the data is partitioned either horizontally or vertically and kept in different servers to increase performance and scalability. Although this concept has been around for a while, it has been popularized by Google engineers for improving the throughput and overall performance of high-transaction, large database-centric business applications Database vendors support two types of partitions, vertical partition and horizontal partition. In vertical partition a table is split vertically and kept in different servers and in horizontal partition different rows are kept in different database servers. When a read or write operation comes from a client it is routed to the server where the data resides to execute the operation.

The main problem with relational database partition is performing join queries, without partition all the join queries will execute in the same server. Once the data is shared across multiple servers, it is not feasible to perform join queries in a reasonable time that spans multiple servers. For example, if we are creating a partition for a twitter website in a relational database using horizontal partitions, one approach is to store the different set of users and their tweets in different servers. When a request comes to display all the twittes for a particular user, it is easy to read the data from one server and send it to the client. Twitter allows a user to follow other users, by following other users you can see their tweets. Now if a request comes from a user to see all the tweets posted by their "following users", we have to perform join query to join the twittes across all the servers where the "following users" data resides.

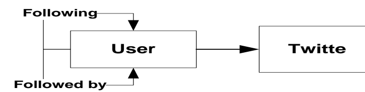

Figure 1. Twitter Information flows

3. Distributed Databases and Scalability

The Internet pioneers Amazon and Google created distributed database systems that follows Eric Brewer's CAP theorem [6] to satisfy their own needs, which are now widely used by other internet companies.

3.1. CAP Theorem: Simply put, the CAP theorem suggests that no distributed database systems can have all of the following properties of Consistency, Availability and Partition Tolerance, at the same time.

1. **Consistency**: Set of operations performed on distributed databases is perceived as a single operation by the clients. It is similar to "C" in ACID properties. In low volume transactions where the data is stored in single database, it is possible to achieve consistency. But for web based applications where the data is stored in multiple servers, it is not possible to achieve strict consistency.
2. **Availability**: The system must be always available to the clients
3. **Partition Tolerance**: All the operations on the distributed database will complete even if individual database instances are not available.

3.2. BASE: Distributed databases which have only two of the properties mentioned in the CAP theorem, are said to satisfy BASE properties, or **B**asically **A**vailable **S**oft State **E**ventually Consistent. BASE properties suggest that distributed systems are available all the time even if the individual database instances are not consistent and it will become consistent eventually. BASE is opposite to ACID, ACID is pessimistic and forces consistency on every operation. BASE, on the other hand, is optimistic and accepts the fact that the data will be in inconsistent state for clients, by relaxing the consistency distributed databases achieve scalability, availability, and performance which are the backbone for web applications. Of course BASE is not suitable for all types of applications, for example it is not possible to design an online trading system using BASE, one will need to use relational database that supports ACID. But there are web applications where it is acceptable to have BASE properties and for those distributed databases, it is fast becoming the standard.

3.3. Distributed Databases

Google created Big Table [7] and Amazon created Dynamo [8] to solve their own scalability, performance, and availability issues. Numerous open source projects created distributed databases based on these two technologies that are called in different names like key-value db, document store, extended record, etc. While, explaining all the projects and their differences is beyond the scope of this paper, we will explore the big two: Big Table and Dynamo to see how they achieve BASE properties.

3.3.1. Big Table : Google's BigTable is a multi dimensional map built on top of their propriety Google File System, GFS [9]. In BigTable data is arranged in rows and columns that can be accessed by giving row and column positions. To manage a huge table, it is split across row boundaries called Tablet. By default GFS replicates the data into three servers.

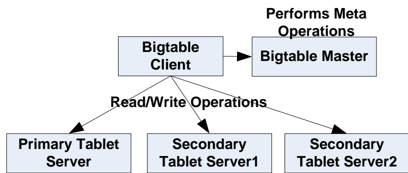

Figure 2. BigTable Architecture

BigTable has a Master server that manages the Tablet Servers and performs other Meta operations. For read operation BigTable Client sends a request to BigTable Master to get the location of the Primary and secondary Tablet Servers. Once the client gets the locations of the tablet servers, it directly contacts the closed Tablet server to read the data. For write operations once again the client contacts the Master server to get the locations of the tablet servers and sends the data directly to the Primary Tablet Server. The Primary Tablet Server appends the new data with its existing data and keeps it in memory. It then notifies the two secondary servers to perform the same merging operation. If everything succeeds the Primary Tablet Server notifies the client the success of the write. If any of the merging operation fails then the operation is repeated until the data is consistent across all the servers. Based on the above, it is clear that BigTable supports C and P properties of the CAP theorem. When the write operation is executed on the BigTable, it is guaranteed that all the data is in consistent state for subsequent read operations. If any of the merging mechanism fails then the write operation fails, by giving importance to Consistency and Partition Tolerance it reduces the availability of the BigTable. Though it is rare all the Tablet servers will fail but it is still possible.

3.3.2. Dynamo: Dynamo is Amazon's distributed key-value pair database used by several Amazon applications. Based on CAP theorem Dynamo gives importance to A and P, any time a client puts a value with an associated key, Dynamo adds version to the key-value and replicates it to other nodes within the same cluster.

Number of nodes that will be replicated is a configuration parameter. In the figure, Client1 executes a put command with key-value pair to Node6, if the replication configuration parameter is set to three; Node6 replicates the key-value along with version to Node1, Nod2, and Node3. Any time a Node gets a key-value with version, first it checks if it already has the key, if it finds the same key, checks if it is newer to the key it just received. If it is new, deletes the older version with newer version. This automatic reconciliation is called syntactic reconciliation. If more than one client reads a key-value concurrently and writes it back, Dynamo accepts both the write and stores all the updates from the clients. In this case, Dynamo has more than one version of the same key-value pair and it cannot reconcile automatically. When a read comes for the same key, Dynamo sends all the versions for that key and makes the client to do manual reconciliation which is called semantic reconciliation.

Dynamo's architecture is based on eventual consistency model, replication is performed asynchronously. Any time a client writes the key-value, even before it replicates the data to all the nodes, the write operation returns to the client. Amazon's business requires high scalability on read and writes operations, they get millions of concurrent requests to view their catalog and write the contents of the shopping cart. In the CAP properties Dynamo sacrificed C and has A and P.

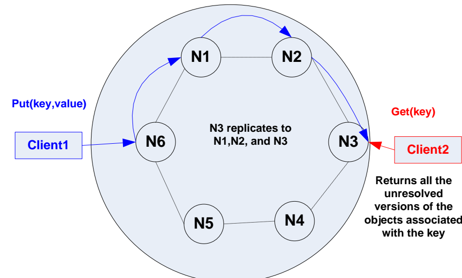

Figure 3. Dynamo Architecture

4. System of Systems Approach

Based on BigTable and Dynamo there are many open source projects which are being actively developed and some of them are used in commercial web applications. While it is true that the CAP theorem states it is not possible to achieve all three of C, A, and P

simultaneously, a using a system of systems approach where the architecture is dynamically reconfigurable can support all three properties adequately. For example, by setting the number of nodes to replicate the data asynchronously, we can have distributed DB which behaves like a RDBMS with strong consistency. Based on the applicative needs, distributed database can be modified from strong to weak consistency models and vice versa.

4.1. Software Processing Unit Architecture

Traditional multi layered architecture based approach has been used to architect large-scale complex software systems. In a multi layered architecture approach, the lower layer provides services to the upper layer. Any time a software object moves between layers it has to go through serialization, de-serialization and it has to pass through different networks which affect both performance and scalability. Multi layer architecture works well where the load on the system is predictable and provides vertical scalability. But the complex nature of web application need massive scalability with minimal cost and buying bigger hardware is expensive and hence we may quickly reach the upper limit on scalability.

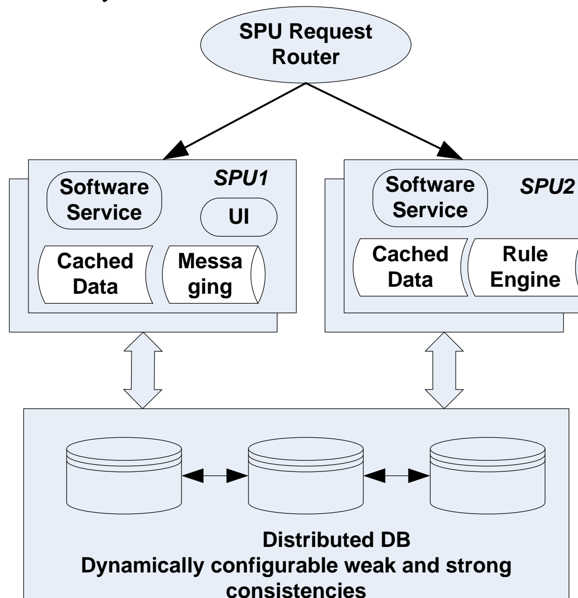

Figure 4. Proposed SPU Architecture

Building on similar concepts proposed by space-based [10] and shared-nothing architectures [11], and adopting a loosely-coupled systems approach, we propose a new architectural paradigm for web applications design, in which the application is built as stateless software services and deployed in its own server with all other software components are contained in a single module called Software Processing Unit, SPU. This SPU architecture, as depicted in the figure 4, will operate on top of a distributed database and provides both ACID and CAP properties based on the needs of the SPUs. Each SPU is a self contained unit that has its own cached data, messaging, rule engine, UI and other software components needed. Only time an SPU communicates with external resource is when it needs to either read or write data into the distributed database. SPU Request Router routes the incoming requests to appropriate SPU for processing. Once the connection is established between the client and an SPU all the communication happens between them directly.

4.1. Designing software for SPU Architecture

The SPU architecture collapses traditional multi-layer design into single module called SPU. This does not mean that UI, businesses services, data access logic must be designed into one single monolithic code. Individual components must be designed with high cohesion and low coupling. The only requirement is when deployed all the components for a SPU must collocate in a single physical server. Intra communication between components within a SPU will happen through messaging or local method invocation depending on asynchronous or synchronous nature of the call. Since there are no remote calls and network delays for the components inside a SPU, this design scales rapidly.

Only time a SPU communicates with external system is when it need to persist their data in a permanent store. For this purpose all SPUs run on top of a Distributed DB. Based on the needs of an individual SPU the distributed DB can have either weak or strong consistency. All the read operations from SPU to the Distributed DB happen without delay. Based on the need of the individual SPU write operations can happen either through weak or strong consistency.

4.2. Cloud Ready SPU Architecture

The loosely coupled nature of the SPU architecture allows it to be deployed on a cloud computing platform. The cloud allows SPU provisioning and virtualization based on the needs of external loads. The cloud monitors the individual loads on a SPU and either increases or decreases the number of instances of the SPU. The cloud can also monitor the load and switch the Distributed DB from weak consistency to strong and vice versa.

4.3. Limitations of SPU architecture

SPU architecture provides massive scalability for read intensive applications. For write intensive applications it provides good scalability compared with traditional multi-tier architecture that runs on relational databases, but it is limited by the communication between SPU and Distributed DB. Also write intensive applications

need to communicate the state changes between SPU instances so that the cached data will be in sync in all SPU instances.

5. Conclusion

In this paper, we have summarized and outlined some of the big challenges in addressing performance, scalability, and availability of applications with large and complex backend database systems. We have presented the limitations imposed by the CAP theorem and the approaches that can be used to address such limitations. We have articulated the use of a loosely-coupled system of systems approach to scale the software architecture further. Our future work will be focused on studies that validate the applicability of the SPU architecture approach with open source Distributed Database on industrial strength software systems and issues that arise therein.

6. Acknowledgments

This work is based in part, upon research supported by the National Science Foundation (under Grant Nos. CNS-0619069, EPS-0701890, OISE 0729792, CNS-0855248 and EPS-0918970). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

1. ---, "The Digital Universe Is Still Growing", EMC / IDC study, 2009. http://www.emc.com/digital_universe
2. J. Dean, S. Ghemwat, "MapReduce: simplified Data Processing on Large Clusters", Communications of the ACM, 51(1), Jan 2008, pp. 107-113.
3. M. Tyler, J. Ledford, "Google Analytics", JWS, 2006.
4. http://www.google.com/analytics/
5. Dan Pritchett, "BASE: An ACID Alternative", ACM Queue, July 28th 2008.
6. I. Filip, I. C. Vasar, R. Robu, "Considerations about an Oracle database multi-master replication", 5th International Symposium on Applied Computational Intelligence and Informatics, May 2009, pp 147 – 152
7. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", 7th Symposium on Operating System Design and Implementation, OSDI'06, pp 205-218.
8. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, 'Dynamo: Amazon's Highly Available Key-Value Store", ACM SIGOPS Operating Systems Review, 41(6), December 2007, pp. 205-220.
9. Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, the google File System, ACM SIGOPS Operating Systems Review, 37(5), December 2003, pp. 29-43.
10. C. Dumont, F. Mourlin, "Space Based Architecture for Numerical Solving" Intnl Conf. on Comp. Intel. for Modelling, Control & Automation, 2008. pp. 309-314.
11. J. Lifflancer, A. McDonald, O. Pilskalns, "Clustering Versus Shared Nothing: A Case Study", 33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC '09, pp. 116-121.