International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume1, Issue 9 (December '2015)

A COMPREHENSIVE STUDY ON CLOUD STORAGE SYSTEMS

V.Kalaivani ,
Assistant Professor,
Department of Computer Science,
Mahendra Arts and Science College,
Kallipatti, Namakkal, Tamilnadu.

N.Keerthi,
Assistant Professor,
Department of Computer Science,
Mahendra Arts and Science College,
Kallipatti, Namakkal, Tamilnadu.

Abstract: Cloud computing is still a rather new field, which is not yet entirely defined. As a result, many interesting research problems exist, often combining different research areas such as databases, distributed systems or operating systems. This paper focuses data storage Consistency Rationing as a new transaction paradigm, which not only allows defining the consistency guarantees on the data instead of at transaction level, but also allows for automatically switching consistency guarantees at run-time. We present a number of techniques that make the system dynamically adapt the consistency level by monitoring the data and/or gathering temporal statistics of the data. The last part of the paper is concerned with XQuery as a unified programming model for the cloud and, in particular, the missing capabilities of XQuery for windowing and continuous queries. XQuery is able to run on all layers of the application stack, is highly optimizable and parallelizable, and is able to work with structured and semi structured data.

Keywords: Cloud computing, data storage, XQuery, transactions

I. INTRODUCTION

Cloud computing is characterized by off-site access to shared resources in an on demand fashion. It refers to both, the service delivered over the Internet and the hardware and software in the data centers that provide those services [1][2]. Cloud computing allows companies to outsource the IT infrastructure and thus, profit from the economies of scale and the leverage effect of outsourcing. Cloud storage is an online virtual distributed storage provided by cloud computing vendors. Cloud storage services can be accessed via a web service interface, or a web based user-interface. One of the advantages is its elasticity. Customers get the storage they need, and they only pay for their usage. By using cloud storages, small organizations save the complexity and cost of installing their own storage devices. The same as cloud computing, cloud storage has also the properties of being agile, scalable, elastic and multi-tenant. This paper discusses cloud storage systems in more detail because of their fundamental role in building database application in the cloud. Here, we use the name cloud and database service interchange, as none of the database services in the cloud really offers the same comfort as a full-blown database and, on the other hand, cloudstorage services are extended with more functionality making them more than a simple storage service [3].

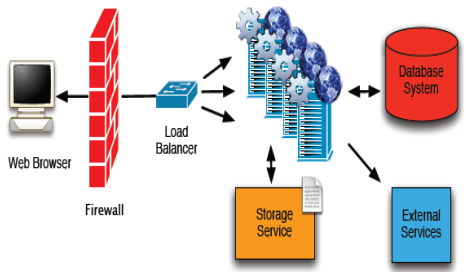

Figure1: Cloud Data storage Architecture

II. FOUNDATIONS OF CLOUD STORAGE SYSTEMS

The section explains the importance of the CAP theorem for developing cloud solutions before presenting some of the basic tools used to build cloud services.

a) The Importance of the CAP Theorem

To achieve high scalability at low cost, cloud services are typically highly distributed systems running on commodity hardware. Here scaling just requires adding a new off the-shelf server. Unfortunately, the CAP theorem states that it is not possible to achieve Consistency, Availability and tolerance against network Partitioning at the same time [4]. In order to completely avoid network partitioning, or at least to make it extremely unlikely, single servers or servers on the same rack can be used. Both solutions do not scale and, hence, are not suited for cloud systems. Furthermore, these solutions also decrease the tolerance against other failures (e.g., power outages or over-heating). Also, to use more reliable links between the networks does not eliminate the chance of partitioning, and increases the cost significantly. Thus, network partitions are unavoidable and either consistency or availability can be achieved. As a result, a cloud service needs to position itself somewhere in the design space between consistency and availability.

b) Consistency Guarantees: ACID vs. BASE

Strong consistency in the context of database systems is typically defined by means of the ACID properties of

transactions [5]. ACID requires that for every transaction the following attributes hold:

- Atomicity: Either all of the tasks of a transaction are performed or none.
- Consistency: The data remains in a consistent state before the start of the transaction and after the transaction.
- Isolation: Concurrent transactions result in a serializable order.
- Durability: After reporting success, the modifications of the transaction will persist.

If ACID is chosen for consistency, it emphasizes consistency while at the same time diminishing the importance of availability. Requiring ACID also implies that a pessimistic view is taken, where inconsistencies should be avoided at any price. As a consequence, to achieve ACID properties, complex protocols such as 2-phase-commit or consensus protocols like Paxos are required.

On the other extreme, where availability is more important than consistency, BASE[4] is proposed as the counter-part for ACID. BASE stands for: Basically Available, Soft state, Eventual consistent. Where ACID is pessimistic and forces consistency at the end of every operation, BASE is optimistic and accepts inconsistency. Eventual consistency only guarantees that updates will eventually become visible to all clients and that the changes persist if the system comes to a quiescent state [6]. In contrast to ACID, eventual consistency is easy to achieve and makes the system highly available.

III. CLOUD STORAGE SYSTEMS

Basic techniques to compare different cloud services. However, the focus here is on distributed algorithms. Standard database techniques (e.g. 2-phase-commit, 3-phasecommit etc.) are assumed to be known.

Master-Slave/Multi-Master: The most fundamental question when designing a system is the decision for a master-slave or a multi-master architecture [7]. In the master-slave model one device or process has the control over a resource. Every change to the resource has to be approved by the master. The master is typically elected from a group of eligible devices/processes. In the multi-master model the control of the resource is not owned by a single process; instead, every process/device can modify there source. A protocol is responsible for propagating the data modifications to the rest of the group and resolve possible conflicts.

Distributed hash-table (DHT): A distributed hash-table provides a decentralized lookup service[8]. Within a DHT the mappings from keys to values are distributed across nodes often including some redundancy to ensure fault tolerance. The key properties of a DHT are that the disruptions caused by node joins or leaves are minimized, typically by using consistent hashing , and that no node requires the complete information. DHT implementations normally differ in the hash method they apply (e.g. order preserving vs. random), the load-balancing mechanism and the routing to the final mapping . The common use case for DHTs is to load-balance and route data across several nodes.

Quorums: To update replicas, a quorum protocol is often used. A quorum system has three parameters: a replication factor N, a read quorum R and a write quorum W.A read/write request is sent to all replicas N, and each replica is typically on a separate physical machine. The read quorum R (respectively the write quorum W) determines the number of replicas that must successfully participate in a read (write) operation. That is, to successfully read (write) a value, the value has to be read (written) by R(W)numbers of replicas. Setting $R + W > N$ ensures that always the latest update is read. In this model, the latency of read/write is dictated by the slowest of the read/write replicas. For this reason, R and W are normally set to be lower than the number of replicas. Furthermore, by setting R and W accordingly the system is balanced between read and write performance. The quorums also determine the availability and durability of the system

Vector Clocks: A vector clock is a list (i.e, vector) of (client, counter) pairs created to capture causality between different versions of the same object [9].Thus, a vector clock is associated with every version of every object. Each time a client updates an object, it increments its (client, counter) pair(e.g., the own logical clock) in the vector by one. One can determine whether two versions of an object are conflicting or have a causal ordering, by examining their vector clocks. Causality of two versions is given, if every counter for every client is higher or equal to the counter of every client of the other version. Else, a branch (i.e., conflict) exists. Vector clocks are typically used to detect conflicts of concurrent updates without requiring consistency control or a centralized service [9].

Paxos: Paxos is a consensus protocol for a network of unreliable processors [10]. At its core, Paxos requires a majority to vote on a current state - similar to the quorums explained above. However, Paxos goes further and can ensure strong consistency as it is able to reject conflicting updates. Hence, Paxos is often applied in multi-master architectures to ensure strong consistency - in contrast to simple quorum protocols, which are typically used in eventually consistent scenarios.

Gossiping protocols: Gossiping protocols, also referred to as epidemic protocols, are used to multi-cast information inside a system [9]. They work similar to gossiping in social networks where a rumor (i.e., information) is spread from one person to another in an asynchronous fashion. Gossip protocols are especially suited for scenarios where maintaining an up-to-date view is expensive or impossible.

Merkle Trees: A Merkle tree or hash tree is a summarizing data structure, where leaves are hashes of the data blocks (e.g., pages) [1]. Nodes further up in the tree are the hashes of their respective children. Hash trees allow to quickly identify if data blocks have changed and allow further to locate the changed data. Thus, hash trees are typically used to determine if replicas diverge from each other.

IV. CLOUD STORAGE SERVICES

This section gives an overview of the available cloud storage services including open source projects that help to create private cloud solutions.

a) Commercial Storage Services

Amazon's Storage Services: The most prominent storage service is Amazon's S3[11]. S3 is a simple key-value store. The system guarantees that data gets replicated across several data centers, allows key-range scans, but only offers eventual consistency guarantees. Thus, the services only promise that updates will eventually become visible to all clients and that changes persist. More advanced concurrency control mechanisms such as transactions are not supported. Not much is known about the implementation of Amazon's S3. Internally, Amazon uses another system called Dynamo [10]. Dynamo supports high update rates for small objects and is therefore well-suited for storing shopping carts etc. The functionality is similar toS3 but does not support range scans. Dynamo applies a multi-master architecture where every node is organized in a ring. Distributed hash tables are used to facilitate efficient look-ups and the replication and consistency protocol is based on quorums. The failure of nodes is detected by using gossiping and Merkle trees help to bring diverged replicas up-to-date.

Google's Storage Service: Two Google-internal projects are known: BigTable and Megastore. The latter, Megastore, is most likely the system behind Google's App Engine storage service.

Google's BigTable [12] is a distributed storage system for structured data. Big-Table can be regarded as a sparse, distributed, persistent, multi-dimensional sorted map. The map is indexed by a row key, a column key, and a timestamp. No schema is imposed and no higher query interface exists. BigTable uses a single-master architecture. To reduce the load on the master, data is divided into so-called tablets and one tablet is exclusively handled by one slave (called tablet server). The master is responsible for (re-)assigning the tablets to tablet servers, for monitoring, load-balancing, and certain maintenance tasks. Because BigTable clients do not rely on the master for tablet location information, and read/write request are handled by the tablet server, most clients never communicate with the master.

Yahoo's Storage Service: Two systems are known: PNUTS [13] and a scalable data platform for small applications [13]. The first is similar to Google's Big-Table. PNUTS applies a similar data model and also splits data horizontally into tablets. In contrast to BigTable, PNUTS is designed to be distributed across several data centers. Thus, PNUTS assigns tablets to several servers across data center boundaries.Every tablet server is the master for a set of records from the tablets. All updates to a record are redirected to the record master and are afterwards propagated to the other replicas using Yahoo's message broker (YMB). The mastership of a record can migrate between replicas depending on the usage and thus, increases the locality for writes. Furthermore, PNUTS offers an API which allows the implementation of different levels of consistency, such as eventual consistency or monotonicity.

Microsoft's Storage Service: Microsoft offers two services: Azure Storage Service and SQLAzure Database [14]. Windows Azure storage consists of three sub-services: blob service, queue service, table service. The blob service is best compared to a key-value store for binary objects. The queue service provides a message service, similar to SQS, and also does not guarantee first in/first out(FIFO) behavior. The tablet service can be seen as an extension to the blob service. It allows to define tables and even supports a simple query language. Within Azure Storage Service data is replicated inside a single data center and monotonicity guarantees are provided per record but here exists no notion of transactions for several records. Little is known about the implementation, although the imposed data categorization and limitations look similar to the architecture of BigTable.

b) Open-Source Storage Systems

This section provides an overview about existing open-source storage systems. The list is not exhaustive and many other systems such as Tokyo Cabinet , MongoDB, and Ringo exist. However, those systems were chosen as they are already more stable and/or provide some interesting features.

Cassandra: Cassandra [15] was designed by Facebook to provide a scalable reverse index per user-ailbox. Cassandra tries to combine the flexible data model of BigTable with the decentralized administration and always-writable approach of Dynamo. To efficiently support the reverse index, Cassandra supports an additional three dimensionaldata structure which allows the user to store and query index like structures. For replication and load-balancing, Cassandra makes use of a quorum system and aDHT similar to Dynamo.

CouchDB: CouchDB [16] is a JSON-based document database written in Erlang.CouchDB can do full text indexing of the stored documents and supports expressing views over the data in JavaScript. CouchDB uses peer-based asynchronous replication, which allows updating documents on any peer. When distributed edit conflicts occur, a deterministic method is used to decide on a winning revision. All other revisions are marked as conflicting. The winning revision participates in views and further provides the consistent view. However, every

replica can still see the conflicting revisions and has the opportunity to resolve the conflict. The transaction mechanism of Couch DB can best be compared with snapshot isolation, where every transaction sees a consistent snapshot. But in contrast to the traditional snap shot isolation, conflicts do not result in aborts. Instead, the changes are accepted in different versions/branches which are marked as conflicting.

HBase: HBase [17] is a column-oriented, distributed store modeled after Google's BigTable and is part of the of the Hadoop project [The09c], an open-source MapReduce framework . Like BigTable, HBase also relies on a distributed file system and a locking service. The file system provided together with Hadoop is called HDFS and is similar to Google's File System architecture.

Redis: Redis [18] is a disk-backed, in-memory key-value store written in C. The most interesting feature is the data model. Redis not only supports binary strings, integers etc., but also lists, queues and sets, as well as higher level atomic operations onhem (e.g., push/pop/replacement of values in list). Redis does a master-slave replication for redundancy but no shard; thus, all data must fit in a single system's RAM.

Scalaris: Scalaris is an in-memory key-value store written in Erlang. It usesa modified version of the Chord algorithm to form a DHT, and stores the keys in lexicographical order, thus enabling range queries. Data is replicated using a quorum system. Furthermore, Scalar is supports transactions across multiple keys with ACID guarantees by using an extended version of Paxos. In a way, Scalaris combines the concept of Dynamo with Paxos and thus, offers strong consistency.

Project-Voldemort: Project-Voldemort [19] is yet another key-value store designed along the lines of Dynamo written in Java. However, it applies a layered architecture, which makes it possible to exchange the different components of the system. For example, it is easily possible to exchange the storage engine or the serialization method. The system seems to be in a reliable state and is in production at LinkedIn.

c) XQuery as the Programming Model

This section provides an overview of existing programming models for database applications. As cloud applications are typically accessed over the internet, the standard user frontend is web-based. Thus, for the following we will concentrate on web-based database applications.

V. PROGRAMMING MODELS OVERVIEW

The standard model for web-based applications is still a three-tier architecture, where the user interface, functional process logic and data access are developed and maintained as independent modules as demonstrated in Figure 2 [20]. On the different layers, different languages and data representations are used. On the client tier the standard format is HTML or XML which are then interpreted by the browser. To "program" the client the standard languages are JavaScript or Microsoft's ActionScript.The middle-tier renders the HTML or XML documents on request of the client.

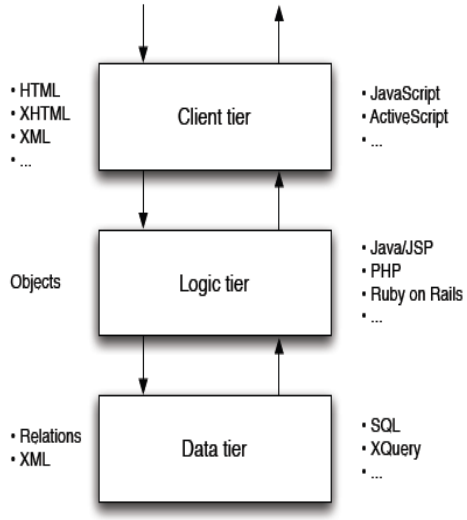

Figure 2: Application layers

The prominent languages for the middle-tier are Java/JSP, PHP, Ruby on Rails and Python. If the application is running inside the cloud, the middle-tier might be already a cloud service (e.g., a Platform as a Service)or hosted on a virtualized machine. The client typically interacts with the middle-tier by means of REST calls containing JSON or XMLdata. This data is then transformed to data types of the host language suchas objects. Furthermore, the middle-tier is often a layer consisting of several services which communicate over XML/JSON as well. To persist and access data, the data tier is responsible for using a structured(e.g., relational) or a semi-structured (e.g., XML) data model. To access the data declarative languages such as SQL or XQuery are used. The communication between the middle-tier and the data tier is normally done either with remote procedure calls (RPC) or by means of XML/JSON messages.

Extending the data-tier language to make it independent of a host language as done with PL/SQL and IBM's SQL PL has been quite a successful approach. Today, programming extensions are supported by most SQL implementations.SQL as a programming language has been used extensively in building many commercial applications including salesforce.com and the Oracle application suite. In general, industry experience suggests that it is easier to add a few carefully selected control flow operations to a database query language than to embed a foreign type system and persistence model into a procedural programming language.

Given the proliferation of XML data, XQuery [21] has recently been proposed as an alternative language which is able to run on all layers. XQuery is a declarative language particularly developed for XML, although not restricted to XML. In the following sub-section, XQuery is explained in more detail as a general programming model for web-based database applications.

XQuery : XQuery is a declarative programming language and builds on top of the XML Schema data types. Hence,XQuery is well-suited for parallelization and avoids the impedance mismatch between XML data types and the types of the hosting programming language. Since 2007, XQuery 1.0 is recommended by theW3C[21]. So far, almost fifty XQuery implementations are advertised on the W3C web pages, including implementations from all major database vendors and several open source offerings.

XQuery itself relies on several standards like XML Schema and XPath and has its own data model, whichis also shared with XPath and XSLT. In the XPath and XQuery DataModel (XDM), every value is an ordered sequence of zero or more items, which can be either an atomic value or a node. Anatomic value is one of the atomic types defined by XML Schema or is derived from one of those. A node can be a document, an element, an attribute, a text, a comment, a processing instruction or a namespace node. So an instance of the data model may contain one or more XMLdocuments or fragments of documents, each represented by its own tree of nodes. XQuery has several predefined functions and language constructs to define what and how to transform one instance to another. The most commonly known feature is the FLWOR (pronounced "flower") expressions, which stands for the keywords "ForLetWhere Order Return" and is the equivalent to "select from where order by" in SQL.

XQuery itself is defined as a transformation from one instance of the data model to another instance, similar to a functional programming language. This also allows connecting several XQueries to each other, as every result is also a valid input. Input data from outside the XQuery engine can be inserted applying functions such as document or collection, or by referencing to the external context (pre bound variables). Each of these methods then returns an XDM instance that can be processed by the XQueries.

XQuery for Web-Based Applications: By now, XQuery is already present in all layers of a web-application. At the data-tier, XQuery is supported by all mayor database vendors and several open-source implementations exist [21]. Thus, web-applications are already able to store data directly as XML and retrieve it using XQuery. In the middle-tier, XQuery serves several purposes. One prominent example is the transformation and routing of XML messages [20] between services. Another example is enterprise information integration. A third example involves the manipulation and processing of configuration data represented in XML. At the client-tier XQuery is not so established yet, but some initial projects are available to make XQuery also useable inside the browser. For example, by default Firefox allows to execute XPath (a subset of XQuery)inside JavaScript or the XQuery USE MEplug-in enables to execute user defined XQueries to customize web-pages. However, in the middle-tier as well as the client-tier XQuery is used inside a hosting language and therefore the impedancemismatch still exists.

XQuery as a complete programming language for the middle-tier has first been investigated inside theXL-platform in the context of web services . The XL platform provides a virtualized machine for XQuery code, several language extensions and a framework responsible for triggering XQuery programs based on events (i.e., a web service message). Today, the most successful XQuery-all solution is the Mark Logic Server . Mark Logic Server combines an XML database with an XQuery application server. Thus, the data- and middle-tier are combined in one server. Mark Logic Server is particular well suited for document-centric applications because of its full-text search and text analyzing capabilities but not restricted to those scenarios. Finally, theXQuery in the browser project at ETH ,investigates XQuery as an alternative to JavaScript making it possible to use XQuery at all layers and completely avoid the impedance mismatch.

VI.CONCLUSION

Cloud computing has become one of the fastest growing fields in computer science. It promises virtually infinite scalability and 100% availability at low cost. To achieve high availability at low cost, most solutions are based on commodity hardware, are highly distributed, and designed to be fault-tolerant against network and hardware failures.

However, the main success factor of cloud computing is not technology-driven but economical. Cloud computing allows companies to outsource the IT infrastructure and to acquire resources on demand. Thus, cloud computing not only allows companies to profit from the economics of scale and the leverage effect of outsourcing but also avoids the common over-provisioning of hardware.

REFERENCES

[1] Brian Hayes. Cloud Computing. ACM Communication, 51(7):9–11, 2008.

[2] The Economist. Let it rise: A special report on corporate IT. The Economist, 2008.

[3] Peter Mell and Tim Grance. Nist working definition of cloud computing. http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, Aug. 2009.

[4] Eric A. Brewer. Towards Robust Distributed Systems. In Proc. of PODC, page 7, 2000.

[5] Raghu Ramakrishnan and Johannes Gehrke. Database Management Systems. McGraw Hill Higher Education, 3rd edition, 2002.

[6] Yasushi Saito and Marc Shapiro. Optimistic Replication. ACM Comput. Surv., 37(1):42–81, 2005.

[7] Andrew S. Tanenbaum and Maarten Van Steen. Distributed Systems: Principles and Paradigms. Prentice Hall, 2 edition, 2006.

[8] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In Proc. of SIGCOMM, pages 149–160. ACM, 2001.

[9] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, andWerner Vogels. Dynamo: Amazon's Highly Available
Key-value Store. In Proc. of SOSP, pages 205–220, 2007.

[10] Tushar Deepak Chandra, Robert Griesemer, and Joshua Redstone. Paxos Made Live - An Engineering Perspective. In Proc. of PODC, pages 398–407, 2007.

[11] Amazon. Amazon Web Services: Overview of Security Processes. http://awsmedia.s3.amazonaws.com/pdf/AWS Security Whitepaper.pdf, Nov. 2009.

[12] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A Distributed Storage System for Structured Data. In Proc. of OSDI, pages 205–218, 2006.

[13] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. PVLDB, 1(2):1277–1288, 2008.

[14] Max Ross. Transaction Isolation in App Engine. http://code.google.com/appengine/articles/transaction isolation.html, Sep. 2009.

[15] Jason Lee. SQL Data Services - Developer Focus (Whitepaper). http://www.microsoft.com/azure/data.mspx, Jun. 2008.

[16] L. Youseff, M. Butrico, and D. Da Silva. Towards a Unified Ontology of Cloud Computing. In Grid Computing Environments Workshop (GCE08), 2008.

[17] Facebook. Cassandra. http://incubator.apache.org/cassandra/, Aug. 2009.

[18] The Apache Software Foundation. HBase. http://hadoop.apache.org/hbase/, Aug. 2009.

[19] The Apache Software Foundation. The CouchDB Project. http://couchdb.apache.org/, Aug. 2009.

[20] The Apache Software Foundation. The Apache Hadoop project . http://hadoop.apache.org, Aug. 2009.

[21] Don Chamberlin and Jonathan Robie. XQuery 1.1 - W3C Working Draft 3, Dec. 2008.

