VICTORIA UNIVERSITY OF WELLINGTON  
*Te Whare Wananga o te Upoko o te Ika a Maui*



School of Engineering and Computer Science

Te Kura Tatau

PO Box 600 Tel: +64 4 463 5341

Wellington Fax: +64 4 463 5045

New Zealand email: [office@mcs.vuw.ac.nz](mailto:office@mcs.vuw.ac.nz)

**Data Storage in Cloud Computing**

Harsha Subramania Raja

300206987

**Supervisor**: Dr. Pavle Mogin

Submitted in fulfilment of the requirements for

SWEN 440 Directed Individual Study

**Abstract**

Cloud data storage is an important and integral part of cloud computing, and has introduced a wave of change in the way data is stored today. Databases on cloud adopt data models and database architecture that are still evolving and very different from the popular and traditional data models like relational data model. This study reviews the different approaches cloud databases take to data organisation, by studying a few popular and widely used cloud databases. Cloud databases Bigtable, Amazon Dynamo, Cassandra, HBase and Microsoft SQL Azure are studied as a part of understanding general trends and architecture in cloud data storage. One of the databases, Cassandra is installed and used briefly to learn about accessing and using cloud data storages.This study also covers the fundamental differences between cloud databases and the traditional Relational Database Management Systems.

Table of Contents

[Introduction to the Research 6](#_Toc285791961)

[Introduction to Cloud Computing 7](#_Toc285791962)

[1.1 What is Cloud Computing? 7](#_Toc285791963)

[1.2 Layers of Cloud Computing 7](#_Toc285791964)

[1.3 Advantages of Cloud Computing 8](#_Toc285791965)

[1.4 Disadvantages of Cloud Computing 8](#_Toc285791966)

[Cloud Data Storage 10](#_Toc285791967)

[2.1 What is Cloud Storage? 10](#_Toc285791968)

[2.2 How Cloud Storage Works 10](#_Toc285791969)

[2.3 Cloud Storage and Data organisation 11](#_Toc285791970)

[2.4 Cloud Databases 11](#_Toc285791971)

[2.5 Advantages of Cloud Storage 12](#_Toc285791972)

[2.6 Disadvantages of Cloud Storage 13](#_Toc285791973)

[Cloud Data Storage: Cloud Data models 14](#_Toc285791974)

[3.1 Key Value Data Model 15](#_Toc285791975)

[3.2 Document Data model 17](#_Toc285791976)

[3.3 Relational Data Model 18](#_Toc285791977)

[3.4 Summary 19](#_Toc285791978)

[Bigtable 20](#_Toc285791979)

[4.1 Introduction 20](#_Toc285791980)

[4.2 Bigtable Architecture 20](#_Toc285791981)

[4.3 How Bigtable works 23](#_Toc285791982)

[4.4 Summary 25](#_Toc285791983)

[Amazon Dynamo 26](#_Toc285791984)

[5.1 Introduction 26](#_Toc285791985)

[5.2 Dynamo Architecture 26](#_Toc285791986)

[5.3 How Dynamo works 29](#_Toc285791987)

[5.4 Summary 30](#_Toc285791988)

[Cassandra 31](#_Toc285791989)

[6.1 Introduction 31](#_Toc285791990)

[6.2 Cassandra Architecture 31](#_Toc285791991)

[6.3 How Cassandra works 33](#_Toc285791992)

[6.4 Summary 33](#_Toc285791993)

[HBase 34](#_Toc285791994)

[7.1 Introduction 34](#_Toc285791995)

[7.2 HBase Architecture 34](#_Toc285791996)

[7.3 How HBase works 37](#_Toc285791997)

[7.4 Summary 37](#_Toc285791998)

[Microsoft SQL Azure 38](#_Toc285791999)

[8.1 Introduction 38](#_Toc285792000)

[8.2 SQL Azure Architecture 38](#_Toc285792001)

[8.3 How SQL Azure works 40](#_Toc285792002)

[8.4 Summary 41](#_Toc285792003)

[NoSQL vs. RDBMS 42](#_Toc285792004)

[9.1 Introduction 42](#_Toc285792005)

[9.2 NoSQL vs. RDBMSs 42](#_Toc285792006)

[9.3 Summary 47](#_Toc285792007)

[Conclusion 48](#_Toc285792008)

[Appendix A 50](#_Toc285792009)

[Appendix B 53](#_Toc285792010)

[Cassandra Installation 53](#_Toc285792011)

[Add and retrieve Data from Cassandra 55](#_Toc285792012)

[Appendix C 57](#_Toc285792013)

[Appendix D 58](#_Toc285792014)

[Bibliography 59](#_Toc285792015)

**Table of Figures**

[Figure 1: Basic Cloud Computing Diagram 7](#_Toc285791042)

[Figure 2: Database Diagram for Student Database 14](#_Toc285791043)

[Figure 3: Sample data in Student database. 14](#_Toc285791044)

[Figure 4: JSON Column representation 15](#_Toc285791045)

[Figure 5: Column example 15](#_Toc285791046)

[Figure 6: JSON SuperColumn representation example 15](#_Toc285791047)

[Figure 7: A SuperColumn example 16](#_Toc285791048)

[Figure 8: A JSON ColumnFamily representation example 16](#_Toc285791049)

[Figure 9: A ColumnFamily example 16](#_Toc285791050)

[Figure 10: A JSON SuperColumnFamily representation example 17](#_Toc285791051)

[Figure 11: A SuperColumnFamily example 17](#_Toc285791052)

[Figure 12: A JSON representation for Document Database 18](#_Toc285791053)

[Figure 13: Splitting a table into tablets 21](#_Toc285791054)

[Figure 14: Bigtable Timestamp example 21](#_Toc285791055)

[Figure 15: SSTable (Courtesy: Michalis, 2009) 21](#_Toc285791056)

[Figure 16: SSTable structure in tablets 22](#_Toc285791057)

[Figure 17: Metadata tablets (Courtesy: Chang et al., 2006) 22](#_Toc285791058)

[Figure 18: Master Server initiation steps 23](#_Toc285791059)

[Figure 19: Write request in Bigtable 24](#_Toc285791060)

[Figure 20: Read request in Bigtable 24](#_Toc285791061)

[Figure 21: A ring of nodes in Dynamo 26](#_Toc285791062)

[Figure 22: Consistent hashing in Dynamo 27](#_Toc285791063)

[Figure 23: A preference list example 27](#_Toc285791064)

[Figure 24: Eventual Consistency in Dynamo 28](#_Toc285791065)

[Figure 25: Vector Clock (Courtesy: DeCandia et al., (2007)) 28](#_Toc285791066)

[Figure 26: Hinted Handoff in Dynamo 29](#_Toc285791067)

[Figure 27: Seed node in Dynamo 29](#_Toc285791068)

[Figure 28: Handling user requests in Dynamo 30](#_Toc285791069)

[Figure 29: A Cluster of nodes in Cassandra 31](#_Toc285791070)

[Figure 30: Single read in Cassandra 32](#_Toc285791071)

[Figure 31: Quorum read in Cassandra 32](#_Toc285791072)

[Figure 32: Write request in Cassandra 33](#_Toc285791073)

[Figure 33: Regions in HBase 34](#_Toc285791074)

[Figure 34: Meta Region Representation 35](#_Toc285791075)

[Figure 35: Assigning Regions in HBase 35](#_Toc285791076)

[Figure 36: HRegionServer Failure 35](#_Toc285791077)

[Figure 37: Root Directory for HBase 36](#_Toc285791078)

[Figure 38: Handling User Requests in HBase 37](#_Toc285791079)

[Figure 39: Database Creation in SQL Azure (Courtesy: SQLAzure, 2011) 38](#_Toc285791080)

[Figure 40: Database Manager in SQL Azure (Courtesy: SQLAzure, 2011) 38](#_Toc285791081)

[Figure 41: Table Creation in SQL Azure (Courtesy: SQLAzure, 2011) 39](#_Toc285791082)

[Figure 42: Select result in SQL Azure 39](#_Toc285791083)

[Figure 43: Azure Platform architecture 39](#_Toc285791084)

[Figure 44: SQL Azure Layers (Courtesy: Campbell et al. (2010)) 40](#_Toc285791085)

[Figure 45: Horizontal partitioning in SQL Azure 40](#_Toc285791086)

[Figure 46: Cassandra Installation 53](#_Toc285791087)

[Figure 47: Configuration Folder 53](#_Toc285791088)

[Figure 48: “bin” folder 53](#_Toc285791089)

[Figure 49: Cassandra.yaml 54](#_Toc285791090)

[Figure 50: Cassandra in running mode 54](#_Toc285791091)

[Figure 51: Thrift client 55](#_Toc285791092)

[Figure 52: storage-conf.xml 55](#_Toc285791093)

[Figure 53: Set Data 55](#_Toc285791094)

[Figure 54: Get Super Column 56](#_Toc285791095)

[Figure 55: Get Column 56](#_Toc285791096)

Introduction

# Introduction to the Research

Cloud Computing has been gaining popularity over the years, with many companies and users using cloud services. Cloud computing offers economic benefits as storage, infrastructure, Database Management Systems (DBMS) etc. are available as cloud services and users pay only for the services they use and do not pay for any unnecessarily features or services.

While cloud computing has been evolving and adapting to offer many different services, an important part of cloud computing that has been progressing is the cloud data storage. This paper looks at understanding the data storage on cloud computing and studies a few cloud DBMSs currently popular and widely used. Understanding the cloud databases requires a good understanding of the underlying data models and architectures such databases are built upon. Data models and database architecture on the cloud is a lot different from the traditional data models and architectures as the features of cloud computing is greatly different. This paper first understands the basic concepts of Cloud computing in Chapter 1 and then studies the concepts of Cloud data storage in Chapter 2.

Since the data models are varying in the different cloud databases, some of the key data models like Key Value data model, Document data model etc are studied in Chapter 3. This gives a foundation to understand the various cloud DBMSs like Bigtable, Cassandra etc. Each of these DBMSs are described in terms of its database architecture and the way the DBMS works along with some case studies of users who have implemented and benefitted from these DBMSs. Bigtable is described in this fashion in Chapter 4, Amazon Dynamo in Chapter 5, Cassandra in Chapter 6, HBase in Chapter 7 and Microsoft SQL Azure in Chapter 8. The user case studies of cloud DBMSs is given in Appendix A. Appendix C includes the brief description of Windows Azure Platform architecture.

As a part of understanding the practical aspects of installing and using a cloud database, Cassandra was installed and tested with small amounts of data. Details are provided in Appendix B. Cassandra was chosen as the other databases included in the study either involved usage costs or the database were not available for public to use. Bigtable and Dynamo are currently not available for public users and are limited to being an in-house database for its developers. Microsoft SQL Azure and HBase are available for public users to use like Cassandra, but Microsoft SQL Azure involved usage costs. Deploying and using both HBase and Cassandra for this study would have caused time constraints and hence it was decided to deploy only one of these cloud databases.

Cloud DBMSs are generally termed as NoSQL (Not Only SQL) in the world of cloud computing. The NoSQL DBMSs are fundamentally different from the traditional and popular Relational Database Management Systems (RDBMS). Readers are expected to have sound knowledge of relational model and RDBMSs and SQL Server .A comparison of NoSQL DBMSs and RDBMS is given in Chapter 9. This comparison was based on the study done to understand the various NoSQL DBMSs. While comparing the NoSQL DBMSs and RDBMSs, a comparison with the current popular cloud RDBMS, Microsoft SQL Azure is also included. This is primarily to understand the way in which cloud RDBMSs are being evolved to suit cloud computing and cloud data storage. This also helps in understanding the reasons why RDBMSs are not considered suitable for the cloud currently by many users and prominent researchers.

The conclusion of the entire study is given in Chapter 10 with bibliography and Appendix following Chapter 10.

Chapter 1

# Introduction to Cloud Computing

## What is Cloud Computing?

Cloud computing, a major paradigm, would bring about a shift in the way IT services and tools are going to be used in the industry. It is perceived that cloud computing would help extend the capabilities of many IT and online services without the need for costly infrastructure. According to IDC (2008), the spending on IT cloud services is likely to triple in the next 5 years, indicating that cloud computing is here to stay.

Cloud computing, a by-product of remote computing where other machines or computers are accessed from the local machine through a network, brings with it the virtualisation of applications and services. Virtualisation gives users the feeling that the applications are running on the user’s machine rather than the remote cloud machine (Cloud Computing Defined, 2010), removing the need for installing the actual software by the users. Both expert and naive users could thus work with applications without worrying about the technical details and configurations.

Cloud computing usually is a subscription based model where users would pay as per their usage, making cloud computing very similar to utility services like electricity, gas or water etc. The coalescence of virtualisation, where applications are separated from the infrastructure is what makes cloud computing easy to use. Users need not invest in buying software applications as they can access such applications on the cloud. Cloud service providers have large server farms where applications and databases are stored. Users only pay for the services they use. For example, they pay only for the amount of storage their cloud database uses or pay only for the bandwidth consumed by the servers they rent from the cloud providers. This is most beneficial for the medium and small enterprises as they need not have huge investments in databases or servers.

The architecture of cloud computing services has users who avail cloud services as the front-end. The back end of the architecture includes the cloud servers, databases, and computers etc., which are abstracted from users (Figure 1). All the components like the servers, applications, the data storages work together through a web service to provide the users with the cloud services.

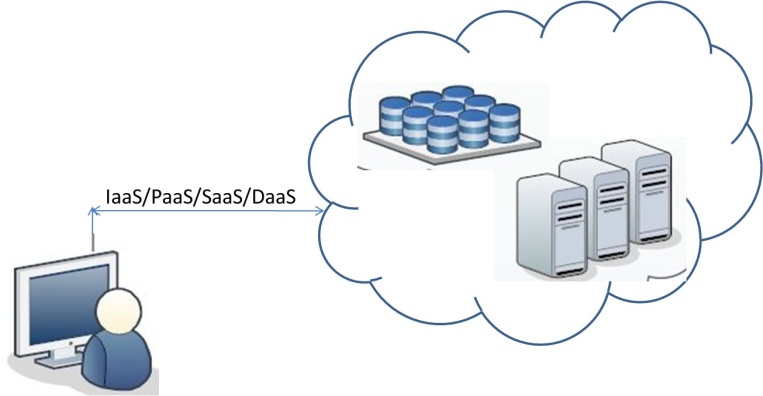


Figure : Basic Cloud Computing Diagram

## Layers of Cloud Computing

The concepts of cloud have been generalised into five basic layers that provide different services to the users and helps in understanding the overall structure of cloud computing (Zaki Sabbagh, 2010, Bime, 2008). The five layers are described below:

1. *User*: This could be any hardware or software application that relies on cloud computing to do its work.
2. *SaaS*: means Software as a Service, where the cloud providers give software as the service so that users do not have to install these software applications.
3. *PaaS*: means Platform as a Service, where a hardware or software platform is given as the service to users. Here, a platform could be an operating system, programming environment, hardware, runtime libraries etc.
4. *IaaS*: means Infrastructure as a Service. The users could use the expensive hardware like network equipments, servers etc as a service and only pay for what they use.
5. *Server*: is the key component which is specifically designed to support the hardware and software needed for the cloud computing to be run. This could be the cloud operating systems, processors, individual machines etc.

Throughout the chapters the term ‘client’ refers to any software applications or APIs (Application Program Interface) that is used to perform cloud computing, while ‘users’ refers to the end-users, like database administrators or programmers or any user who benefit from cloud computing services.

## Advantages of Cloud Computing

While economic gains are a major benefit, some of the other benefits achieved by using cloud computing are:

* *Maintenance*: Since applications and data are stored at one place on one of the servers of the service providers, any updates to the software are given only at a single place where the application is remotely saved. Users of the services do not have to worry about having the updates done on time or install it themselves as it is the responsibility of the service provider (Zdnet, 2010).
* *Security*: Since users are relieved of the cost of infrastructures, they could invest more in security measures, like securing their network, or databases etc (Nubifer, 2010). Since cloud databases are centralised and stored on a single server farm, securing such a single point is easier than having to secure individual user databases. But the failure of such a centralised single point could affect the many users using the cloud database.
* *Accessibility*: Cloud computing provides a common place to store data and applications ,giving users simple and easy access to data and applications (IDC, 2010). Users could use the cloud services from anywhere as long as they are connected to the internet with sufficient access privileges. Users do not have to be connected to any specific local intranet or be in a specific location to access their applications or databases. So people of different departments could also access other department’s documents without any network problems.
* *Greener*: Resources, like servers and hardware, which are large power consumers and heat generators are shared on cloud. Having such resources shared among hundreds of users reduces the number of individual resources the users would have owned otherwise. This in a way becomes eco-friendly in the long run.
* *Sharing*: As mentioned before, applications and resources are widely shared and used by many users. If resources were not shared, users would have had individual resources and these resources would have been idle when not in use. But in the cloud these resources are reallocated to other users, enabling reuse and sharing.

## Disadvantages of Cloud Computing

* *Lack of control*: Since applications and data are stored in a remote server farm, users do not have any administrative control or physical access to such resources. Users cannot control the security of their cloud databases and nor can they implement their company specific rules or security filters.
* *Trust*: Cloud service providers should be trustworthy so that users can rely on their services. Perhaps security certificates and verifications of the credentials of service providers might be useful so that the users can be assured about the safety of their data.
* *Connection dependency*: Dependency on the internet could be another problem for users. Internet connections are not always reliable, causing poor data accessibility and loss of some data (Zdnet, 2010). Not all service providers provide the feature of using the data while being offline. Unless internet becomes more reliable, cloud computing can be unreliable for most users for this reason.
* *Efficiency*: The performance of cloud applications may not always be as efficient as local applications as cloud computing may not always support all the features of local applications (Zdnet, 2010). Many features of locally stored applications or DBMSs may not be viable in a cloud computing environment, due to the distributed nature of cloud computing.

Chapter 2

# Cloud Data Storage

## What is Cloud Storage?

More and more businesses and users avail cloud services, owing to the benefits offered by the cloud services (like low costs, sharing etc). These benefits are some of the reasons why cloud data storage is also gaining popularity (SNIA, 2009).

Many definitions for cloud storage are available today. According to Steve Lesem (2009), cloud storage is “File Storage accessed through Web Services API's over a network”. This definition implies that users can store their data as a file and to access data, users would communicate to the cloud storage systems through the internet. Cloud storage systems provide APIs to users for simple and easy data access. As cloud storage improves and as cloud storage providers include better features to cloud DBMSs, definitions also tend to change (Lesem, 2009). Today, cloud data storage is synonymous to cloud databases, which is explained in Section 2.4.

Cloud applications handle user data, which could be in the form of storing user information or sending data to users upon requests. In such cases, storage of application or user data is necessary (Kennedy, 2009). Traditionally, users would store data in files or databases on dedicated database servers or on local disks. In cloud computing, data is stored within data centres. Data centres house many servers, computers and telecommunication infrastructure, including back up and security facilities. Such data centres can be owned by hosting companies like Google, Amazon etc. and users can rent or buy the storage space they need.

Cloud databases need to be scalable across the several servers that might be used for data storage. Scalability in the context of cloud storage often refers to the ability of dynamically incorporating changes to the number of users or storage space, without affecting the functioning of the databases or the availability of data to the users. In other words, even when more machines are added to increase storage capacity, or when more users access the same data, cloud databases should cope with the increased workload and yet maintain the same throughput.

Cloud DBMSs provide APIs to users to manage organise and access their stored data. Most cloud DBMSs keep APIs simple so that it remains scalable to the many different and diverse users. Removing complexity and keeping the API simple is a daunting task when the storage grows and when the user-base includes diverse users. User-base in general refers to the number of users using a database or any cloud service. (Cooper, 2010).

The cloud storage service, Database as a Service (DaaS) or Storage as a Service (StaaS) refers to storage facilities, like DBMSs, that are offered as cloud services (Wu et al., 2010). Cloud databases are hosted in the cloud on a pay-as-you-go model, just like SaaS or PaaS, and offer data management, data retrieval, etc. Users pay only for the storage space they use. There are many DaaS service providers currently, like Amazon, Google, IBM, Microsoft etc (Mateljan et al., 2010). More details about DaaS and cloud DBMSs are provided in Section 2.5.

## How Cloud Storage Works

Cloud storage involves storing data on remote machines on the cloud that are owned or maintained by third parties, instead of storing data on local machines. Users connect to cloud storage systems through the internet.

In most data centres the storage space is managed dynamically, i.e., cloud DBMSs would allocate the storage space for data on any of the available servers or nodes in data centres and can even move the user data to other servers in other data centres. Users of cloud DBMSs are unaware of the exact location of their stored data and are guided to the location by the APIs, making data access easy and simple (Wu et al., 2010). Dynamic allocation of resources reduces the cost involved in data storage from the user’s perspective as storage space is efficiently used and managed.

Most cloud databases partition data, which means that data is split into distinct individual parts and saved on different nodes in the data centre across several databases. Thus, nodes would have a subset of data or rows from each table in the database (DeWitt et al., nod.).This eventually means that querying would take longer time as the data is spread across several databases, possibly on different servers and would include multiple joins on the datasets.

## Cloud Storage and Data organisation

Data storage on the cloud can be organised into Static File storage, Database storage, Cache storage, which are explained briefly below. Such organisation methods organise data of various types like files, objects, databases, cache data etc.

* *File Storage*: According to IDC (2008b), most of the machines (nearly 70%) store data in the form of files or objects that are commonly referred to as unstructured data. Some researchers claim that cloud storage is primarily suited for file storage mainly for this reason and it is ideal for file storage in cloud networks that are accessible through APIs over the Internet (Lesem, 2009).

Small file requests that can be handled in a single request call to the data storage is consumed quicker by users, while large files are broken to manageable bits before it is accessed by users. Cloud data storage can suit such needs and provide access to most kinds of files and generally require a robust and simple API.

Some of the current cloud data storage services using this kind of organisation are (Kennedy, 2009): Google App Engine Data Store, S3 (Simple Storage Service by Amazon) etc.

* *Database Storage***:** To store structured data, databases are considered by some researchers as the best preferred solution (Mateljan, 2010). Cloud databases have been made scalable to support the diverse and large number of users who store structured data and to support various applications that users use. Today, cloud databases are replicated, distributed, simplified and often specialised (Cooper, 2010). Cloud databases are replicated so that multiple copies of data are available to cater to many users who access the same data at the same time. This also helps in cases of server crashes or network failures, as copies of the data are available. Just like cloud networks, cloud databases are distributed and reside on more than one node in a data centre. Cloud databases are specialised as certain cloud related problems are addressed. For example, some cloud databases are built to provide high scalability while others are built to store huge amounts of interconnected data. More about such cloud databases are explained later.

Unlike traditional DBMSs, cloud DBMSs is simple in their structure with minimum querying support and have a simple API for users.

Cloud database storage is utilised by Amazon EC2, Microsoft SQL Azure, and Amazon Dynamo, Cassandra etc.

* *Cache*: The API for the cloud storage (either file storage or cloud databases) would in real time receive multiple requests for the same data from various users. It becomes essential that the cloud maintain a cache for these requests just like the local cache management on any machine (Kennedy, 2009). Some of the current services implemented for this are Memcache used in Google App Engine, Velocity caching in Microsoft Azure etc. (Kennedy, 2009).

## Cloud Databases

According to Mateljan et al. (2010), cloud databases fall into three categories:

* *Cloud Native RDBMS***:** These RDBMSs provide relational databases on cloud and provide users with database administration facilities and APIs to scale relational databases and to perform operations on stored data, like updating, inserting, deleting data etc. The cloud RDBMSs offered today vary according to the vendors and each of the vendors propose alternative solutions to problems like scalability and latency. For example, Amazon Relational Data Service provides the users with an RDBMS that automates the administration tasks to hide the complexity of cloud data storage. Most cloud RDBMS automate most of the database management tasks. Users hence do not have to worry about installation or updating their database software.
* *Cloud Native Non***-***relational Databases***:** Such databases provide simple querying and indexing features in the database and have a high rate of redundancy as data is not normalised like in RDBMS. Since storage is abundant and cheap in the cloud, this is not a potential problem. Amazon SimpleDB uses this model and allows users to store and retrieve their data through simple web requests. It automatically does the indexing of the user data, replicates it so that copies are available in case of any failure and also performs database tuning for the users.
* *Cloud Capable Relational or Non-Relational databases:* Virtual machines are used with various platforms and databases to provide complete control over the database servers. This also avoids the actual installation or deployment of expensive database servers or other infrastructure. Some of the cloud capable relational databases are IBM DB2, MySQL, Sybase etc, while MongoDB, CouchDB, Cassandra etc are cloud capable non-relational databases.

In general, cloud DBMSs are found to be less efficient than traditional databases because of the dynamic scalability required by cloud databases to support a changing user-base (Hogan, 2008). Hogan (2008) claims that data partitioning in cloud databases increases complexity as a database is spread across several servers and querying the database would involve complex Joins and more time. This moves the databases and the user applications farther apart, increasing latency (Murphy, 2010). Murphy (2010) suggests that cloud bursting could be a possible alternative to avoid problems due to such latency. Cloud bursting involves having the user use the cloud data storage only when additional capacity is needed by the user. This would mean that a local copy of the database would have to be maintained by the user along with the cloud database. Users are redirected to the correct copy of data appropriately whenever the switch is made between the local and cloud databases.

## Advantages of Cloud Storage

There are many advantages of having data stored on the cloud, which are mostly similar to the benefits of cloud computing in general. Some of the key benefits are discussed below:

* *Easy Management*: Using cloud data storage removes the complexity of data maintenance and administration tasks from the user side. Updating the cloud DBMS with new security patches or new versions are done by the cloud database provider and users only need a web browser and an internet connection.
* *Cost Reduction*: By using cloud storage, the entire cost of deploying a cloud DBMS, updating it, managing it, is removed from the user’s side. Users need not invest in database servers or other infrastructures to store their data. Using cloud storage also removes the extra people needed to maintain databases too. By leasing storage users pay only for the storage used.
* *Recovery and Backups*: As explained previously, cloud databases store data at multiple servers, allowing backup copies to be preserved and these copies are made available to users even at times of any failures.
* *Planning reduced*: Planning for a database management is removed when users use cloud data storage services. This gives the users more time to plan for application development or project budgeting etc.
* *Controlled management*: The data storage is centrally managed by the cloud storage service provider, ensuring that updates and hardware/software upgrades are performed periodically.
* *Sharing*: When users store their data in cloud, they can share their data with any number of users or stakeholders they wish to. With the right access levels outsiders can view the data and perhaps even reuse the data for their applications as well.
* *Specialisation*: Some cloud databases store specific types of data, like images or objects or files etc. This gives users precise choice in selecting the right kind of database they would like to integrate with their applications, giving users users flexibility .Also, some cloud databases address certain problems in data storage. Users can thus choose the right kind of cloud database to suit their needs.

## Disadvantages of Cloud Storage

* *Data security*: In cloud databases, data is remotely stored on different nodes, with users having minimal control over data management. Thus, there is no guarantee of the security and privacy of data.
* *Integrity*: In real time, a cloud database can be accessed by many users at the same time. Although locking mechanisms and read-only features are enabled in cloud databases, there is always the problem of ensuring consistency of data.
* *Latency*: When data and applications are not stored in the same location, there could be potential delays in data retrieval and data updating. This gives scope for data corruption and failure in committing transactions too.
* *Complex APIs*: Most of the APIs offered by the DaaS and other cloud storage mechanisms are in their nascent stages now and larger data and increase in the number of users augment the complexity of these APIs. More security-related features as well as other added features creep in making the underlying storage system hard to access.
* *Limited functionalities*: Currently, most cloud DBMSs offer only limited querying, limiting the functionalities available to cloud databases users.
* *Reliability*: Since data is remotely stored in data centres managed by third parties, users have to be dependent on them for data integrity and security. This is an extremely risky step for organisations to undertake.
* Vendor lock-in and control over the performance of the cloud storage system is also out of the hands of the users (Wu et al., 2010).

Chapter 3

# Cloud Data Storage: Cloud Data models

Data models describe the structure of a database and give the users information on how a database can be used or implemented. On the cloud different types of data models are prevalent. The selection of a data model for a cloud database depends on the problem the cloud database is trying to address or a feature it is incorporating. Some of the current popular data models on the cloud are:

* Key Value data model (or Key-Value Store data model)
* Document data model
* Relational data model

Cloud databases also include Graph databases that are based on the graph theory model where data is stored as a graph with edges, nodes and some properties of the data. These databases are faster and better scalable than relational databases since graphs have tree structures that can be traversed, thus requiring less Join operations or extensive queries (Denman, 2010). Details of Graph databases can be found in Denman (2010).

Throughout the report, the data models and the different cloud DBMSs are explained using a student database (), where data about Students, Courses and Faculty in a school are saved and maintained in the traditional relational model, a separate table would be created for students, faculty and courses. Each table would have primary keys to maintain the uniqueness of the data; students have a unique “StudID”, Faculty have “FacultyID”, and courses have “CourseID”. There would be tables that maintain the relationship between these tables. For example, to show the courses taken by a student, there would be a table that maintains the relationship between the student IDs and course IDs. Similarly, faculty data for a course would be maintained in a separate table. shows some sample data of Student Database.

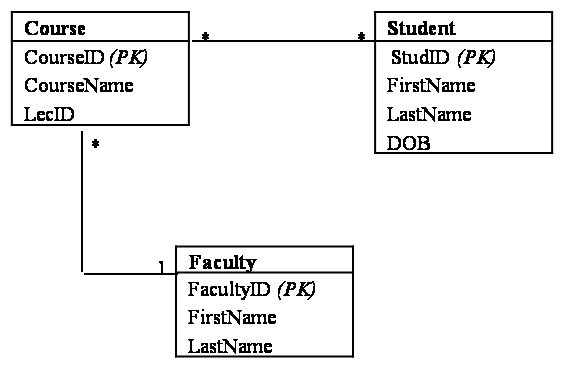


Figure : Database Diagram for Student Database

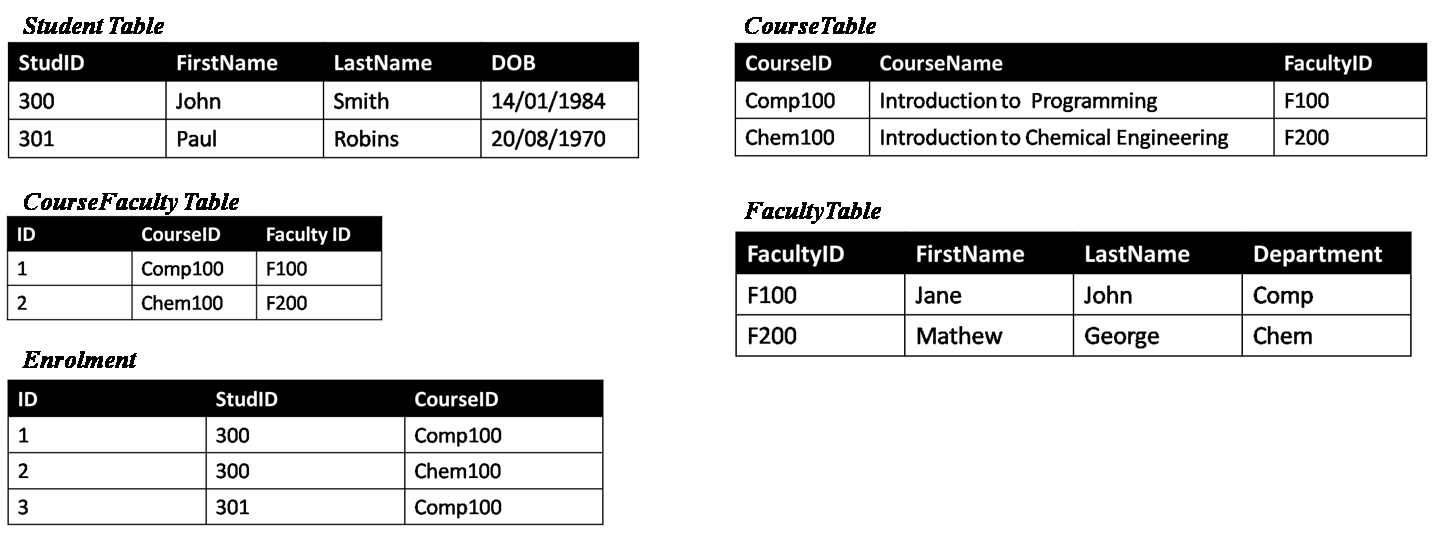


Figure : Sample data in Student database.

The aforementioned data models are explained in the following sections.

## Key Value Data Model

In simple terms, the key value data model represents data as a key-value tuple consisting of a key, value and a timestamp. The value is the actual data that has to be saved and it is associated with the key, which is used to retrieve the value from the database. The value is commonly a string data type. This is similar to the way data is stored in a map. A timestamps is a 64-bit integer and it records the time at which the value was inserted or updated in any way. Most cloud databases adopt the key value data model.

Generally, key value data model on cloud adopts the column-oriented approach .The building blocks of column-oriented key value databases are columns, SuperColumns, ColumnFamily, SuperColumnFamily, KeySpace.

* *Columns*: A column is the basic unit of a table and it contains data represented as a key, value and a timestamp tuple. The JSON notation for a column in the Student database is given in Figure 4.

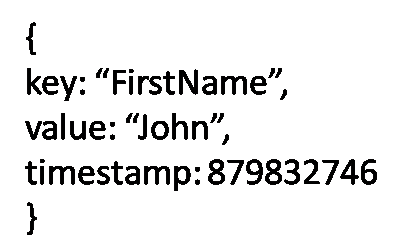


Figure : JSON Column representation

A column in a key value databases is equivalent to a cell in an RDB. A few columns in a key value database are shown in , where “John”, “Smith” etc. would each be a single column with a key and timestamp.

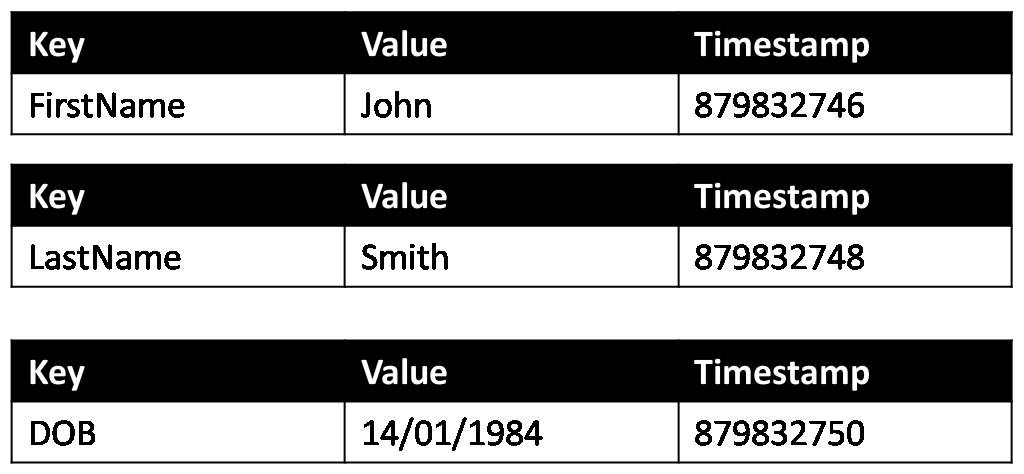


Figure : Column example

* *SuperColumns*: A SuperColumn is a key-value pair, where key or row key, is the unique identifier for each SuperColumn. A SuperColumn can be understood as a row in a table in RDBs. Row keys are just arbitrary string values and can be provided by the users in most key value databases. The value of the key-value pair is a map of columns. Hence, the values in a SuperColumn are actually key-value pairs. In the example, the SuperColumn would be a single row containing the columns of a single student (), and shows how a SuperColumn is a map. The JSON representation of a SuperColumn is given in .

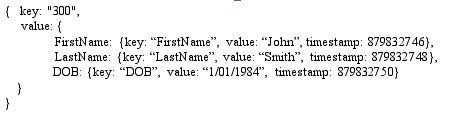


Figure : JSON SuperColumn representation example

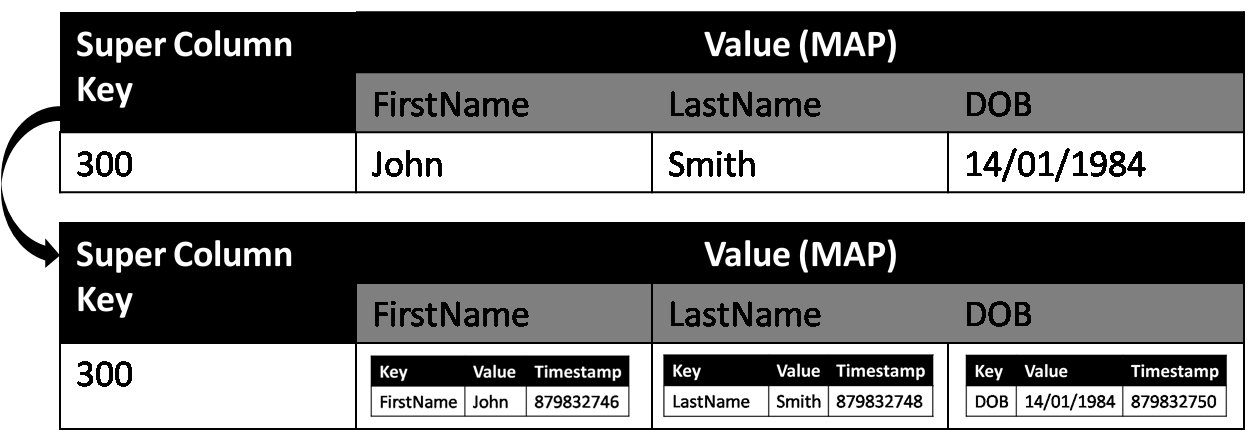


Figure : A SuperColumn example

Unlike columns, SuperColumns do not have timestamps for its key-value pairs.

* *ColumnFamily*: Columns can be grouped together into a ColumnFamily (), which is a structure with several rows containing key-value pairs. The key is the row key and the value is a map of column names. This map of columns contains key value pairs, of column keys and columns. A JSON ColumnFamily representation for the student example is given in .

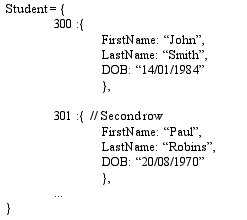


Figure : A JSON ColumnFamily representation example

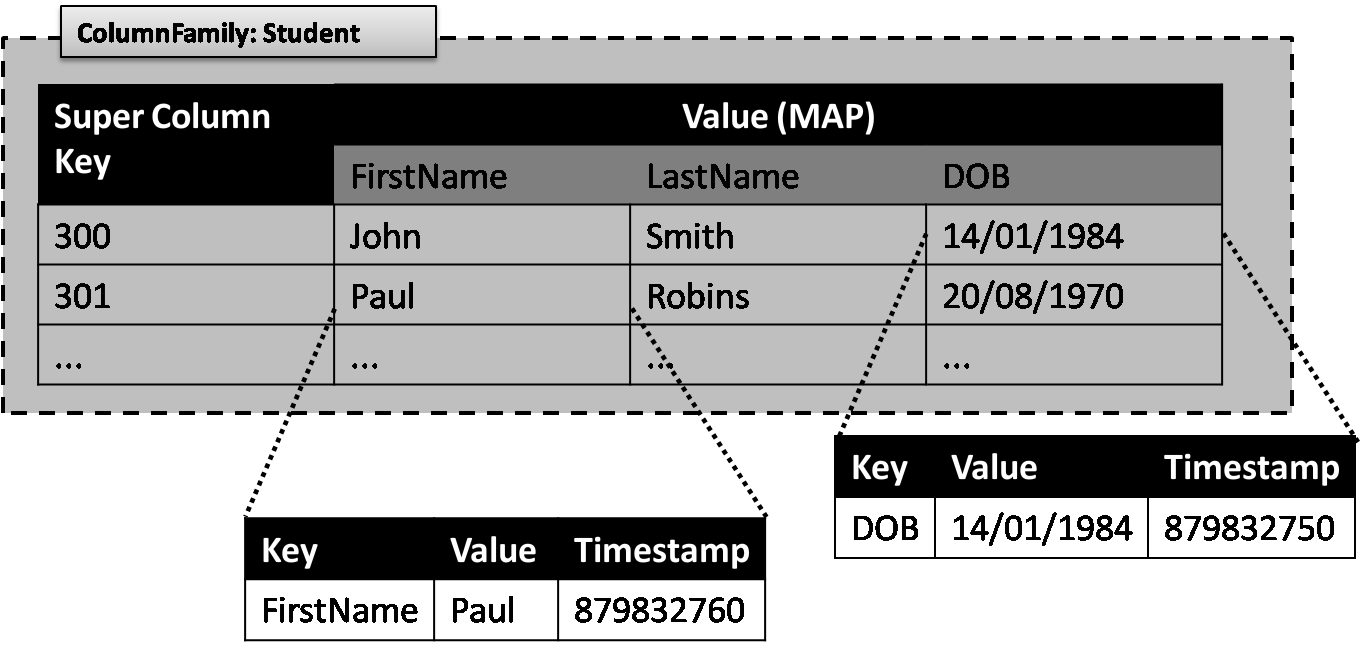


Figure : A ColumnFamily example

A ColumnFamily is analogous to a table in an RDB. Key value databases are schema-less and thus no structure is enforced on the tables. Hence, the rows in the ColumnFamily do not have any constraints with the number of columns it can have (Sarkissian, 2009). This means that it is valid even if one of the rows in the above example has more columns, for example, 301 could have an additional column of “Address” which no other rows have values for. Or one row can have 100 columns and another row could have only 2.

* *SuperColumnFamily*- A SuperColumnFamily contains rows where each row of a SuperColumnFamily contains a map of SuperColumns. In this map, the key is the SuperColumn key and the value is the SuperColumn itself. In the student example, a row in a SuperColumnFamily to show the faculties who teach a course would contain the SuperColumns for the course and the faculties who teach the course (). The JSON representation for SuperColumnFamily is given in .

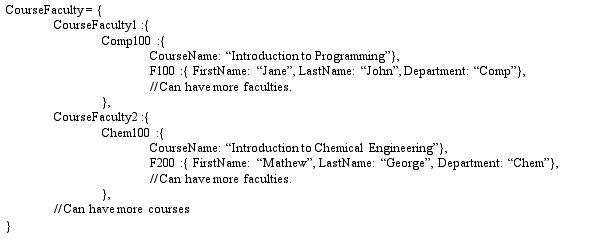


Figure : A JSON SuperColumnFamily representation example

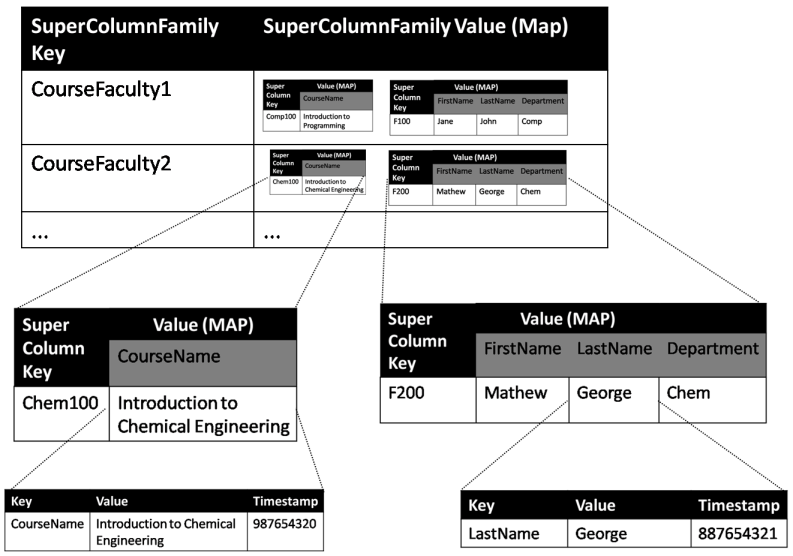


Figure : A SuperColumnFamily example

shows how two SuperColumns are stored in one single row. Here, the SuperColumns containing data about the lecturer “Mathew” who takes the course “Introduction to Chemical Engineering” is given in a single row identified by the key “CourseFaculty1”. If a course has more than one lecturer, then the SuperColumnFamily row would contain another SuperColumn of that lecturer too.

* *KeySpace*: All the ColumnFamilies are grouped into a KeySpace, which is in general named by the user. A KeySpace can have multiple ColumnFamilies but no relationship between such ColumnFamilies is maintained. A KeySpace can be considered similar to a database in traditional relational databases, minus any relationships.

The way in which key value databases are implemented on the cloud, to match the scalability, data availability and other requirements on cloud, is studied while exploring the architecture of the various cloud DBMSs in later chapters.

## Document Data model

In a document data model, data is stored in a format, like XML, JSON, etc. that the database is designed to understand. As long as the database can handle the document type, users can store any type of data. At the core, document databases are key value databases, where documents are stored with a key and this key is used to retrieve the document (Rahien, 2010).

A JSON notation for storing a document, under the key “studdoc1”, in a document database is given in .

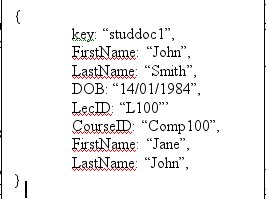


Figure : A JSON representation for Document Database

Like key values databases, document databases require no schema for databases, thus allowing users to store complex or arbitrary data easily in any structure, like tree structure or dictionaries. This is unlike key value databases where data is stored as blobs or just strings. Unlike traditional relational databases, document databases do not support any relationships or data integrity. All the documents in a document are treated as independent data stores and integrity of references to other documents is not maintained.

CouchDB, Riak, Raven, MongoDB etc. are some of the document databases that are gaining popularity. Most document databases support querying as long as the database is designed to understand the document type. Views are supported by most document databases and different approaches to update the views exist (Rahien, 2010). For example, CouchDB updates the views when a query is run on the view but Raven will update the view whenever the related documents are changed. These approaches affect the querying time, i.e., updating the view while querying will delay the query results to be prepared and updating the view as and when the documents used in the view changes will reduce query runtime.

## Relational Data Model

Just like traditional relational model, relational model on the cloud also supports relations or tables with rows and columns to store structured data and adheres to a schema that is specified while database design. The data is given the structure through the process of normalisation where tables are normalised atleast to First Normal Form (1-NF). This ensures that data is organised and not redundant. Normalisation causes databases to have more smaller and structured tables by removing duplicate data from large and badly organised tables and by imposing constraints on the data. Throughout the chapters normalization refers to making databases atleast in 1-NF. and show how the Student database is designed in a relational model, where data is not duplicated and separate tables like “CourseFaculty” hold the relationship between the tables “Faculty” and “Course”.

Data can be queried from the relational databases using a Query language, like SQL. While querying, data is retrieved from the tables and if the query involves multiple tables, then the result sets are joined and the final resultset is produced.

Relational databases are implemented through the RDBMS that help in the creation, maintenance of these databases. RDBMSs give control to database administrators and database users to manipulate their databases.

Fundamentally, the key value data model is different from the relational model in many ways. As seen above, while relational data model aims at giving data a structure, providing data integrity and referential integrity, key value databases just store data as blobs or string values and generally do not maintain many relationships between data. The key-value association and the ColumnFamily grouping etc can be considered as the minimum relationship that is maintained within cloud key value databases

The key value databases, document databases and other databases that support non-relational data models, on the cloud are loosely termed as NoSQL (Not only SQL) databases. NoSQL DBMSs are considered the next generation cloud DBMSs that aim to provide non-relational distributed DBMSs with open-source content and development for the cloud (NOSQL, n.d.). A comparison of the NoSQL DBMSs and the traditional and popular RDBMSs is covered later.

## Summary

This chapter studied the key value and relational data models and also briefly touched on Document data model. This chapter showed how databases adopting these different data models handle and store data. The implementation of these data models on the cloud, and how databases solve scalability and data availability and other cloud related issues by implementing the different data models, would be studied when the different cloud DBMSs are introduced in the remaining chapters

Chapter 4

# Bigtable

## Introduction

To understand the data storage mechanisms deployed in cloud computing, it is vital to understand the architecture and storage mechanisms in Bigtable, a DBMS developed by Google in 2004, as Bigtable lays the foundation for many cloud DBMSs. Many cloud DBMSs have adopted key features from Bigtable. Bigtable is, thus, one of the pioneers of new kinds of DBMSs for the cloud.

Google has many products that are used by millions of users and require large amounts of structured data to be stored. Data storage for such large amounts of data required good scalability, high availability of stored data, consistency of data etc (Chang et al., 2006a). Google thus built Bigtable as a distributed storage system that could store petabytes of structured data. Bigtable was also designed to be scalable over a large cluster of servers and nodes. Many of the Google services and products like Google Earth, YouTube etc. use Bigtable to store their large amounts of data. Even though these products have varying demands in terms of storage size, latency etc., Bigtable has been able to support such products with good flexibility to suit their varying demands (Chang et al., 2006a).

Bigtable was mainly an in-house development to support the data storage needs of the various Google services like Google Earth, Google Search engine, Google App Engine, Orkut, YouTube etc. But recently Google have allowed outside developers and users to access Bigtable through their cloud service platform Google AppEngine (Hakala, 2009).

## Bigtable Architecture

Bigtable is built on the Google File system (GFS), a distributed file system developed by Google for its own use to run on its computing cluster. Bigtable uses GFS to store its data and log files and also uses various other Google programs like Chubby lock service, which is explained later.

Bigtable relies on the key value data model and adopts the column oriented approach. This means that Bigtable stores data in tables that have columns, ColumnFamily, rows, row keys etc. According to Chang et al. (2006a), “Bigtable is a sparse, distributed, persistent multi-dimensional sorted map”. Bigtable is a map as the ColumnFamilies in Bigtable are a collection of key value tuples where values are columns. These columns are again key value pairs, where value is the actual data. From the student example, as seen before, a ColumnFamily in Bigtable would be as shown in . Bigtable is distributed as it is designed to scale to petabytes of structured data over thousands of servers and nodes within the Google network and data centres. Being persistent means that the data stored in Bigtable would continue to exist even after the application that last used it stops running. As seen before, every column has a timestamp recording its last updated or inserted time. Having the timestamps adds a new dimension of time to the data stored in Bigtable, making Bigtable a multi-dimensional map.

Using the student database example, a column is again the same as in Figure 5. Every piece of data is a column value with a key and timestamp associated with it. The key of a column is called a column key. A column key is named using the ‘family: qualifier’ syntax (Chang et al., 2006a). Here, the ‘family’ refers to the ColumnFamily it belongs to and the ‘qualifier’ is the contents of the column cell For example, in the column key for “John” would be “Student:301” where “Student” is the ColumnFamily key and “301” refers to the column key for John.

Just as in column oriented key value databases, Bigtable also have rows. Every row in Bigtable is a SuperColumn which is a map containing key value tuples of columns (Figure 7), where all rows have row keys, which are just string values. Each table is given a range of row keys it can have dynamically and when a table grows large, it is broken into tablets at these row boundaries specified by the row key range (). Tablets could be approximately 100-200 MB in size and these form the unit of distribution and load balancing, according to Chang et al. (2006a). For example, if the Student table grows larger than 200 MB, then the table is split into tablets of 100-200 MB size.

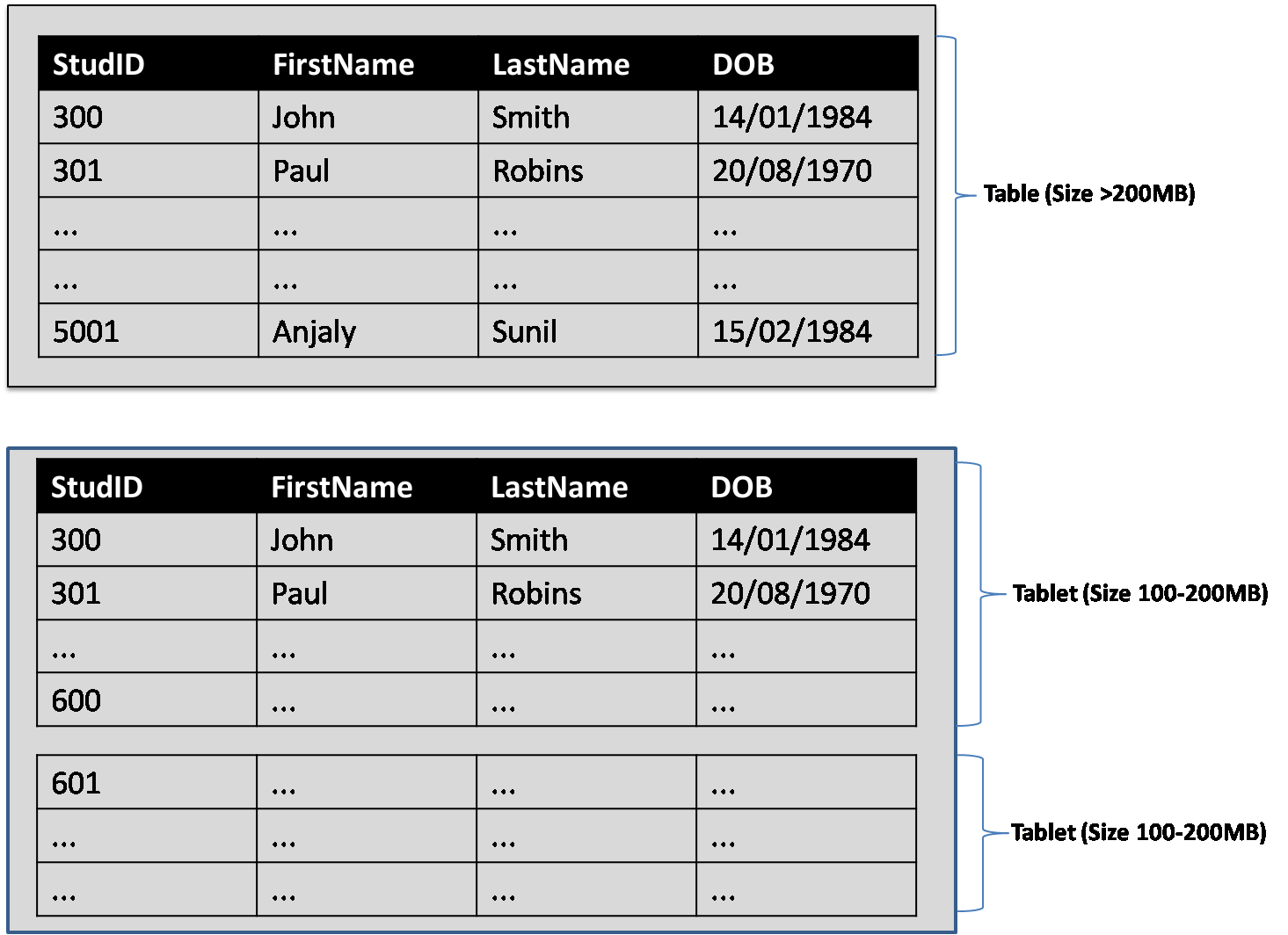


Figure : Splitting a table into tablets

Bigtable also give users the options to give the timestamps themselves, if the users wish to apply logic to provide timestamps suiting their applications. If users do not give the timestamps, Bigtable assigns the real-time in microseconds to the column whenever data is updated or inserted by users. The column cell would hold the multiple versions of the column value. In Bigtable this is possible as the column values are indexed by their timestamps. For example, in , if the column value “John” is updated to “Johns” by a user, then an updated version of that cell is stored in Bigtable. This is shown in .

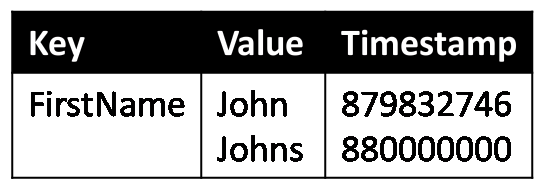


Figure : Bigtable Timestamp example

*SSTable*: As previously mentioned, Bigtable uses GFS to store its files. To store the internal data of Bigtable, the Google SSTable format is used. An SSTable () is a Sorted String Table that store key-value pairs where both keys and values are arbitrary strings. Every SSTable contains blocks of data, where each block could be 64KB in size (Michalis, 2009).An index at the end of the SSTable is used to located the blocks and this index is loaded into the memory when the SSTable is opened (Chang et al., 2006a). The blocks in the SSTable can be compressed and the compression format can be decided by the user themselves.

A tablet is built of several SSTables as seen in . This means that data in the tablets, which are row ranges of a table, are split into 64K blocks of data and these blocks are indexed in SSTable structure.

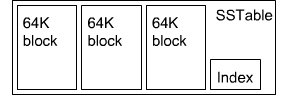
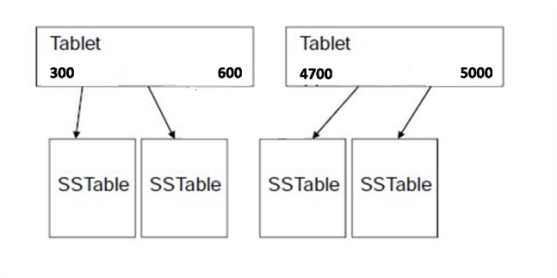


Figure : SSTable (Courtesy: Michalis, 2009)



**...**

Figure : SSTable structure in tablets

*Metadata tablets:* The physical locations of tablets are stored in the metadata tablet called *Meta1* (or METADATA). The *Meta1* tablets store the locations of a set of tablets under the tablet’s row key. This row key identifies the table the tablet belongs to. The location of *Meta1* is stored in *Meta0* (or Root Tablet), and *Meta0* is queried by users to find the location of the *Meta1* that has the required tablet in it. This structure is illustrated in , where a Chubby file holds the location of the Root tablet or *Meta0* tablet and the root tablet holds the location to the other metadata tablets or the *Meta1* tablets. Chubby files are explained later in this section.

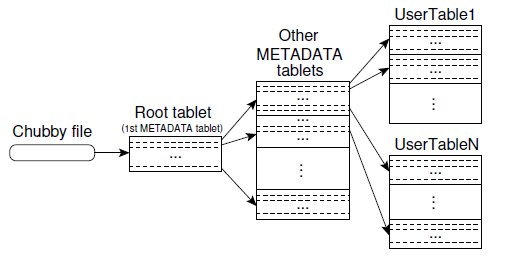


Figure : Metadata tablets (Courtesy: Chang et al., 2006)

Since *Meta0* is what the users query to locate the correct *Meta1*, *Meta0* is located on a server of its own to handle all the queries it gets. The amount of information about the location of *Meta1* is small and also users cache the location of *Meta1* once they know it. This avoids any bottleneck even if too many users query *Meta0* to find out the location of tablets. In some cases, the user library pre-fetches the tablet location of more than one *Meta1* tablets for any possible future use. All these measures reduce the amount of queries *Meta0* has to handle.

*Chubby*: As seen in , a file in the Chubby stores the location of the *Meta0* tablet (Root tablet). Chubby is a distributed lock service that Bigtable uses. According to Chang et al. (2006), the Chubby service consists of five replicas and one of these is elected to be the master. This master replica then serves all the requests from users.

Chubby provides Bigtable with a namespace which has directories and small files. The Chubby directory or file can be used as a lock and this help in making each read or write atomic. Each time a user wishes to read or write to a Chubby file, it starts a session with the Chubby service. As long as the user has the session with the service, it has the lock exclusively to itself and it loses this lock once its session is expired. A session expires when the user is unable to renew its session within the expiration time. Chubby ensures that at least one master is active at any time and also to bootstrap the location of the *Meta0* tablet. According to Chang et al. (2006), when Chubby becomes unavailable for a long period of time, Bigtable itself becomes unavailable. This is mainly because Chubby metadata files hold the location of all the tablets and if Chubby is unavailable due to any reason, Bigtable would not be able to locate the tablets or the SSTables.

Chubby files also store the Bigtable information that specifies the ColumnFamily information for each table, which helps in tablet assignment. For correct tablet assignment, Chubby is used to keep track of the tablet servers.

The tablet servers are assigned their tablets by the master server. Both master server and tablet server are major components of the implementation of Bigtable. The master server detects addition and expiration of tablets on the tablet servers and also performs garbage-collection of files on the GFS (Chang et al., 2006a).

## How Bigtable works

There are three major components that are vital for the implementation of the Bigtable; The master server, the tablet server and the library that is linked to every user (Chang et al., 2006a).The master server is responsible for allocating the tablets to the tablet server and communicates with the tablet servers to find out if the tablet servers are active or not and to delete or reassign tablets accordingly. Each tablet sever manages the set of tablets it is assigned with and responds to the write and read requests from the users. The users do not communicate with the master server and instead send all their requests to the tablet server. This makes the master server have lesser load.

When a master server starts, various steps are carried out to learn about the tablet assignments within the cluster currently. As shown in , (1) the master server does so by taking hold of a unique master lock in Chubby. This would prevent other servers from trying to be the master and ensures that there is only one master server in the cluster. The master server then scans the server directory in Chubby to discover the tablet servers that are live (2). The master then communicates with all the live tablet servers and find out the tablets assigned to them and adds a root tablet if one was not already assigned. It then scans the Root table or *Meta0* tablet and finds out the *Meta1* tablets that are live and available. The master server then scans the *Meta1* or Metadata tablets to know the set of tablets (3). While scanning the *Meta1* tablet, if the master server finds any tablet that is yet to be assigned to a tablet server it makes it eligible for allocation and later allocates it to a tablet server (4).

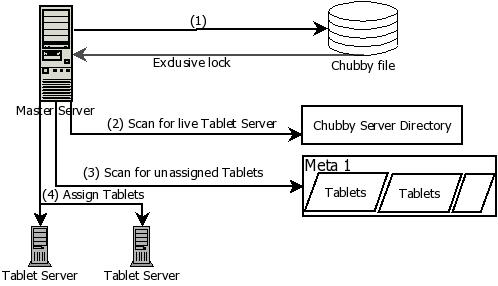


Figure : Master Server initiation steps

Tablet servers are dynamically added to suit the workloads of the cluster. A tablet server manages approximately ten to thousand tablets. The read and write requests to each of the tablets from the users are handled by these tablet servers. Tablet servers also split the tablets when it grows too large, according to the tablet’s row range (). When a tablet server starts, it gets an exclusive lock on a unique file in a Chubby directory. The tablet server records the information for the new tablets in the *Meta1* tablet. Once it is saved into the *Meta1*, the master is notified of the change and the process of assigning it is done by the master server as explained previously.

When a user sends a write operation to a tablet server, the tablet server checks whether the user has the authorisation to receive the required service. This is done by checking the Chubby file that holds the list of authorised writers. The tablet server then commits the operation into the log file that is common to all the tablets on the same machine. After the write has been committed, the contents of the write operation are written into the memtable. Memtable is a sorted buffer in the memory that is used to store the recently committed updates. This is illustrated in .

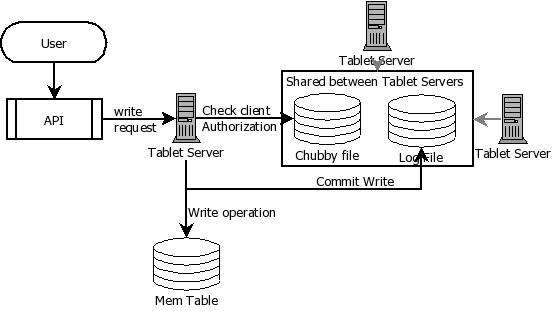


Figure : Write request in Bigtable

When a user sends a read request to a tablet server, the tablet server checks for the authorisation of the request. The read operation is then executed on the SSTables and the memtable ().

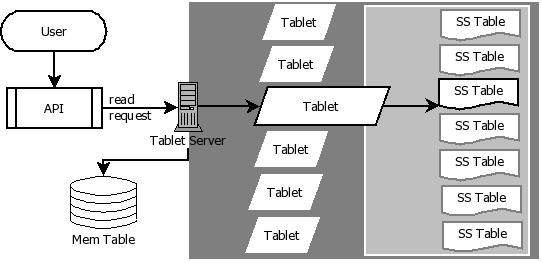


Figure : Read request in Bigtable

When the memtable reaches a threshold, the memtable is frozen and a new memtable is created. This frozen memtable is then converted to an SSTable and saved on the GFS. Bigtable exercises compactions on the memory used by the tablet servers. Minor compactions shrink the memory usage of the tablet server. It also reduces the amount of data that has to be read from the log during the recovery process when the tablet server dies. Another type of compaction is the merging compaction where the contents of a few SSTables are read along with the memtable and this is written into a new SSTable. When all the SSTables are merged into a single SSTable due to such a merging compaction, it is known as the major compaction. Bigtable often scans its SSTables and performs major compactions on them (Chang et al., 2006a).

A tablet server can die, or stop serving tablets if it loses the exclusive lock. It could lose the lock perhaps due to network partition, latency in requests etc. The master server detects when a tablet server is no longer serving its tablets. In such cases, the master server reassigns the tablets on that tablet server to other tablet servers. It finds out if a tablet server is serving its tablets by periodically asking each tablet server for its status. Tablet servers stop serving their tablets only when they lose the lock to the Chubby file. The master then acquires the lock on the Chubby file to find out if the Chubby service is still live. If it acquires the lock, the master knows that the tablet server is not serving the tablets and allocates its tablets to other live tablet servers. According to Hitchcock (2005), each tablet server takes one tablet off the offline server.

Since there is a large amount of redundant data, Bigtable makes use of compression and use variations of BMDiff and Zippy. More details of these techniques can be found in Burton (2008).

Since large amounts of replicated data exist, Bigtable allows garbage collection, where users can specify how many of the last versions have to be kept or whether only new versions are to be maintained.

## Summary

This chapter introduced Bigtable and its architecture. Bigtable has paved way for non-traditional database models to suit the demands of the world of cloud computing. It has shown how Bigtable can support vast amounts of structured data, yet scale to suit the demands of the services using it without compromising on essential features like locking, indexing, structuring and versioning data etc. This chapter shows how efficient and cloud-suited Bigtable is and why understanding Bigtable is essential to comprehend some of the other existing cloud database systems as well as to understand the future progress in this field

Chapter 5

# Amazon Dynamo

## Introduction

Amazon has a wide range of services in its e-commerce platform that it offers to millions of users worldwide. The scalability and reliability of these services rely on the efficiency with which the application state is stored. Amazon services are run across tens and thousands of servers spanning several data centres worldwide, hence the probability of a failure of servers or an outage was large. The chances of a part of the widespread network failing was a large risk to data availability and data loss. Dynamo is a storage technology developed by Amazon, with a goal of providing fault tolerance and data availability at all times to these services so that service to the users would not be disrupted even in the case of an outage within a single node or a whole data centre (DeCandia et al., 2007).

According to Iskold (2007), for large e-commerce platforms adding more web servers to meet their user demands would not be advantageous unless the database at the backend was scalable too. Some of the Amazon services did not benefit from the traditional relational databases (Iskold, 2007).Traditional RDBs when used to store product and user information for such sites prove to be a bottleneck as these databases do not support redundancy or parallelism. These services needed key access to the stored data as they stored and retrieved data using primary keys. According to DeCandia et al. (2007), these services also had no complex queries or data management to feature in their use. Such systems were better off with key value databases, minimising unwanted functionalities. Dynamo may not be the perfect solution, but it is more appropriate when data availability and fault tolerance are vital for a business that uses several data centres (DeCandia et al., 2007).

## Dynamo Architecture

Dynamo is a distributed storage system based on the key value data model where data is stored as blobs (Binary Large OBjects). The blob objects that the database store is paired with a key, which is also used to look up the objects. The objects are stored without any semantic or structural information, making Dynamo schema-less (Henry, 2008). This allows the database to store any kind of objects, like images or files, and querying is limited to just identifying the correct object with the key unlike relational databases where querying is done to retrieve more information. For example all the data about a student “John” could be saved as a blob, with a unique key given to the student blob object or data of many students could be saved as a blob object in Dynamo.

In Dynamo, the physical nodes that store the copies of the object are organised as a ring (). The data is distributed among these nodes, which spans different data centres, and data is made redundant to cope with outages and breakdowns of nodes or data centres. This means that each object would be stored multiple times in the Amazon network (Iskold, 2007).

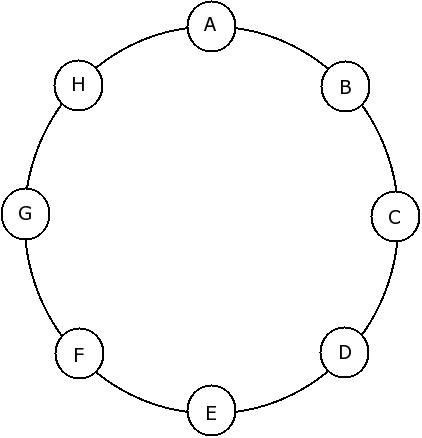


Figure : A ring of nodes in Dynamo

Dynamo uses the Peer-to-Peer (P2P) technology and Gossip, the P2P communication protocol, is also implemented. This protocol involves many complex mathematical models and help in significantly reducing the time taken to propagate requests from node to node (Raja, 2010). The Gossip-based protocol broadcasts any membership changes to all the nodes and allows nodes to reconcile such changes with its peer nodes.

Dynamo provides a simple interface with two operations: get and put. Get () retrieves the most updated value associated with a key and Put () updates a value (Henry, 2008, Iskold, 2007, DeCandia et al., 2007).

Using the P2P key-value store model for storage, Dynamo employs various techniques to address the problems like partitioning data, consistency, replication, failure management, failure recovery, updating strategy etc. Some users note the failure tolerance as the crucial feature of Dynamo while some others note the partitioning algorithm is the key element of the Dynamo architecture. Dynamo enforces a Service-Level-Agreement (SLA) for its services to perform. The SLA and the various techniques that form the architecture of Dynamo are described below:

* *Partitioning Data*: According to DeCandia et al. (2007), Dynamo uses a variant of consistent hashing to partition the data over nodes. Each node is assigned a random value that shows its position in the ring. Objects are assigned to the nodes after the object’s key is hashed. The ring of nodes is traversed clockwise and the node with its value closest to the hashed key is assigned the object (). Higher performing nodes are assigned multiple points in the ring, making them virtual nodes that can be assigned more objects. Virtual nodes are assigned workloads from failed nodes or overloaded nodes, thus performing load-balancing in Dynamo. Scalability issues are also solved, as when new nodes are added, workloads form other nodes can be assigned to the new nodes.

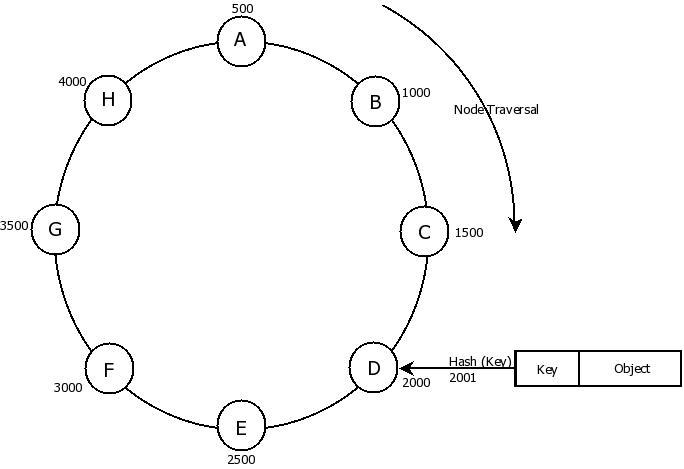


Figure : Consistent hashing in Dynamo

* *Replicating Data*: Every object is replicated across several nodes using a replication strategy of replicating an object to N hosts. Each object key is assigned to a coordinator node and this node replicates the assigned objects (DeCandia et al., 2007). Dynamo also maintains a preference list that contains the coordinator nodes that are responsible for storing a key (). This is the routing information of the node and every node in the ring knows which nodes are responsible for a key.

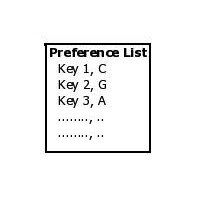


Figure : A preference list example

* *Consistency Protocol*: Since there are many replicas of the object existing in the system, it is vital that the replicas are consistent. Dynamo uses a consistent protocol with two values, R and W, which can be configured to suit the application’s requirements. R denotes the number of nodes that has to participate in a successful read operation and W denotes the minimum number of nodes that has to participate in a successful write operation (DeCandia et al., 2007). A high W means more nodes participate giving more durable writes. A high R means that more nodes are read returning more replicas of the object while low R would mean that the first few reads would be used. When W is low, a write operation is accepted as long as a single node is active on the ring to store the replica. According to the availability required by the user, the W and R values can be set by the user (Anderson, 2010).
* *Eventual consistency*: Dynamo chooses a variant of eventual consistency to maintain consistency among the various replicas of the data stored within it (DeCandia et al., 2007, Henry, 2008). Eventual consistency is where every replica of the data will see the update eventually after a time delay, which could translate to latency in Amazon’s wide network. Eventual consistency in Dynamo, as shown in , means that every replica agrees to the most recent value after a certain point in time and allows updates to be propagated to all the replicas asynchronously (Henry, 2008). This requires that the replicas update the values in an order.

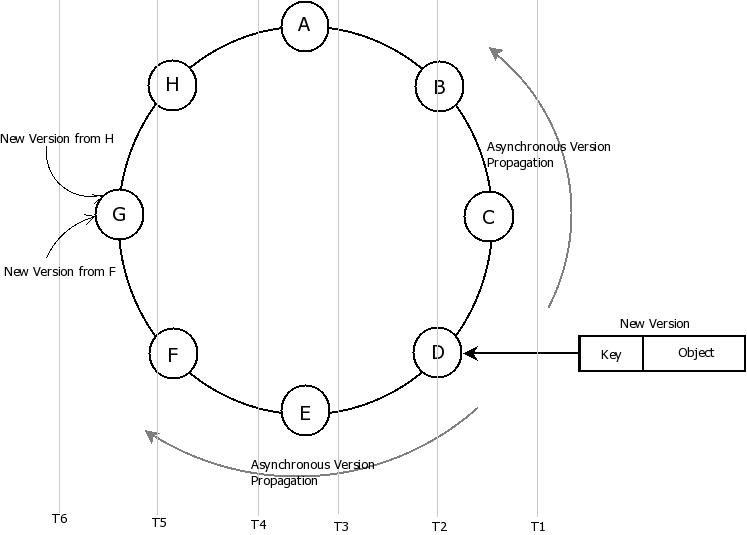


Figure : Eventual Consistency in Dynamo

Dynamo uses Vector clocks for such versioning purposes. Vector clocks contain a list of the nodes and its counters. Each replica maintains a list of the updates it has received as its counters. These counters are used to verify the version of the replicas. Each time a replica receives an update, it increases the counter in its vector clock, as seen in (Henry, 2008). In this way, if one replica has a counter less than the counter of another replica on a different node, then it can be understood that the first replica is an ancestor of the second replica. Hence, if a replica receives simultaneous updates from other nodes, it can verify their counters and perform the right updates. If there is a conflict in the versions of the replicas, the versions are sent to the users and the user is expected to resolve the conflict and update the correct version.

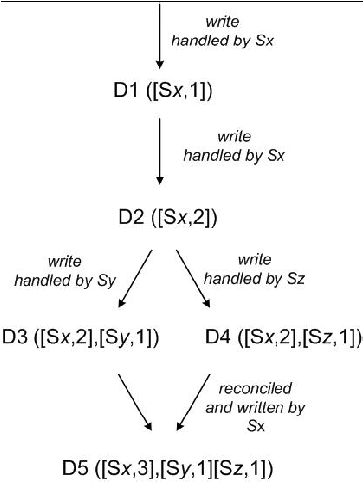


Figure : Vector Clock (Courtesy: DeCandia et al., (2007))

* *Hinted Handoff*: This is the mechanism that Dynamo uses to ensure that the number of reads and writes to the nodes are successful and never fail despite node failures or outages. According to DeCandia et al. (2007), “Dynamo does not enforce a strict membership and instead uses a ‘sloppy quorum’”. This means that the first N number of healthy and active nodes, from the preference list, is used to perform the read and write operations. Upon node failures replicas are sent to another node in the ring (). The replicas would hold a metadata about its actual owner node. Using this metadata or the hint, the recipient node would keep the replica in a separate data store and scan the ring periodically to detect the status of the failed node. Once the failed node is active the recipient node would send the replica back to the active node.

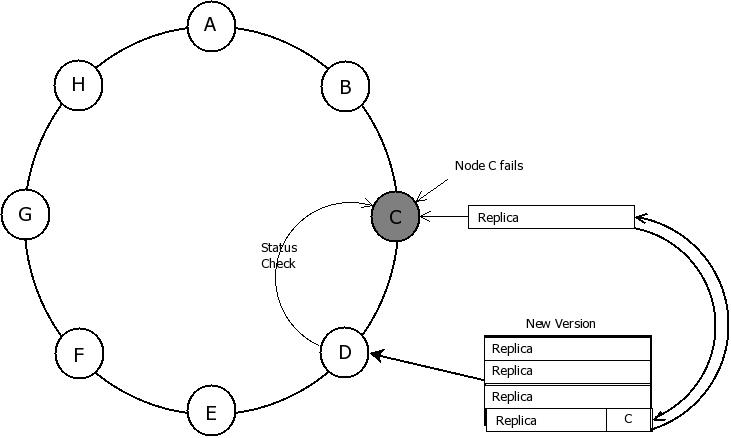


Figure : Hinted Handoff in Dynamo

Dynamo also uses some nodes as seed nodes, which are available and accessible via an external mechanism (DeCandia et al., 2007). Seed nodes are known to all the nodes. Using this approach, a new node addition is made easier to detect by the existing nodes (). Once the new node is synchronised with the seed node all the other nodes would update their membership information upon synchronising with the seed node.

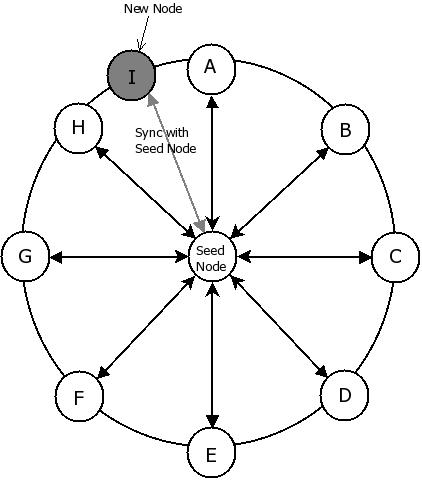


Figure : Seed node in Dynamo

Dynamo aims at achieving the Availability and Partitioning of data of the CAP theorem (Williams, 2010). According to Williams (2010), the CAP theorem proposed by Brewer states that a distributed system can have only two of the following properties: Consistency, Availability, and Partition tolerance.

Dynamo uses the push-on-change model, where the propagation of the updates on data is replicated through the nodes using simple operations and the read operations are kept simple.

## How Dynamo works

According to DeCandia et al. (2007), Dynamo uses Java to implement all the fault tolerance, membership, and failure detection services on its nodes. It uses different storage techniques like Berkeley Database (BDB), MySQL and an in-memory buffer to store persistent data. This storage is pluggable, which means that the service that deploys Dynamo can choose what kind of data store to use for the persistent storage. For example, some services might choose BDB or MySQL while some others might choose the in-memory buffer. This depends on the estimated size of the local persistence of the user services that use Dynamo (DeCandia et al., 2007).

Every user using Dynamo has to engage in a Service-Level-Agreement (SLA) that binds the users to a contract where the user and service agree on various characteristics. This also includes the expected response time of the user and determines the expected latency from the user side. The SLA ensures that the response time for the requests would still provide Amazon with 99.9% performance level and this is analysed in terms of peak load hours too (DeCandia et al., 2007).

Using the Gossip-protocol, a node propagates its partitioning and membership information, i.e., preference list, to random peers periodically, thus keeping the preference list updated. Also, nodes regularly synchronise with the seed nodes to update preference lists.

According to DeCandia et al. (2007), when a user sends any requests, along with the key, to a node, the node scans its preference lists and forwards the request to the coordinator node of the key (). A write request is a Put (key, context) request, where the context is the version details of the object. The coordinator node generates the vector clock for the new version of object. The node then sends the new version and the new vector clock to the reachable nodes in the preference list.

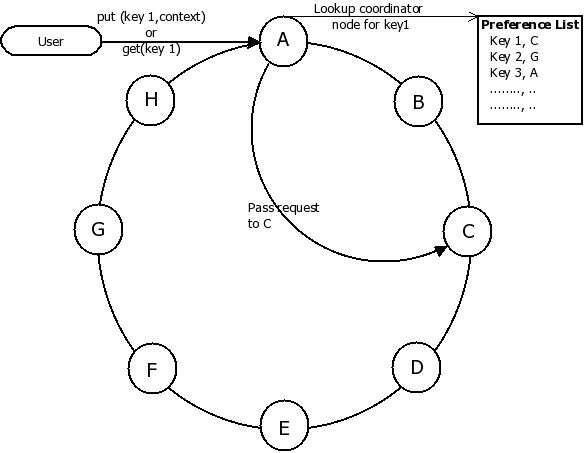


Figure : Handling user requests in Dynamo

Similarly, when a user sends a Get (key) request, the coordinator node requests data from nodes in the preference list that is responsible for the given key. The coordinator node waits for the minimum responses from nodes (R nodes) and performs version-checking of the received replicas of the object. The correct version of the object is sent to the user.

After the read response is returned to the user, the coordinator node performs a read repair. Any stale versions that were generated from any nodes during the process would be updated with the latest version of the data. This ensures that all the replicas have the latest data. If the versions are conflicting, the conflicting versions are sent to the user. The user then resolves the conflict and sends a Put () request with the correct information and hence a new version would be generated (Sharma, 2009). According to DeCandia et al. (2007), Dynamo uses special kind of tree called Merkle tree for replica synchronisation on nodes. More about Merkle trees can be found in Merkle (1987).

## Summary

This chapter showed the architecture and the techniques Dynamo uses to store data efficiently and explains how vector clocks, eventual consistency are implemented to maintain data integrity and to make data available even during failure of nodes.

Chapter 6

# Cassandra

## Introduction

Cassandra was initially developed by Facebook, a popular social networking site. Facebook has a large user-base with many interconnected data that is active each time a user is online on Facebook. For example, when a Facebook user comes online, all the updates about the user’s friends list has to be retrieved. Cassandra was open sourced by Facebook and later its development was undertaken by Apache (Gunda, 2010).

## Cassandra Architecture

Cassandra is based on the key value data model and stores data in tables that have columns, ColumnFamily, rows, row keys etc. Dynamo adopts the column oriented storage in Bigtable and adopts Dynamo’s partitioning methods, replication strategy and consistent hashing. As a result of this close similarity to Bigtable and Dynamo. Many details important to Cassandra, like Partitioning data, Eventual Consistency, Replication of data, have been explained in Chapter 5 and the data model is explained in Chapter 3.

Like Dynamo, Cassandra has nodes connected to other nodes forming a cluster (). The nodes in a cluster communicate with each other using the Gossip communication protocol, as explained in Section . Nodes thus update their routing information about other nodes periodically too. This also allows nodes to perform load balancing operations, i.e., some workload is given to other nodes, when a node fails or has a high workload. This is similar to in Dynamo, where replicas from a failed node are sent to other live nodes.

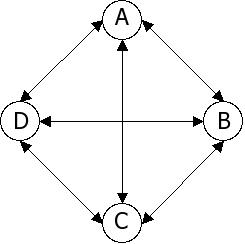


Figure : A Cluster of nodes in Cassandra

Instead of seed nodes like in Dynamo, Cassandra requires booting of the Cassandra process on the cluster when a new node is added to the cluster. This simple process makes Cassandra scalable as any new node addition is easy.

Just like in Bigtable, Cassandra has ColumnFamilies that contain rows of key-value pairs, where values are a map of columns. Key of the key-value pair is the column key belonging to the column. Columns are the actual data stored as a key-value pair (). Cassandra also supports SuperColumns which are key-value pairs. In this pair, the key is the row key and the values are the columns. As seen in Section , this means SuperColumns and ColumnFamilies actual contain maps of key-value pairs. For example in Cassandra, a SuperColumn for student “John” would look like and a ColumnFamily for students is similar to . Such a nesting of columns or data connect the large interconnected data and keeps the read operations relatively simple. Cassandra has simple read queries as data is accesses using keys. For example, to access the SuperColumn for “John” the row key “300” is given by the user. Cassandra focuses more on the write functions (Eure, 2009).

Whenever a write or update function is invoked by a user, Cassandra implements a replication strategy to replicate the data on all nodes in a cluster. This strategy determines the distance from the node that gets the requests to other nodes in the cluster. The distance is broken into three buckets, according to whether the current node is in the same cluster or if the other node is in the same data centre or thirdly if the other node is in a different data centre (Perham, 2010a). Like Dynamo, replication of data is crucial to provide high data availability even during failures of nodes.

Cassandra adopts Dynamo’s eventual consistency. As explained in Section , eventual consistency means that nodes receive the new replicas or version asynchronously, at a later point of time. In Cassandra, users can decide the level of consistency of the data they would receive. Cassandra allows this by giving two options; single read and the quorum read. The single read returns the first data that a node receives for another node as a response to user’s read request (). The quorum read makes the node that receives a read request to collect all the replicas of data from the nodes in the cluster and return the most updated data to the user (). This is similar to setting the R value in Dynamo.

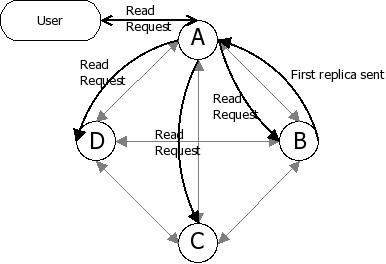


Figure : Single read in Cassandra

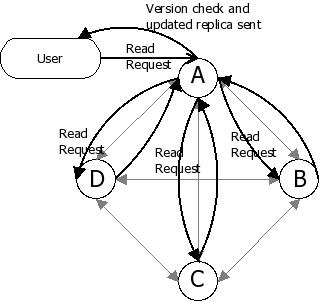


Figure : Quorum read in Cassandra

Cassandra used timestamps like Bigtable to record the version of the data instead of vector clocks until version 0.7. But from version 0.7, Cassandra uses vector clocks like Dynamo, to record the version of data. While checking the consistency of replicas, nodes refer to the vector clocks to find out whether the replica is the latest one or not.

Cassandra also adopts the read-repair strategy of Dynamo to remove stale replicas on nodes. This strategy, as explained in Section , lets nodes that perform read operations to do version checks on the received replicas and to update nodes in the cluster with the latest versions of replica.

Cassandra uses the push-on-change model, like Dynamo, where the changes in the data are propagated using simple queries within the cluster. This is essential as there exists a vast amount of replicated data. According to Williams (2010), Cassandra has chosen Availability and Partition tolerance of the CAP theorem, whilst also providing consistency of data to the users. Like Dynamo, high availability is a property of Cassandra as failures of nodes do not hinder the availability of data and tolerance to partitions in Cassandra is its ability to access data spread on the nodes on different clusters.

## How Cassandra works

As previously mentioned, Cassandra was developed for satisfying the needs of large web applications, where scalability and response time to user requests are critical. As a result, Cassandra is better utilised when it is run on multiple machines in a cluster (Perham, 2010a). Cassandra is a single Java process run on each node in the cluster (Williams, 2010).

When users wish to write data into Cassandra tables, they send a write request to a random node in the cluster (). This node acts as the proxy and writes the data to the whole cluster, thus efficiently replicating the data. Like in Bigtable, a user can set the number of nodes that should have the replicated data copied on it. The replicas are saved on to nodes in the same data centre and on other nodes in the other data centres. These nodes would act as proxies when they receive requests from users (Perham, 2010a). This way even if nodes fail, data is recoverable from other nodes.

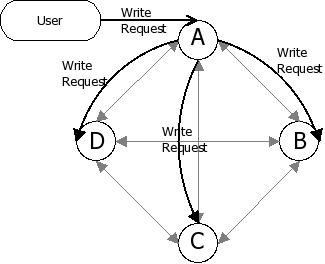


Figure : Write request in Cassandra

Similarly, when a user makes a read request, to a node, it acts as a proxy node and forwards the request to all the other nodes in the cluster. The level of consistency is specified by the user, i.e., single or quorum read. Nodes return data to the user after checking the versions of the replicas and send the latest replica to the user and perform read repairs on other nodes in the cluster (Perham, 2010b).

Cassandra provides APIs to its users for performing data operations like insert, update, delete, retrieve data etc. The Thrift API can support many programming languages, making it easy for most applications to store data in Cassandra (Gunda, 2010).

## Summary

This chapter showed how Cassandra is very similar to Dynamo in architecture and various features. This chapter also showed how Cassandra stores data inspired from the data model used in Bigtable. Cassandra takes the good features from both Dynamo and Bigtable and produce a DBMS that is scalable with high data availability and good fault tolerance. Case studies also show how popular websites use Cassandra.

Chapter 7

# HBase

## Introduction

HBase was initially developed by Powerset and is now developed by Apache. HBase is a distributed database that is an open source subproject of the Hadoop software framework. HBase was primarily designed to provide a Bigtable-like storage mechanism for the Hadoop Distributed computing Environment and it is designed and built on top of the Hadoop Distributed File System (HDFS) (Apache, 2010a).

## HBase Architecture

As mentioned above, HBase is built on HDFS, quite similar to the way Google’s Bigtable is built on the GFS (Apache, 2010a). Like Bigtable, HBase uses the column-oriented key value data model. In HBase, data is stored in rows and columns, where columns are grouped into ColumnFamilies with ColumnFamily names and column keys.

According to Apache (2010a), tables in HBase are a collection of rows that are identifiable by their row keys, like ColumnFamilies in Bigtable in . ColumnFamilies are designed and maintained by the administrators. A column in HBase is like the column in Bigtable and has a timestamp to perform version checking ().

Physically, these tables are stored based on their ColumnFamilies (Apache, 2010a). Columns with null values are not physically given any disk space similar to Bigtable. The tables are broken into regions, based on the range of the row keys ().

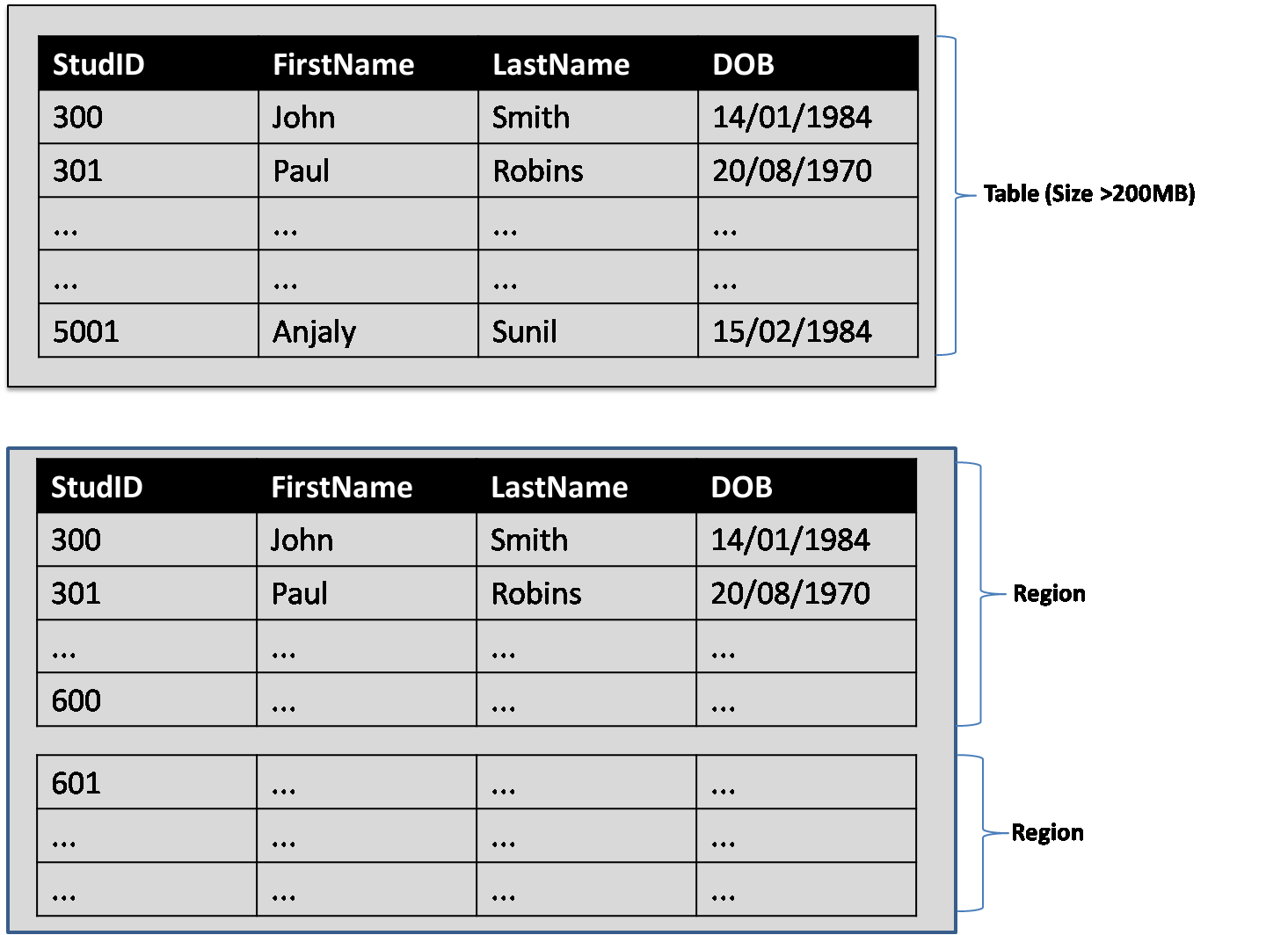


Figure : Regions in HBase

Regions form a table and a table can be identified with the table name and the start key of the key range. Every ColumnFamily within such a region is managed by an HStore (Apache, 2010a). HStores are stores that contain map files that store the internal details of HBase. These map files help in performing lookups for data based on row keys.

HBase architecture, according to Apache (2010a) have three major components; HBaseMaster, HRegionServer and the HBase User. These are discussed below:

* *HBaseMaster*: According to Apache (2010a), when HBase starts, the HBaseMaster assigns regions to the HRegionServers. The first regions that get assigned are the root regions, which hold all the Meta regions. These Meta regions have a root table and store information about user regions, like the start and end region keys of the user regions, status, its HRegionServer (). HBaseMaster assigns the meta regions and user regions to the HRegionServer next (Apache, 2010a). This is shown in .

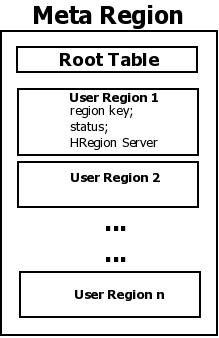


Figure : Meta Region Representation

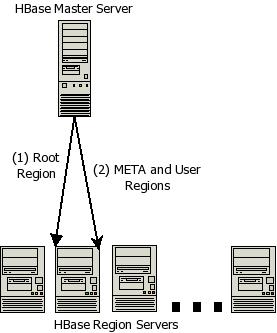


Figure : Assigning Regions in HBase

The HBaseMaster checks the HRegionServer periodically and creates write-ahead-logs in case of any failure of a HRegionServer (Apache, 2010a). The write-ahead-log (WAL) contains logs about all the changes made to the data. In case of a failure, the HBaseMaster will split the failed HRegionServer’s WAL, creating a WAL for every region assigned to the failed HRegionServer. The HBaseMaster then reassigns the regions to other active HRegionServer ().

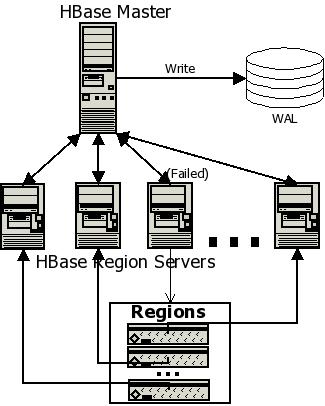


Figure : HRegionServer Failure

The HBaseMaster also performs administrative tasks like adding and removing ColumnFamilies etc. One of the main concerns is that if the HBaseMaster fails, the entire cluster could suffer and shut down. One of the reasons for this is that HBase uses the HBaseMaster as a single point access to all the HRegionServers. This is unlike the Bigtable by Google, where the master has a chubby lock manager that controls and provides access to the other servers (Apache, 2010a).

* *HRegionServer*: While HBaseMaster handles all the mapping and administrative tasks, the HRegionServer handles the entire user read and write requests. It communicates with the HBaseMaster to retrieve the list of regions to satisfy these user requests.

When it receives user requests, it writes the requests to WAL and stores the request to Memcache. Memcache is the in-memory cache and each HStore has a Memcache. Every read request prompts a search within the Memcache to see if the requested data is within the Memcache. Only if it does not exist in the Memcache are the user regions searched for results.

Regions in a HRegionServer are split when map files in a HStore reaches about 256MB (Apache, 2010a). The HBaseMaster is informed of the split or the split information is picked by the HBaseMaster when it periodically scans the Meta regions.

* *HBase Client*: Clients or users are responsible to find the HRegionServer responsible for serving the row ranges users are interested in. Users continue to communicate with the HRegionServers until it locates the correct user region, as seen in . The communication between the user and the HRegionServer is described later.

Users also cache routing information and HRegionServer details for its future use. This saves the users a lot of time and hops in the network. If the cache information is not updated or if the user is unable to successfully locate the regions in its cache, it rescans the Meta region and determines the new location of the row ranges.

HBase also makes use of ZooKeeper. ZooKeeper is a service that provides centralisation of various services that are common in distributed environments. It helps the HBaseMaster in maintaining the configuration information, naming, synchronisation etc (Apache, 2010b). This essentially helps in relieving the master and the region server of some processes.

HBase relies on HDFS and uses many HDFS files to store the data that is handled within HBase and these files are maintained by the HRegionServers. HBase uses two kinds of file types: One for maintaining WAL and the other for data storage for its internal operations (George, 2009). HBase has a root directory that is configurable called “/hbase” (). The root directory has files that are used for logging. It also has subdirectories for each of the HRegionServers. Files for the actual regions are maintained in these sub directories (George, 2009).

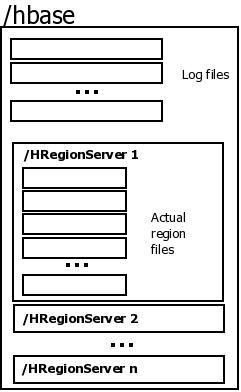


Figure : Root Directory for HBase

HBase stresses more on Consistency of data rather than Availability of CAP theorem, unlike Dynamo and Cassandra (Peschka, 2010). Strong consistency ensures that the distributed data storage will behave like a database stored on a single machine (Henry, 2008).

## How HBase works

For a user to retrieve or update data, it has to locate the appropriate row ranges satisfying the row keys of the data. illustrates the following discussion. To get the row ranges, a user contacts the ZooKeeper to find a particular row key. The user retrieves the HBaseMaster name from the zookeeper. The user then communicates with the HBaseMaster to locate the HRegionSever that has the root region. The user then contacts the HRegionServer and the root region is scanned to locate the correct Meta region ranges. This Meta region contains information of the user region that holds the desired row. The user then has to contact the HRegionServer responsible for that user region to issue the read or write request.

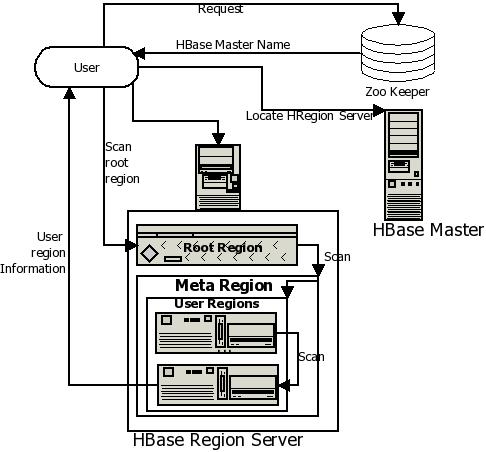
****

Figure : Handling User Requests in HBase

The user issues a put request to the HRegionServer to perform a write operation. When a write request is received by the HRegionServer, the request is written in the write-ahead-log and Memcache (George, 2009). This makes it possible to retrieve the entire log in times of a failure and replay the whole requests to make the servers up to date, right until before the failure, giving HBase a good failure recovery mechanism (George, 2010). The HRegionServer sends the details of the HRegion instance back to the user. The request is written in the Memcache after the Memcache is checked to see if it is full or not. If it is full, then the cache is flushed, as explained before.

## Summary

This chapter looked at the architecture and data model of HBase, a popular cloud DBMS. It was shown how HBase is built around Bigtable and how its data model is very similar to Bigtable’s data model. The architecture of HBase was studied and this helped in understanding how various parts of the architecture interact and communicate with each other to provide HBase users with good consistency while also giving the users the feeling of using a locally stored DBMS.

Chapter 8

# Microsoft SQL Azure

## Introduction

Microsoft SQL Azure Database is a cloud RDBMS, built on the foundations of the Microsoft SQL Server technology and comes both as a part of the Azure platform, the cloud platform offered by Microsoft, as well as a DaaS on the cloud. The main aim of Microsoft SQL Azure is to provide support for SQL and controlled scalability while storing data in a relational model (Campbell et al., 2010).

Microsoft SQL Azure Database is maintained and supported by the Azure platform. The data saved in this database resides on the servers in the Microsoft data centres in different parts of the world.

## SQL Azure Architecture

Data in Microsoft SQL Azure is stored in the form of tables with rows and columns, just as in traditional RDBMSs. Databases are created through a developer portal () that comes as a part of the Azure Platform (SQLAzure, 2011).

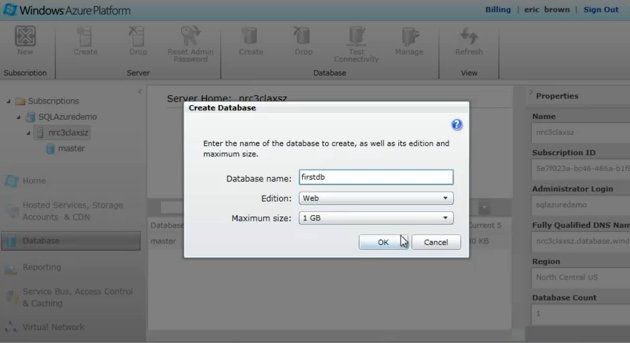


Figure : Database Creation in SQL Azure (Courtesy: SQLAzure, 2011)

The portal provides a database manager for users to manage their databases, run queries, create views etc. ().

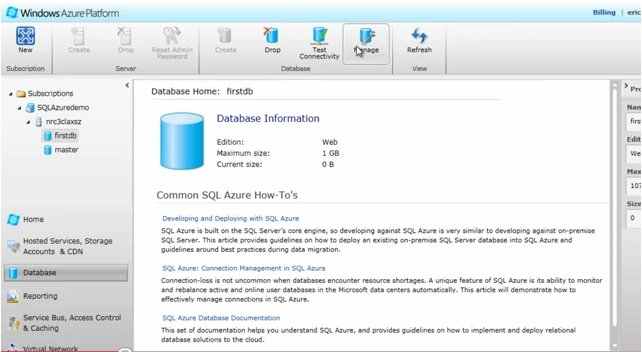


Figure : Database Manager in SQL Azure (Courtesy: SQLAzure, 2011)

When a table is created in SQL Azure (), it is given a default schema called “dbo” and users can specify the properties of the table like columns, primary keys etc.(Microsoft, 2010c). For example, to enter the details of a person or student, a table is created in the database and the columns are specified along with constraint information like primary key, required field etc.

Like in SQL Server, tables can be created using the SQL command CREATE DATABASE too. Data is entered into the rows of the tables either using the interface or by executing SQL INSERT command. SELECT statements can be given to view data and the result is presented in tables of rows and columns. shows how SQL Azure saves data just like traditional RDBMSs. SQL Azure also supports stored procedures and views in databases.

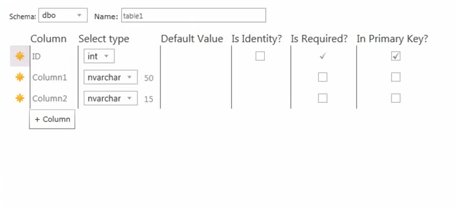


Figure : Table Creation in SQL Azure (Courtesy: SQLAzure, 2011)

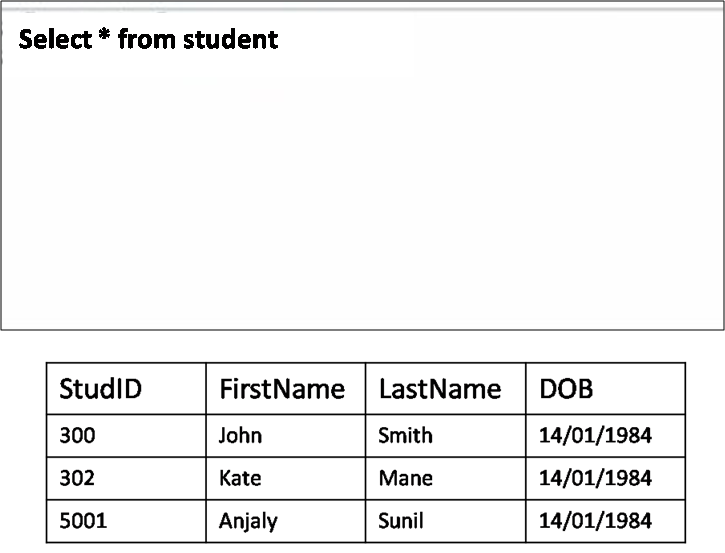


Figure : Select result in SQL Azure

Architecture and the implementation of SQL Azure are similar to the Azure platform model. Understanding the Azure Platform architecture (shown in Figure) prior to learning SQL Azure architecture is helpful. Appendix C describes Azure Platform architecture.

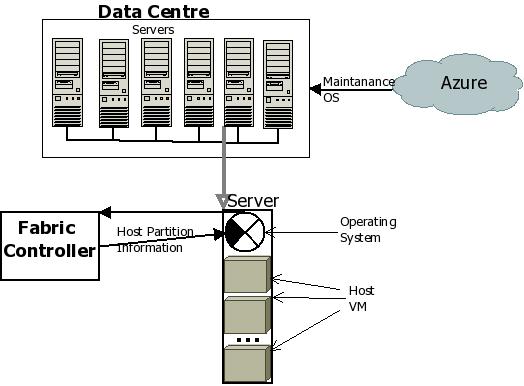


Figure : Azure Platform architecture

Since SQL Azure resides on the Azure Platform, SQL Azure also has a similar architecture. According to Campbell et al.(2010), in Microsoft SQL Azure Database, the Infrastructure and Deployment Services layer coordinates the activities of the nodes in the cluster and helps in booting the servers and downloading the maintenance OS ().

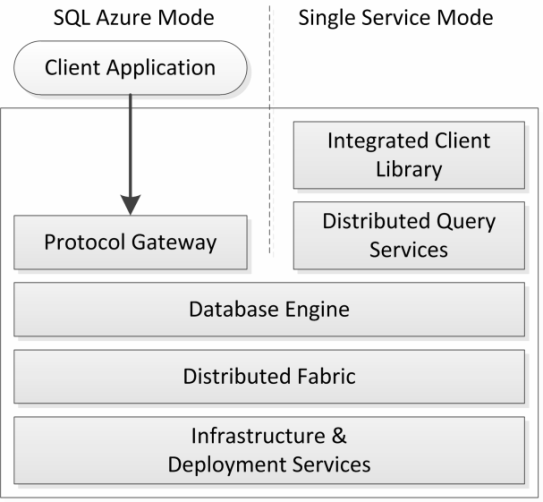


Figure : SQL Azure Layers (Courtesy: Campbell et al. (2010))

The Distributed Fabric layer is similar to the Fabric Controller in Azure Platform and does the tasks of partition management, failure detection etc (). The Database Engine is based on the kernel of the Microsoft SQL Server engine which helped in retaining most of the SQL features pertaining to relational database models as seen in the way the tables are created and queries executed above. The Distributed Query Service is the layer responsible for routing the SQL queries to the appropriate partitions.

The Protocol Gateway, used only in Microsoft SQL Azure Database, supports the protocol of SQL Server and is responsible for the database connections and bindings to the appropriate nodes in the cluster. It also maintains client sessions.

Microsoft SQL Azure Database implements horizontal partitioning, where data is splits or sharded into fragments and these fragments are stored on different nodes. In SQL Azure the rows in a table are split and the fragments are spread on the nodes in the Microsoft Data Centres. For example, in the Student database, the student table is split on DOB “14/01/1984” and the rows containing “14/01/1984” in its DOB column would be grouped and saved as a fragment on one of the nodes in the data centre (). Queries spanning multiple partitions do not have full ACID compatibility, unlike queries that involve a single partition. This might be seen as a drawback of Microsoft SQL Azure Database.

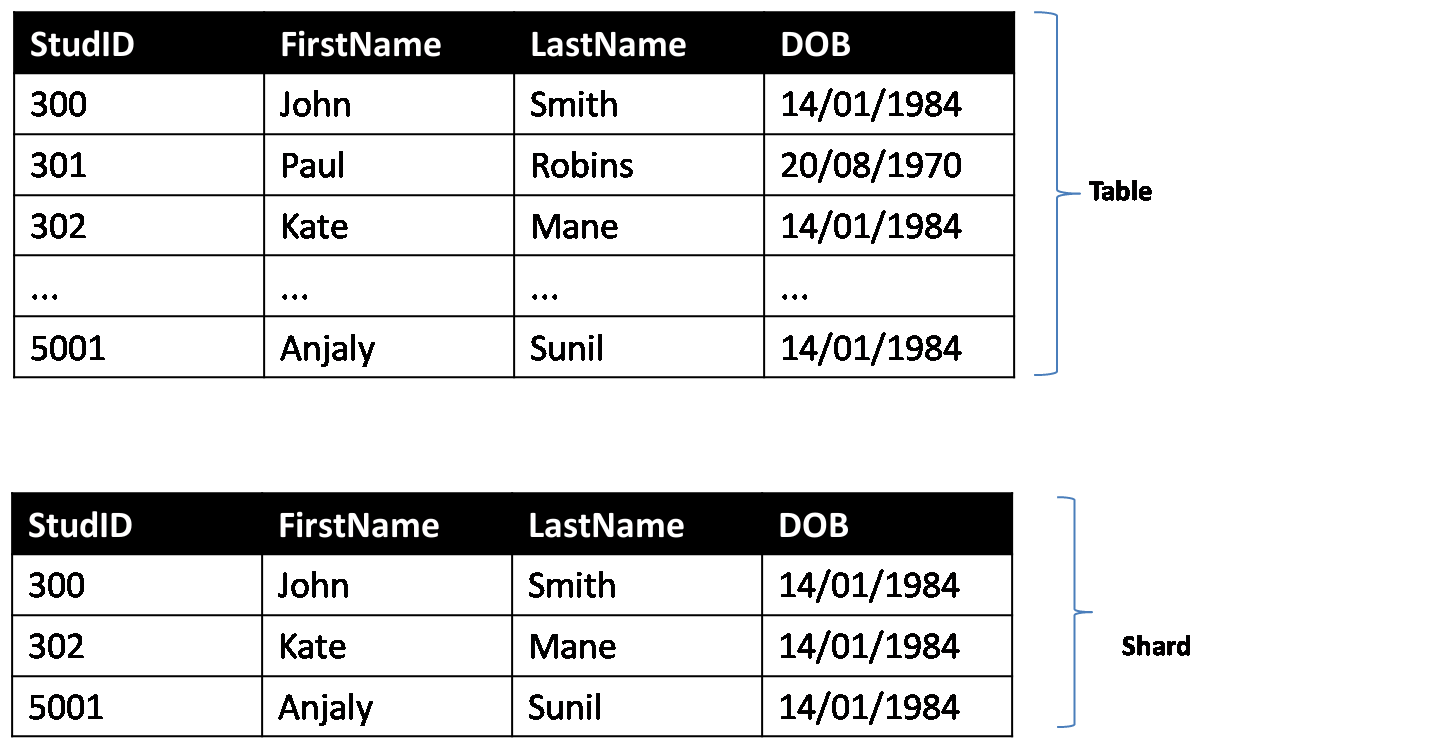


Figure : Horizontal partitioning in SQL Azure

## How SQL Azure works

Each time a new database is created by a user, SQL Azure creates three replicas, using the CREATE DATABASE command and saves each replica on different servers in Microsoft Data centres, within the same geographic location. Replicating the data and synchronising the replicas are handled by SQL Azure and the client does not have to worry about the integrity of the data (Microsoft, 2010a). As studied in previous chapters, replication would help in times of failures and provides good data availability and fault tolerance. In the event of a node failure, SQL Azure automatically transfers requests to a replicated copy. Hence, clients do not have to worry about data loss or data recovery. SQL Azure handles the administration of the database like optimising queries, security administration, schema creation etc (Microsoft, 2010c).

Data stored in SQL Azure is accessible using the ADO.NET and other Windows interfaces making data migration from SQL Server to SQL Azure easily, letting clients have their SQL Server services, like the reporting services, to work in their relational cloud DBMS too (APC, 2010).

SQL Azure also supports cloud bursting , by allowing clients who need additional data storage to use SQL Azure Databases as and when they need it and dropping the databases after the use.

## Summary

This chapter explored the data storage and the underlying architecture of SQL Azure and showed how similar SQL Azure and SQL Server are.

Chapter 9

# NoSQL vs. RDBMS

## Introduction

Previous chapters have shown that most of the popular and current cloud databases use key-value DBMSs or NoSQL DBMSs rather than the traditional RDBMSs. RDBMSs have been around for nearly 20 years, and have been one of the most widely accepted and used database management systems. Over the years, many attempts have been made by many new DBMS to overthrow RDBMSs, but not many have succeeded as RDBMSs were widely used and had a large user-base (Bain, 2009). RBMSs were potentially used by most kinds of applications to store their information. Some DBMSs do have evolved with specialised architectures which could handle specific problems, which the RDBMS is not too efficient at (Stonebraker et al., 2007). For example, Vertica is designed for query-intensive applications and large data warehouses, Monet is designed for intensive data mining etc. But all this while, RDBMSs have not been considered extinct or on the way out, like it is being predicted now with the advent of many non-RDBMSs for the cloud.

This chapter compares the NoSQL DBMSs and the RDBMSs based on features related to the cloud and also attempts to determine whether RDBMSs could be more adaptable in the cloud. The following section briefly looks at RDBMSs and its core features, before comparing the NoSQL DBMSs against the RDBMSs.

A brief background about relational model and RDBMS has been given in Section . RDBMSs have always been widely popular and supported by most database vendors. Many major vendors like Oracle and Sybase etc, have developed and maintained RDBMSs for a long time now. This is also one of the reasons why RDBMS have a large user-base and such popularity. RDBMSs are suitable for storing structured data extremely well, and are suitable to most practical data storage needs that are commonly found, for example, inventory databases, retail databases, school databases etc. RDBMSs also help preserve the relationships and constraints within the data.

## NoSQL vs. RDBMSs

NoSQL DBMSs are fundamentally different in architecture from the RDBMS and are designed to address specific issues that are cloud related, example scalability, large amounts of data etc. NoSQL DBMSs grew in popularity as cloud concepts emerged. This is mainly because cloud soon became a strong backbone for many applications and more users are migrating their applications and data storage onto cloud (Wilkes, 2010).

When it comes to deciding which database type, i.e., relational databases or NoSQL database, is more suitable to cloud environment, views and opinions of most users and researchers are divided on this topic. While most cloud service providers and cloud database users predict the NoSQL DBMSs to be best suited for the cloud (Vogels, 2009, Linthicum, 2010, Jackson, 2008) others are confident that RDBMSs are to stay and would rightly take its place in the cloud in the near future (Bain, 2009, Wilkes, 2010, Clarke, 2010).

What is mostly agreed by RDBMS supporters as well as NoSQL DBMS supporters is that cloud RDBMSs are still evolving and have to incorporate essential features, like distributed environment, data redundancy, high scalability etc., for storing data on the cloud RDBMS supporters predict RDBMSs to soon evolve and function well on the cloud while most NoSQL DBMS supporters predict that RDBMSs would never be fully cloud compatible because of its limited support for such features. This section looks at some essential features that are needed for efficient and practical data storage on the cloud and NoSQL DBMSs, traditional RDBMSs and cloud RDBMS Microsoft SQL Azure are compared against these features:

* *Scalability*: On the cloud structured or unstructured data, containing user information or cloud service’s information, is spread across several computers that may span several data centres in different geographic locations. When data is duplicated and spread in such a large scale on several nodes, it is quite important that databases are scalable. Scalability, in the context of cloud storage, often refers to the ability of dynamically incorporating changes to user-base or number of nodes without affecting the functioning of the databases or the availability of data to the users. In other words, even when more nodes are added, or when more users access the same data, cloud DBMSs should cope with the increased workload and yet maintain the same throughput.

Relational databases of most RDBMSs get really complex when it comes to scaling to multiple nodes as these are best suited to run on a single node. Even when Relational databases are used in a distributed environment and data is spread on several nodes, it becomes really complex to manipulate or access the data. This is mainly due to normalisation, as data is not duplicated and the data is stored in different tables. Complex queries, spanning several tables in various data centres, would have to be executed to retrieve or write data (Bain, 2009). RDBMSs can scale up by adding more resources to the single node, like more storage or power etc, but perform poorly while scaling out i.e., when additional nodes are added (Ciurana, 2010). Since in traditional RDBMSs, data is not replicated on other nodes, it also raises the issue of node failures and data availability.

As seen in Chapter 8, SQL Azure, is a cloud RDBMS in which scalability is achieved by partitioning a database across several systems (Microsoft, 2010a). But queries in SQL Azure do not have full ACID compatibility, i.e., ACID properties are supported in transactions across tables in the same database but not across multiple databases. Queries in SQL Azure are limited since the queries would have to span multiple partitions to fetch data .

To match the scalability demanded in the cloud, most RDBMSs would require additional layers on the machine hosting the RDBMSs. For example, in Microsoft SQL Azure these complexities are handled by the various layers of its architecture like the Fabric Controller, Distributed Query Services etc. Users of SQL Azure are also involved in determining the amount of partitioning required in the SQL Azure databases and this is not automated as of yet (Microsoft, 2010). This means that the complexity, with respect to partitioning, is not completely hidden or maintained internally within Microsoft SQL Azure. Although such layers would only enhance the complexity of maintaining the RDBMS and could adversely affect the speed and efficiency of data access, it can be seen in Microsoft SQL Azure Database that the attempt has been successful and worth the internal complexity.

NoSQL DBMS are horizontally scalable, which means that these DBMSs scale out by adding more nodes to the existing cluster of nodes that store the databases. This means that as more storage space is needed, or when the user-base or workload increases, more nodes can be added to share the workload without changing the existing configuration of nodes. In most NoSQL DBMSs, this is dynamically done by the DBMS and the users do not have to worry about it. As seen previously , in Dynamo and Cassandra, as more storage capacity is needed, new nodes are added without any reconfiguration. This also involves steps like replication of data across the nodes in the cluster, synchronisation of the various replicas within the whole cluster etc. that are managed internally by these NoSQL DBMSs. These capabilities make NoSQL DBMSs highly scalable in the cloud than their RDBMS counterparts.

* *Flexibility*: Data on the cloud is commonly unstructured and could change often causing schema changes like increase in columns etc. A DBMS is flexible on the cloud when it can incorporate such changes with minimum or least changes to the schema of its databases. Storing such data requires the databases to have a relatively simple schema.

NoSQL databases are often referred as schema-less databases as these databases do not impose any schema on the data they store. Hence it is relatively simple to add values or ColumnFamilies. For example, in Section , it was shown how one row in a SuperColumn has more columns added without affecting the structure of the table. In general, the flexibility of the NoSQL DBMSs is very suitable to the cloud environment.

In RDBMSs, the relational databases adhere to a schema that is designed and decided before the development stage. Schemas define the tables, the relationship and constraints between data etc. Any changes to this schema would involve the whole database to be altered, which may affect the constraints and relationships between the tables too. Such a change in database design or table layout is complex and hard to achieve in relational databases and is expensive when done after the database deployment.

In Microsoft SQL Azure Database a default schema, DBO, is applied to all the databases created and this commonly refers to the whole structure of a database (MSCERTS, 2010). Changing the schema requires changing the schema to a user-defined one and then changing the ownership of the schema too. All these have to be done manually and could be time consuming or complex. This makes the RDBMS less flexible on the cloud.

* *Failure Management*: As explained before, data storage on the cloud involves several large global data centres that have many machines and other hardware devices, which could be prone to failure any time. Such node failures, network delays or network failures can cause data loss which is a common pitfall on the cloud. Replicating data on different nodes and data centres is necessary and ensures that even in times of failures, data is still available from some other node in the cloud.

Most NoSQL DBMSs store and maintain many replicas of its stored data across several nodes and data centres using replication strategies that are automated and internally handled by the NoSQL DBMSs. Some NoSQL DBMSs like Cassandra allow users to decide the number of nodes that should have the replicas. HBase maintains write-ahead-logs on its region servers to track the replication process and the HBaseMaster replicate data to all nodes in its cluster. Using such ways, most NoSQL DBMSs ensure that even during node failures, users are not affected and have data access at all times. Even when an entire data centre shuts down, replicas copied on the nodes of a different data centre can be retrieved and provided to users.

As seen in the cases of Cassandra and other NoSQL databases, nodes are periodically scanned and in cases of failures, the data residing on the failed nodes are transferred to other active nodes. All these steps are internally handled by the NoSQL DBMSs and users of the DBMSs are relieved of all these tasks.

On the other hand, in traditional RDBMSs, most data is stored on a single node or on a few nodes. Failure of nodes could potentially cause loss of data. But it is very common that such single nodes are always backed up periodically so that data till the last backup can be recovered. There could be some loss of data if there is some gap between the point of backup and actual loss of data. For RDBMSs to provide failure management on cloud it would be necessary to have replicas of its data on several nodes. This has been seen in Microsoft SQL Azure where three replicas of a database are maintained and each of these copies reside on different servers in Microsoft Data Centres within the same geographic location. Although failure of a single node does not affect data availability or cause loss of data, the result of the failure of all the three nodes is still a worry. As explained before, the replications and the synchronisation of the replicas are all done internally by the various layers of the SQL Azure architecture. It is quite evident that the replication is limited in SQL Azure when compared to the replication deployed in other NoSQL DBMS like HBase or Bigtable etc.

* *Data Availability*: On the cloud, most services or applications have large user-bases and many users concurrently access the same data. In such cases, availability of data even while the data is actually being accessed or modified by another user at the same time is crucial (Raja 2010a).

Most NoSQL DBMSs attempt to achieve the A (Availability) of the CAP theorem and thus give due importance to ensure that under most circumstances, data is always available to the user. NoSQL DBMSs do so by maintaining replicas of data on several nodes at the same time and provide versioning of data. For example, Cassandra and Dynamo use vector clocks to check the validity and correctness of the replicas. Such techniques ensure that users always have the latest and updated data available. All these tasks related to replication, updating and version-checking are done internally and users do not have to worry about the correctness of the data presented to them. NoSQL DBMSs also provide efficient locking mechanisms, like Chubby lock service in Bigtable, while writing or updating data which ensures that no read or write conflicts occur that may hinder the availability of data.

In traditional RDBMSs, ensuring availability of data is relatively simpler mainly because data is not replicated and is available using simple queries from a single node. It may still be possible that a large part of the database may be unavailable when large amounts of data are being updated. However, on cloud, RDBMSs would have to copy their data on different nodes to ensure data availability as well as scalability and data recovery. In Microsoft SQL Azure, three replicas are maintained that help in ensuring that data is available to users at all times. But this is again limited when compared to the high availability of data in NoSQL DBMSs.

* *Data Access*: On cloud, accessing stored data is complex as it is spread across several nodes. But it is essential that users are able to access their data with simplicity and that the users get the correct data. The internal details of the data storage have to be hidden from the user and locating the correct database or the value associated with a key has to be done internally by the DBMS.

NoSQL DBMSs provide APIs to access and maintain databases. Retrieving data using these databases are slightly complex and would involve object calls to the databases (Jackson, 2008).This means that in NoSQL DBMS a key has to be provided to query data in a database and the DBMS looks up the object or the value that is associated with the given key. While this could be useful for developers who work with NoSQL databases as object calls are easier to handle in object-oriented programming languages, other users of the DBMSs may be faced with limited querying abilities as SQL and other querying languages are not fully supported. For example, multiple Joins are not supported to retrieve data. Most NoSQL DBMSs do not provide triggers or views either.

In traditional RDBMSs, accessing the data is simple and very efficient. Most RDBMSs support strong querying for users to fetch data without any complexity. On the cloud, RDBMSs are not very simple to maintain if it involves a broad range of tools or extensive querying capabilities . Currently the cloud RDBMSs do not provide such a wide range of tools or options and have limited querying. Microsoft SQL Azure provides support for simple querying and is quite easy to use. Despite lacking strong querying features, the simple querying itself involves complex joins and searches as the data is replicated and spread on different nodes. While this is transparent to the users, RDBMS administrators often have to face these complexities and this also makes the internal functioning of Microsoft SQL Azure Database quite complex. As mentioned previously, some of the tasks have to be maintained by the users, for example designing and changing the schema etc.

* *Data Validity*: As mentioned previously, data that is stored on the cloud is often unstructured or semi-structured and may contain any type of data. Constraining the data in terms of data types like integers, long, float etc may not be always feasible to store the bulk amount of data that the cloud always handles.

NoSQL databases address this by storing every value associated with the key as a string or blobs. Data is not distinguished on the basis of its data type. This allows NoSQL databases to store any kind of data that may be diverse or that could be changed often by the user. While this is a benefit, a potential problem could be that the software applications that handle these data could face errors like number format exceptions etc (Jain, 2010).

Traditional RDBMS on the other hand require data type to be specified for each column. This helps in avoiding the error of storing wrong data types in the columns. In the cloud ,although such constraints prevent errors, these also make the administrative tasks more time consuming and complex, especially when large amounts of data is being stored. In Microsoft SQL Azure many different data types are supported, like integer, float, binary etc. SQL Azure supports XML data and also supports geographic data types, which presents data in a round-earth coordinate system (MSDN, 2011a). More information about such data types can be found in MSDN (2011b).

* *Controlled management*: As seen in Raja (2010c), cloud DBMSs usually provide central management of databases, This is necessary in the cloud as it ensures that all the users can manage their databases with ease and simplicity. If the cloud databases are not centrally managed and the database management is left to the users, the very purpose of having cloud data storage is ruined. The complex data storage techniques like replication, synchronisation of replicas etc would require lots of time and would be expensive from the view point of the user.

NoSQL DBMSs provide central and controlled management of its databases. All the cluster management, replication of data, synchronising of the replicas, database management etc are done internally in the NoSQL DBMSs and are hidden from the users.

Traditional RDBMSs also provide good database administrative and management tools. These have been tried and tested over the years and have grown to include a wide variety of features, like back-up and recovery facilities, security management, concurrency control mechanisms, query optimisation etc. In the cloud, when such extensive features are deployed it would make the cloud RDBMSs complex ,slow and can cause delays in accessing data. However, as mentioned previously, Microsoft SQL Azure does provide various layers that manage the databases and provide some key features like automatic data replication, synchronisation of replicas etc. It currently does not provide all the extensive features that are found in traditional RDBMSs.

* *Cost effectiveness*: One of the reasons why users prefer cloud-related services and cloud databases is mainly because of the associated cost benefits as storage is cheap and users pay only for what they use. Most NoSQL DBMSs are open source databases and this helps in reducing considerable costs for users of the cloud DBMSs. NoSQL DBMS also have a limited set of database administration tools and this eventually means that users do not have to pay for the features they do not need.

Traditional RDBMSs, as mentioned before, offer a wide variety of features and tools for easy database management. Users quite often have to pay for many unnecessary features too when they buy an RDBMS. On the cloud this could lead to less efficiency in terms of cost. SQL Azure has many pricing options, from which the users can select the correct option (Microsoft, 2010).

* *Migration*: In the cloud many different types of cloud DBMSs are available and the users choose a DBMS according to their needs. Migration refers to the transfer of data from one type of database to a different database, probably of a different vendor. It would be really cost effective if a user could migrate from one DBMS to another without incurring much loss , both in terms of loss of money and loss of data.

Majority of the current cloud DBMSs are NoSQL DBMSs and hence share some of the features and key concepts like key-value pairing, or partitioning data etc. Moreover, most NoSQL DBMSs have adopted features and architecture of existing NoSQL DBMSs like Bigtable or Dynamo etc. This also makes it possible for users to migrate their data form one DBMS to another without much change in the structure of their databases. Another important factor is that most NoSQL DBMSs are open-source, thus giving the users or developers the freedom to add features or change the DBMS structure to suit their application and storage needs. Since cost of purchasing the data storage space is very less, users do not incur heavy losses even if they have to buy data storage in a different DBMS altogether, even if the architecture is different.

Traditional RDBMSs mostly have similar architecture and concepts like relations, query-optimisation, normalisation etc. Unless the RDBMS is an open source implementation, the migration costs would be higher when switching RDBMSs as most RDBMSs come with a rich set of tools and are costlier than cloud DBMSs. It might also be necessary to change the application code of the user to suit the change and this could result in time delays too. Microsoft SQL Azure provides easy migration from SQL server to SQL Azure, as seen before in the Microsoft SQL Azure case studies in Raja (2010a).

Based on the above discussion, below summarises the above comparison of NoSQL DBMSs and RDBMSs and presents the results in terms of ‘High’, ‘Moderate’ and ‘Limited’. ‘High’ refers to the high level of support a DBMS provides for a feature and ‘Limited’ refers to the least level of support for a feature. While the comparison shows in general that RDBMSs are less suited for cloud as of now, there are good attempts made by many cloud service providers to bring RDBMSs into the cloud. Microsoft SQL Azure, Drizzle are some of the cloud RDBMSs that are currently prevailing that supports not only cloud features but also addresses most of the problems like scalability, querying, data availability etc efficiently. More time and research is being invested to bring out more cloud suitable RDBMSs by cloud service providers and this is showing remarkable progress too.

|  |  |  |  |
| --- | --- | --- | --- |
| **Cloud features** | **NoSQL** | **Traditional RDBMS** | **Cloud-RDBMS** |
| Scalability | High | Limited | Moderate |
| Flexibility | High | Limited | Moderate |
| Failure management | High | Limited | Moderate |
| Data Availability | High | High | Moderate |
| Data Access | Moderate | High | Moderate |
| Data Validity | Limited | High | High |
| Controlled Management | High | High | High |
| Cost effectiveness | High | Moderate | Moderate |
| Migration | High | Moderate | High |

Table : Summary of Comparison of NoSQLs, traditional RDBMSs and Cloud-RDBMS (SQL Azure)

## Summary

This chapter discussed how NoSQL DBMSs were fundamentally different from RDBMSs. This comparison showed that in the past couple of years much progress and advancement have been made to bring RDBMSs into the cloud and promising steps towards developing cloud suitable RDBMSs is underway. This comparison showed the level of popularity and suitability the NoSQL DBMSs benefit from on the cloud.

Chapter 10

# Conclusion

Cloud storage shares most of its benefits and disadvantages with cloud computing. Understanding cloud storage involves learning about the underlying architecture and the various approaches used by the current cloud storage service providers.

Cloud storage, although a vital component of cloud computing, is still evolving and growing. Appropriate methods of data storage and organisation on the cloud storage are vital for the growth of cloud storage. Although users are affected by its many shortcomings, it’s economic and maintenance advantages to the users outweigh its drawbacks.

Usually, users of cloud services choose cloud databases depending on the type of data they have to store, like structured or semi-structured etc., or depending on the amount of data they have to store or depending on the scope of their applications. Scope of the application refers to the user-base of the client applications. If the user applications have a large user-base they would prefer a cloud database that can meet all the user requests, or if their application tends to have a diverse user-base belonging to different geographical locations, they would prefer a cloud database that can span several data centres. For example, Facebook chose Cassandra to match its scalability of reaching several data centres while Bigtable was developed to store large amounts of structured data that would scale greatly (Raja, 2010a).

Direct implementation of the several existing database systems has mostly failed in matching up to the scalability, performance, and load balancing necessary in the cloud environment. For example, SQL Server or MySQL have not scaled well with the write intensive and replicated data on the cloud. Collaborating the features of many existing systems have already given scope for better storage systems as seen in Microsoft SQL Azure Database, Cassandra etc. Understanding some of the popular current data storage techniques on the cloud helped in realising that there is great scope for improvement of the existing DBMSs so that these are more adaptable to the cloud environment. Current DBMSs pay more attention to faster response time, better scalability and efficient fault tolerance. The trade-off is that data is spread out on the cloud storage with large amounts of data being either replicated or densely interconnected.

Current data stores used by cloud providers vary significantly in terms of their performance and features. Most databases currently deployed and being used are suitable to satisfy certain problem areas, rather than being a perfect solution for most problems, like how Facebook uses Cassandra to store their internal data and HBase to store the data of their new messaging service.

Understanding the underlying architecture and data models of cloud DBMSs showed that most existing databases altered traditional database storage mechanisms like the relational database systems or the key-value pair etc in different ways. This different modelling of existing database architectures is mostly due to the difference in the nature and operations of the cloud services and the diverse user-base that these services have. Some of the users of open-source cloud databases alter the structure further to suit their storage needs. Finding a common solution to such vast and diverse storage problems is currently being researched far and wide, with interesting options being developed.

This study has shown two such databases that are designed to address different problem spaces. While Dynamo is a database that is currently used only within the Amazon internal framework, it has been adopted by several other databases like Cassandra, Voldemort etc. On the other hand, HBase is inspired by Google’s Bigtable. This in itself shows the way databases and data architectures are re-used and evolving. Existing cloud databases as well as non-cloud database architectures have been revolutionized to create new databases that satisfy current cloud –storage related problems. As more and more alterations are made or new forms of database architectures are implemented, more problems are being addressed and a variety of options are being made available to users. This has been evident in this study, where the studied databases involved solving some of the common problems currently being faced by cloud applications in terms of data storage.

Although all the database operations and administration are invisible to the clients using the storage mechanisms, better data organisation and data models could improve memory management and spanning costs incurred by the cloud databases. Much research and work is going on for achieving the same, and the future of cloud storage and the storage services on cloud look promising, with plenty of research going on by key players like Amazon, Microsoft, IBM, Oracle etc.

With the advent of data storage in the cloud, most vendors and users wanted a quick and rapid movement to store their data on cloud. In today’s cut throat competitive market, this was led with many major vendors investing time and money into research and development of databases suited for cloud. The key-value model was initially found more suitable to the cloud due to its simplicity and support for scalability. This sudden rush could possibly be one of the reasons why attempts to bring RDBMSs into cloud was slow, as adapting RDBMSs to the cloud is a slow, daunting and complicated task.

In the past couple of years much progress and advancement have been made to bring new, better scalable and faster DBMSs into the cloud. This is a promising and encouraging step towards developing more cloud suitable data stores. A possible scope for future study is included in Appendix D. The development of DBMSs that could address almost all cloud related problems might take some time, but its arrival would be beneficial to all cloud users, whether they access cloud data stores or only cloud services, because in the end, without a scalable and reliable data store no cloud services would perform efficiently..

Appendix

# Appendix A

User Case studies for Cloud DBMSs Dynamo, Cassandra, HBase and SQL Azure are given below.

* **Case Study for Dynamo**
* ***Amazon*-** As mentioned before, Dynamo is used by many services within Amazon internally. Since Dynamo provides pluggable storage for persistence, it can be used for different services to suit their storage size needs.

Amazon uses Dynamo for its services like the shopping cart, best seller lists, customer preferences, sales ranks, product catalogues etc. This is primarily because these applications store data along with their primary keys and identify them with these keys too. Also, these services do not have the need for complex queries to read or write data into the data store. These services also have a large user-base, some scaling even to more than 3 million users during peak times. These services thus required managing several active sessions and maintaining the state of the users too.

Using a relational database would be inefficient for such applications. These services thus needed a simple key-access database, which could replicate data even in times of failures and outages, as such services needed data availability at all times for their users. As mentioned previously, Dynamo was aimed to handle fault tolerances and network or node outages efficiently by maintaining data availability even in such failure modes.

Amazon has several large data centres and good infrastructure that helps them in providing such a scalable data storage mechanism like Dynamo, where nodes span several data centres across the world.

* **Case studies for Cassandra**
* ***Facebook* –** As previously mentioned, Facebook has a large user base and large amounts of diverse data to store. With such large amounts of data that was interconnected, partitioning was also not a viable option (Hoff, 2009). Partitioning required the data to have frequently used common features. Facebook uses Memcache as the caching layer so that large amounts of active data can be maintained on the cache for easy access to the data.
* ***Digg*-** Digg is a social news website that allows users to submit stories and vote stories online (Digg, 2010). In its initial years Digg had MySQL as the backend. But by 2010, due to increased popularity, the Digg development team did major changes to the website (Quinn, 2010). Changes included migrating from MySQL to Cassandra, changing the interfaces etc. The switch to Cassandra was mainly motivated because of the lack of higher performance and complex write functions in MySQL. MySQL was also not scalable with the redundant and often interconnected data within the data stored by Digg users (Quinn, 2010).

Migrating to Cassandra gave Digg failure tolerance as data was now replicated on several nodes and also gave it write intensive operations that were easy to perform even with large replicated data.

Digg team also did some changes to Cassandra by reducing the overhead in logging when updates or requests are handled by the nodes, giving better comparator speed and row-level caching etc (Quinn,2010).

In Digg,, the propagation of the updates on data is replicated through the nodes and the read operations are kept simple. Digg does not normalise the data, which is also one the reasons why Digg migrated to Cassandra as MySQL was a normalised relational model (Obasanjo, 2009). According to Eure (2009), the Digg team used the concept of buckets, which contained the list of all the users who viewed a news item that the current user was viewing. So for each (user, item) pair there would be a set of buckets. Whenever an item was viewed Cassandra was populated, which resulted in fetching all the users connected to the current user and inserting a column into each of the related user’s buckets. This allows easy accessibility to users lists as well as data connected to users.

* **Case Studies for HBase**
* ***StumbleUpon***: StumbleUpon is a discovery engine that filters through the vast amount of information available on the internet and takes the users directly to the web sites that match their preference. The web sites to which users are directed to are the ones that are recommended by other users who had a similar search preference. This is different from searching, as instead of giving the users a list of results; StumbleUpon directs their users to the relevant web sites. They also provide collaborative ratings, community participation, toolbars etc for their users to save time while searching for their preferred contents (StumbleUpon, 2010).

StumbleUpon had to maintain and manage a lot of user signals, like thumb-up, share etc. These had to be stored and displayed to other users for better decision-making. The decision-making involved what web sites had to be shown to the user to match their preferences depending on these recommendations and user feedbacks (Gray, 2010). According to Gray (2010), this required the data to be organised, quickly retrieved, refreshed or even deleted correctly.

The distributed nature of HBase helped StumbleUpon as data was spread across several cheap machines, instead of one or two expensive servers. It also gave them the provision to add more computers as their user data grew. Failure of computers also did not pose much of a threat as the logging in HBase helped in retrieving the data after any trouble.

* ***Facebook***: Although Facebook uses Cassandra as their data store (Raja, 2010), it uses HBase for the data store for their new messaging service. The Facebook team required an objective pattern to handle their volatile data that was constantly growing, as their user base increased. Facebook engineers claim that the eventual consistency of Cassandra is what posed a problem for this new service (Muthukkaruppan, 2010).

The features of HBase like the logging, the underlying file system HDFS were other key reasons for this migration. Muthukkaruppan (2010) also claim that the past experience of working with Hadoop and HDFS by the Facebook engineers also prompted them to use HBase instead of Cassandra.

Peschka (2010) claims that the difference in the replication strategies of Cassandra and HBase is also a possible reason. Unlike Cassandra, the replication method used by HBase requires only one server node to respond to the write request (Lipcon, 2010). This makes it easy to perform write and read operations even during busy times.

* **Case Studies for SQL Azure**

Amongst these clients 3M and Siemens used the Microsoft SQL Azure Database as a part of the deployment of the Azure platform, while Quosal used the Microsoft SQL Azure Database as a DaaS.

* ***Siemens***: According to Microsoft (2009), Siemens deployed the Azure platform so that it could make its software updates and installations within its common Remote Service Platform(cRSP) easily. cRSP integrates the many registered Siemens devices used around the world. This migration was made so that more devices could benefit from automatic updates and also reduce the large costs involved in individual patching and deployment to all the devices in its large platform. Siemens benefitted from the SQL Azure database as it was possible to synchronise the blob storage in the Azure platform with the relational SQL Azure. SQL Azure helped Siemens by freeing them of load balancing and other security measures involved in database administration over their large distributed platform. It was also easy for them to reuse their existing internal SQL code while migrating to the Azure Storage.
* ***3M***: Another client to profit from the Microsoft SQL Azure Database is 3M. 3M benefitted from the SQL Azure database by having SQL Azure manage the images used by their designers in their Visual Attention Service (Microsoft, 2009). 3M found SQL Azure extremely useful as SQL Azure relieved it of database management, where databases contained too many images and database operations involved complex analysis.
* ***Quosal***: Another client was Quosal, which saw a profound increase in its sales after SQL Azure was used. Quosal offered pay-as-you-go softwares as a service. When their user base for these software services started increasing, Quosal implemented the Microsoft SQL Azure Database. This move saved their user data, which included quotes and proposals for economic markets, on Microsoft data centres around the world. This saved Quosal from investing heavily in their own data centres to host databases to support their software service. Moreover, migrating their data and integrating SQL Azure with their existing applications were made simple due to the similarities between SQL Azure and SQL Server. This benefit was more possible because Quosal had all their applications built on the Microsoft .NET framework and the SQL Server. The only changes they had to incorporate for this easy migration was changing the connection settings in their database connections.

While this proved beneficial to Quosal and many other clients who were already using Microsoft products before migrating to SQL Azure, users who worked with different applications and programming environments could possibly find it troublesome to migrate their applications to Windows Azure or the SQL Azure.

# Appendix B

This section describes the various steps taken to install and run Cassandra on a single laptop machine.

The implementation of Cassandra included creating a Student database for the example used in the study. The implementation was carried out on a single node and had small amounts of data. Due to such limitations, it was not possible to fully leverage the various cloud related benefits of Cassandra. The main purpose of this implementation is to understand the level of complexity of cloud data storage services .

## Cassandra Installation

**Prerequisite:** JDK 1.6

**Step 1:** Download a stable version of Cassandra from <http://cassandra.apache.org/download>

**Step 2:** Unzip the contents as an administrator to the file system as shown in Figure 46.

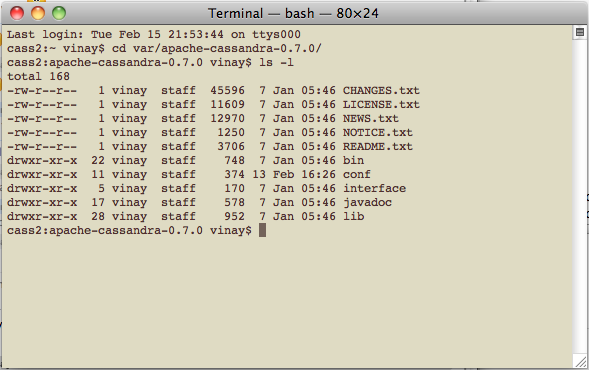


Figure : Cassandra Installation

**Step 3:** Validate the configuration files and bin folders as shown below in Figure 47 and Figure 48.

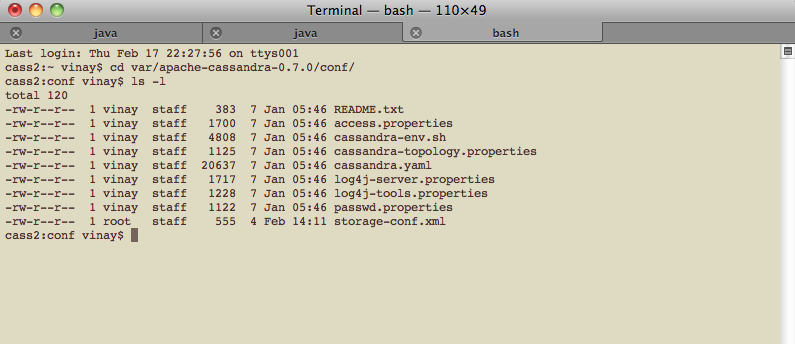


Figure 47: Configuration Folder

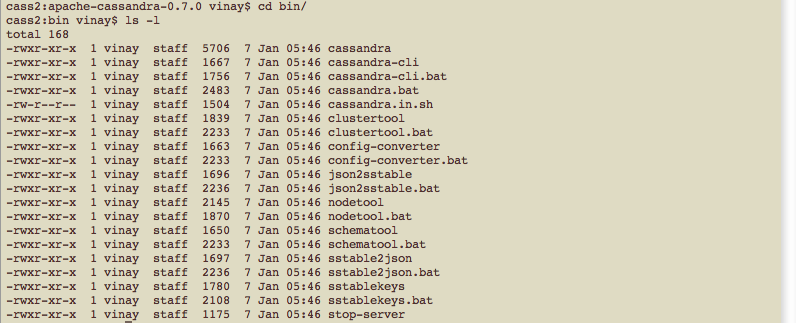


Figure 48: “bin” folder

**Step 4:** For Single node server check and ensure that listen\_address and the rpc\_port parameters are configured in conf/Cassandra.yaml file. Listen address would be the IP address to which the server would be bound to and rpc port would be the port that the server would listen on. Refer for more details

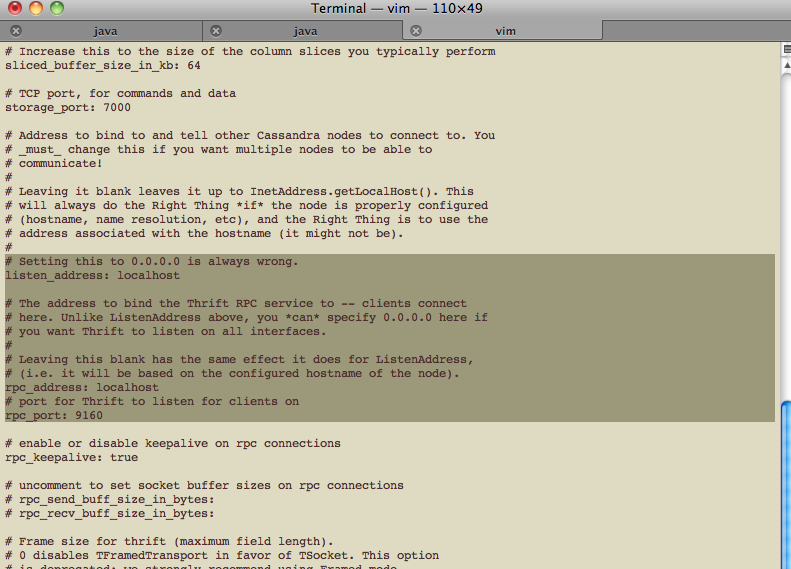


Figure 49: Cassandra.yaml

**Step 5:** To start the server run command bin/Cassandra –f with sufficient privileges. Once the server is started, the logs should be displayed in the console as seen in Figure 50.

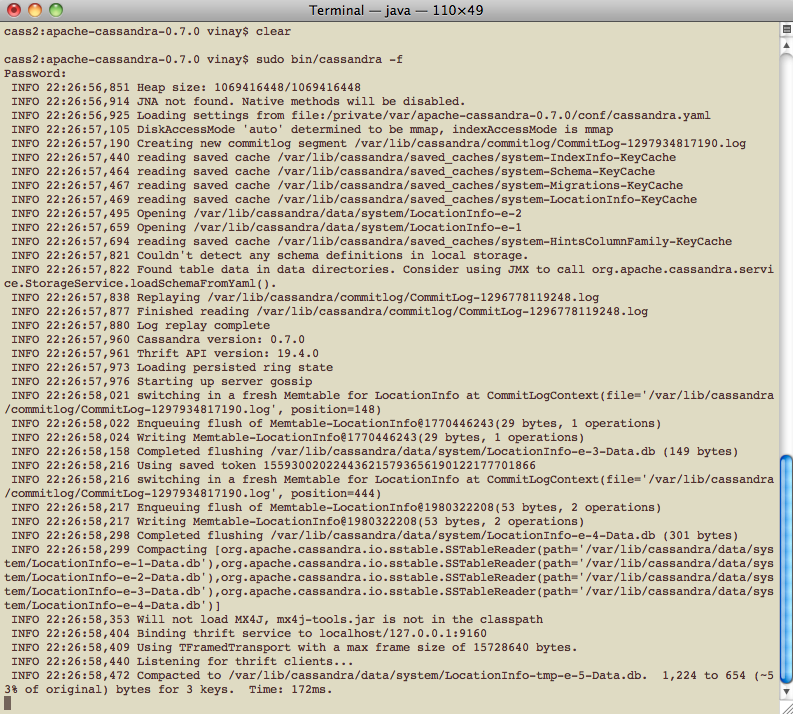


Figure 50: Cassandra in running mode

## Add and retrieve Data from Cassandra

**Step 1:** Connect to the server using the thrift client as shown in figure below. Use “?” to display the list of commands in thrift client.

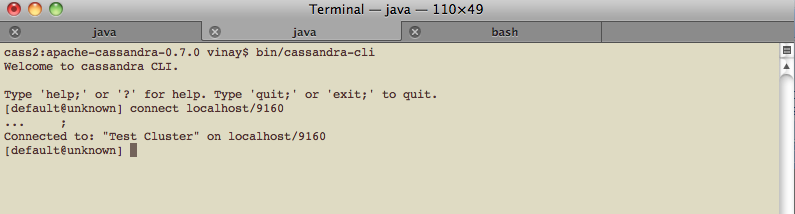


Figure 51: Thrift client

**Step 2:** To show all keyspaces in Cassandra, use command:.

*Show keyspaces*

**Step 3:** To create keyspaces and column open *conf/storage-conf.xml* and add values as shown in below:

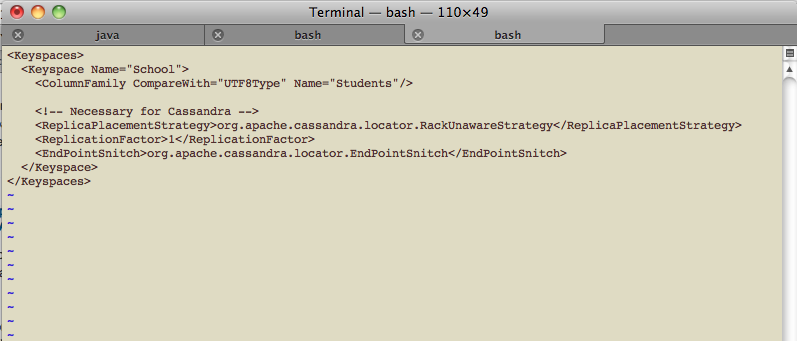


Figure 52: storage-conf.xml

**Step 4:** User can change to keyspace using command

* *Use keyspace <keyspacename>;*

**Step 5:** Data can be entered into the Key Space and Column family using the ‘*set’* command. A typical set command would be:

* *Set <cf>[‘key’][‘column’]=[‘value’];*

OR

* *Set <cf>[‘key][‘column’]=function(‘value’);*

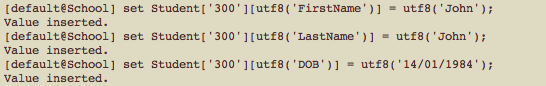


Figure 53: Set Data

**Step 6**: Data can be retrieved using the thrift APIs using the get command as shown in Figures below (, )

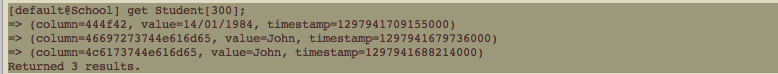


Figure 54: Get Super Column

Cassandra-get option 2.png

Figure 55: Get Column

# Appendix C

This section describes briefly the architecture of Windows Azure Platform. This describes the various steps right from when a server starts in a data centre.

As seen in Figure 43, in the Azure platform, the servers in the Microsoft data centres are connected to the Windows Azure network .These servers are configured to boot from the network and each of the servers download the maintenance OS. This OS talks to the Fabric Controller. The Fabric Controller is the service that knows what resources exist on the platform and manages the overall platform. Fabric Controller talks to an agent residing within the maintenance OS and sends instructions to create a host partition for hostVMs , which are special VMs (Lemphers, 2008).

The host VMs, are responsible for controlling access to the underlying hardware of the server. Customer applications on the cloud are run on one of the guest VMs. The guest VMs are connected to the host VM through a VM Bus. Guest VMs are configured and assigned IP addresses by Fabric Controller so that these guest VMs can communicate with the outside world (Lemphers, 2008).

# Appendix D

Current cloud DBMSs, both NoSQL and RDBMSs, have many desirable features as seen in this study. But they also lack some of the features desirable in a database, that have been used by many users over the past many decades in RDBMSs. Attempts are always made to incorporate the best features of both NoSQL DBMSs and the traditional RDBMSs, so that we get a DBMS that is suitable for the cloud and yet has the rich features of RDBMSs. As a future study, a new model of cloud DBMS is considered .

Databases with a mixed approach or cloud hybrid databases could be very useful in the future. Such databases could be designed around the NoSQL storage models, with high availability, high scalability etc, but also have smaller amounts of data in RDBMSs as a part of the DBMS, to store data in a more structural and schematic manner. Users could store their small amounts of structured data in the relational side of the DBMS as and when needed and yet achieve scalability for the rest of the large amounts of data. This would definitely require less database management overhead when compared to large amounts of data in cloud RDBMSs. Complexity would also be less since only small amounts of data have to be maintained, replicated and synchronised within the RDBMSs and with such small datasets providing good querying abilities would not be too complex either.

Bibliography

# Bibliography

Amazon, 2006. *Amazon Web Services Launches Amazon S3*, Press Release

Anderson, B. 2010. *Dynamo and CouchDB Clusters.* [Online]. Available: http://blog.cloudant.com/dynamo-and-couchdb-clusters [17-Dec-2010]

Apache. 2010a. *HBase Architecture.* [Online]. Available: http://wiki.apache.org/hadoop/Hbase/HbaseArchitecture [18-Dec-2010]

Apache. 2010b. *ZooKeeper*. [Online]. Available: http://wiki.apache.org/hadoop/ZooKeeper [18-Dec-2010]

APC. 2010. *Azure Skies Ahead for Microsoft*.

Bain, T. 2009. *Is the Relational Database Doomed?* [Online]. Available: http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php [4-01-2011]

Bime. 2008. *What’s the Difference Between SaaS and the cloud?*  [Online]. Available:http://bimehq.com/cloud-computing/difference-saas-cloud/. [17-Oct-2010]

Burton, K. 2008. *Google, Bigtable, Compression, Zippy and BMDiff.* [Online]. Available: http://feedblog.org/2008/10/12/Google-Bigtable-compression-zippy-and-bmdiff/ [15-01-2011]

Campbell, D., Kakivaya, G., Ellis, N. 2010. Extreme Scale with Full SQL Language Support in Microsoft SQL Azure. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data,* Indianapolis, 6-10 June 2010, pp. 1021-1023. ACM.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., Gruber, E. 2006a. Bigtable: A Distributed Storage System for Structured Data, *Proceedings of the 7th Symposium on Operating System and Designs and Implementation*, Seattle, WA, 6-8 November 2006., pp. 205-218 ACM

Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., Gruber, E. 2006b. *Google Bigtable* [Online] Available: http://www.slideshare.net/nextlib/Bigtable [15-01-2011]

Ciurana, E. 2010. *Getting Started with NoSQL and Data Scalability*. [Online]. Available: http://refcardz.dzone.com/refcardz/getting-started-noSQL-and-data [05-01-2011]

Clarke, L. 2010. *Base Jumping*. [Online]. Available: http://iq.insight.com/relational-databases/ [03-01-2010]

Cloud Computing Defined.2010. *Cloud Computing Definition*. [Online].Available: http://www.cloudcomputingdefined.com/ [19-Oct-2010]

Cooper, B., 2010. The Prickly Side of Building Clouds. *Internet Computing, IEEE*, 14(6):64-67.

Davies, K., 2009. Taking Next-Generation Sequencing Data to the Cloud. *Bio-IT World*, 8(5):23-24.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. 2007. Dynamo: Amazon’s Highly Available Key-value Store. *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, Stevenson, WA, 14-17 October 2007, pp 205-220, ACM

Denman, 2010. *Graph Database vs. Relational Database* [Online]. Available: http://searchsoa.techtarget.com/tip/Graph-database-vs-relational-database [03-Feb-2011]

DeWitt, D., Madden, S., Stonebraker, M. n.d. *How to build a High-Performance Data Warehouse*. [Online] Available: http://db.csail.mit.edu/madden/high\_perf.pdf [23-Nov-2010].

Digg, 2010. *About Digg* [Online] Available: http://about.digg.com/ [3-Dec-2010]

Dimitrov, M. 2010. *NoSQL Databases*. [Online]. Available: http://www.slideshare.net/marin\_dimitrov/nosql-databases-3584443?src=related\_normal&rel=1786870 [16-Dec-2010]

Ebizq.2008. *Distinguishing cloud Computing from Utility Computing*. [Online]. Available:http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing\_cloud\_computing/. [19-Oct-2010]

Eure, I. 2009. *Looking to the future with Cassandra*. [Online]. Available: http://about.digg.com/blog/looking-future-cassandra [1-Dec-2010].

Gens, F. 2008. *IT cloud Services Forecast – 2008 2012: A Key Driver of New Growth*, http: //blogs.

idc.com/ie/?p=224

George, L., 2009. *HBase Architecture 101 – Storage*. [Online]. Available: http://www.larsgeorge.com/2009/10/ hbase-architecture-101-storage.html [17-Dec-2010]

George, L., 2010. *HBase Architecture 101 – Write-ahead-Log*. [Online]. Available: http://www.larsgeorge.com/2010/01/hbase-architecture-101-write-ahead-log.html [17-Dec-2010]

GigaOm.2010. *Show Me the Gateway — Taking Storage to the cloud*. [Online] Available: http://gigaom.com/2010/06/22/show-methe-gateway-taking-storage-to-the-cloud/ [27-Oct-2010]

Google .2009. *Google Fusion Table*. [Online] Available: http://googlesystem.blogspot.com/2009/06/google-fusion-tables.html, 2009, [1-Nov-2010]

Gray, K., 2010. *Why we love HBase*. [Online]. Available: http://www.stumbleupon.com/sublog/why-we-love-hbase/ [20-Dec-2010]

Grinev, M. 2010. *A Quick Introduction to the Cassandra Data Model* [Online].Available: http://maxgrinev.com/2010/07/09/a-quick-introduction-to-the-cassandra-data-model/ [4-Dec-2010]

Gunda, M., 2010. *Introduction to Apache Cassandra*. [Online]. Available: http://malliktalksjava.wordpress.com/2010/08/17/introduction-to-apache-cassandra/ [2-Dec-2010]

GuruMurthy, S., 2009. Architecting Storage for the Cloud Computing Era. *Micro. IEEE*, 29(6):68-71.

Hakala, D. 2009. *What is Google’s Bigtable system?* [Online]. Available: http://www.itmanagement.com/features/google-bigtable-073108/ [14-01-2011]

Henry. 2008. *Consistency and Availability in Amazon’s Dynamo*. [Online]. Available: http://the-paper-trail.org/blog/?p=51 [11-Dec-2010]

Hitchcock, A., 2005. *Google’s Bigtable*. [Online]. Available: http://andrewhitchcock.org/?post=214 [12-01-2011]

Hoff, T. 2009. *Why* *are Facebook, Digg and Twitter so hard to scale?* [Online]. Available: http://highscalability.com/blog/2009/10/13/why-are-facebook-digg-and-twitter-so-hard-to-scale.html [2-Dec-2010]

Hogan, M. 2008. *Cloud Computing & Databases*. [Online] Available: http://www.scaledb.com/pdfs/CloudComputingDaaS.pdf. [15-Nov-2010].

IBM.n.d. *Cloud Computing*. [Online]. Available: http://www.ibm.com/ibm/cloud/ [13-Nov-2010]

IDC, 2008. *Defining "cloud Services" and "cloud Computing"*. [Online]. Available: http://blogs.idc.com/ie/?

p=190. [25-Oct-2010].

IDC. 2008b. *The Diverse and Exploding Digital Universe*. [Online] Available: http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf. [19-Nov-2010].

Iskold, A. 2007. Amazon *Dynamo: The Next Generation of Virtual Distributed Storage* [Online]. Available: http://www.readwriteweb.com/archives/amazon\_dynamo.php [10-Dec-2010]

Jackson, J. 2008. *Cloud Computing leaving Relational Databases Behind*. [Online]. Available: http://reddevnews.com/articles/2008/09/22/cloud-computing-leaving-relational-databases-behind.aspx [02-01-2011]

Jain, A. 2010. *Key-Value store vs. Relational Database in Cloud context*. [Online]. Available: http://www.infosysblogs.com/cloudcomputing/2010/05/k-v\_store\_vs\_relational\_database\_in\_cloud\_context.html [05-01-2011]

Kennedy, N. 2009. *The anatomy of cloud computing*. [Online] available: http://www.niallkennedy.com/blog/2009/03/cloud-computing-stack.html. [18-Nov-2010].

Lemphers, D. 2008. *Windows Azure- What happens in the Data Centre?* [Online] Available: http://blogs.msdn.com/b/davidlem/archive/2008/12/16/windows-azure-what-happens-in-the-data-center.aspx [30-Nov-2010]

Lesem, S. 2009. *Understanding Cloud Storage APIs*. [Online] Available: http://cloudstoragestrategy.com/2009/11/understanding-cloud-storage-apis.html. [16-Nov-2010].

Linthicum, D. 2010. *SQL and Relational Databases: They are not right for the cloud*. [Online]. Available: http://www.infoworld.com/d/cloud-computing/SQL-and-relational-databases-theyre-not-right-the-cloud-794 [2-01-2011]

Lipcon, T. 2010. *Facebook Messaging- HBase comes of Age*. [Online]. Available: http://facility9.com/2010/11/18/facebook-messaging-hbase-comes-of-age#comment-2194 [21-Dec-2010]

Mateljan, V., Cisic, D., Ogrizovic, D. 2010. Cloud Database-as-a-Service (DaaS) – ROI. *Proceedings of the 33rd International Convention*, Croatia, 24-28 May 2010, pp.  1185 – 1188. MIPRO.

Merkle, R. 1987. A Digital Signature Based on a Conventional Encryption Function," *Proceedings of Crypto '87*, pp. 369-378.

Michalis, A. 2009. *Bigtable: A Distributed Storage System for Structured Data.* [Online]. Available: http://www.slideshare.net/adorepump/Bigtable-a-distributed-storage-system-for-structured-data [11-01-2011]

Microsoft. 2009. *Microsoft* *Case Study: Windows Azure* [Online]. Available: http://www.microsoft.com/casestudies/ [5-Dec-2010].

Microsoft. 2010a. *Scaling out with SQL Azure*.

Microsoft. 2010b. *SQL Azure –Pricing* [Online]. Available: http://www.microsoft.com/windowsazure/pricing/#SQL [25-01-2011]

Microsoft. 2010c. *Microsoft SQL Azure Database: A Scalable, self-managing database service*. [Online] Available: http://www.microsoft.com/en-us/sqlazure/database.aspx. [29-Nov-2010]

Microsoft.N.d.*Cloud Solutions*. [Online]. Available: http://www.microsoft.com/en-us/cloud/cloudpowersolutions/development-andhosting.aspx [13-Nov-2010]

MSCERTS. 2010. *SQL Azure –Security Access Control*. [Online]. Available: http://mscerts.net/SQL\_server/SQL%20azure%20%20%20security%20-%20access%20control.aspx [20-01-2011]

MSDN.2011a. *Data Types SQL Azure Database*. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee336233.aspx [27-01-2011]

MSDN.2011b. *Getting Started with the geometry Data Type*. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb895270.aspx [27-01-2011]

Murphy, I. 2010.Why Databases will follow applications into the cloud. *Computer Weekly, ProQuest Computing*, 22-28 June: 14.

Muthukkarappan, K. 2010. *The underlying Technology of Messages*. [Online]. Available: http://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919 [21-Dec-2010]

NOSQL. n.d. *List of NoSQl Databases*. [Online] Available: http://nosql-database.org/impressum.html. [28-Nov-2010]

Nubifer.2010.*Scaling Storage and Analysis of Data Using Distributed Data Grids*. [Online]. Available: http:// nubifer.wordpress.com/ 2010/02/16/ scaling-storage-and-analysis-of-data-using-distributed-data-grids/. [21-Oct-2010]

Obasanjo, D. 2009 *Building* *Scalable Databases: Denormalization, the NoSQL Movement and Digg* [Online]. Available:http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabasesDenormalizationTheNoSQLMovementAndDigg.aspx [3-Dec-2010].

Ohlhorst, F.2010. *Making Sense of Database as a Service and Cloud Database Technologies*. [Online] Available: http://searchdatamanagement.techtarget.com/news/2240024819/Making-sense-of-Database-as-a-Service-and-cloud-database-technologies. [29-Nov-2010].

Perham, M. 2010a. *Cassandra Internals – Writing*. [Online]. Available: http://www.mikeperham.com/2010/03/13/cassandra-internals-writing/ [5-Dec-2010].

Perham, M. 2010b. *Cassandra Internals – Reading*. [Online]. Available: http://www.mikeperham.com/2010/03/13/cassandra-internals-reading/ [5-Dec-2010].

Peschka, J. 2010. *Facebook Messaging- HBase comes of Age*. [Online]. Available: http://facility9.com/2010/11/18/facebook-messaging-hbase-comes-of-age [21-Dec-2010]

Quinn, J. 2010. *Saying Yes to NoSQL; Going Steady with Cassandra* [Online]. Available: http://about.digg.com/blog/saying-yes-nosql-going-steady-cassandra/ [4-Dec-2010]

Rahien, A. 2010. *That NoSQL Thing – Document Databases*. [Online]. Available: http://ayende.com/Blog/archive/2010/04/11/that-no-sql-thing-ndash-document-databases.aspx [09-Feb-2010]

Rawson, R., 2009. *HBase*. [Online]. Available: http://static.last.fm/johan/nosql-20090611/hbase\_nosql.pdf [18-Dec-2010]

Readwriteweb. 2009. *Is the Relational Database doomed?* [Online]. Available: http:/ /www.readwriteweb.com/ enterprise/2009/02/is-the-relational-database-doomed.php [22-Oct-2010]

Sarkissian, A. 2009. *An Intro into the Cassandra Data Model.* [Online]. Available: http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data-model [7-Dec-2010]

Sharma, J. 2009. *Dynamo: A Flawed Architecture – Part I.* [Online]. Available: http://jsensarma.com/blog/2009/11/01/dynamo-a-flawed-architecture-part-i/ [5-Dec-2010]

SNIA, OGF. 2009. *Cloud Storage for Cloud Computing*.

SQLAzure.2011*. Interacting with a SQL Azure Database*. [Online]. Available: http://www.youtube.com/watch?v=9KPvZ38e6R0 [10-02-2011]

Stonebraker, M., Hachem, N., and Helland, P. 2007. The End of an Architectural Era. *Proceedings of the 33rd international conference on Very large databases.* Vienna, Austria. 23-28 October 2007. ACM

Strickland, J. 2010. *How Cloud Storage Works*. [Online] Available:http://communication.howstuffworks.come/cloud-storage.htm. [17-Nov-2010].

StumbleUpon, 2010. *About StumbleUpon*. [Online]. Available: http://www.stumbleupon.com/aboutus/ [20-Dec-2010]

Theappgap. 2008. *Cloud Computing: Pros and Cons*. [Online]. Available:http://www.theappgap.com/cloud-computing-pros-and-cons.html [12-Nov-2010]

Vogels, W. 2009. *Expanding the Cloud: the Amazon Relational Database Service*. [Online]. Available: http://www.allthingsdistributed.com/2009/10/amazon\_relational\_database\_service.html [03-01-2011]

Wilkes, I. 2010. *Cloud Storage in post-SQL world.* [Online]. Available: http://arstechnica.com/business/data-centers/2010/02/-since-the-rise-of.ars [6-01-2011]

Williams, D. 2010. *HBase vs. Cassandra: Why we moved* [Online].Available: http://ria101.wordpress.com/2010/02/24/hbase-vs-cassandra-why-we-moved/ [4-Dec-2010]

Wu, J., Ping, L., Ge, X., Wang, Y., Fu, J. 2010. Cloud Storage as the Infrastructure of Cloud Computing. *Proceedings of the 2010 International Conference on Intelligent Computing and Cognitive Informatics*, Kuala Lumpur, 22-23 June 2010, pp. 380 – 383. IEEE.

Zaki Sabbagh. 2008. *Cloud Computing/Saas Case study*. [Online]. Available: http:// www.iirpresentations.com/A1090/pdf/D1-12.00-ZakiSabbagh.pdf [19-Ocy-2010]

Zdnet. 2010. *Nasuni cloud Storage Gateway.* [Online] Available:http://www.zdnet.com/blog/virtualization/nasuni-cloud-storagegateway/

1983 [15-Oct-2010]