Otto-von-Guericke-University Magdeburg

Faculty for Computer Science
Department of Technical and Business Information Systems

Master Thesis

A Survey and Classification of Data Management Research Approaches in the Cloud

Author:

Siba Mohammad

September 11, 2011

Supervisor:

Prof. Dr. rer. nat. habil. Gunter Saake,
Dr. -Ing. Eike Schallen

University Magdeburg
Faculty for Computer Science
P.O.Box 4120, D–39016 Magdeburg
Germany

Siba, Mohammad:
A Survey and Classification of Data Management Research Approaches in the Cloud
Master Thesis, Otto von Guericke University Magdeburg, 2011.

Acknowledgments

I want to thank my supervisor Prof. Gunter Saake for giving me the chance to work with the database group. I want to thank my supervisor Dr. Eike Schallehn for his advice and helpful feedback. I am very thankful for the time he spent reading my drafts. Many thanks to my colleagues Azeem Lodhi and Maik Mory for all the discussions and suggestions. Last but not least, many thanks to my lovely family and friends for their support and encouragement during this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

ACID	Atomicity Consistency Isolation Durability
ACL	Access Control List
AGATA	Advanced Gamma Tracking Array
AMP	Access Module Processor
AWS	Amazon Web Services
BASE	Basically Available Soft-state Eventual consistent
BOOM	Berkeley Orders Of Magnitude project
CAP	Consistency, Availability, and Partition tolerance
CLI	Call Level Interface
CQL	Cassandra Query Language
DAG	Directed Acyclic Graph
DB	Database
DBMS	Database Management System
DDL	Data Definition Language
DCL	Data Control Language
DFS	Distributed File System
DML	Data Manipulation Language
GFS	Google File System
HDFS	Hadoop Distributed File System
HiveQL	Hive Query Language
IAM	Amazon Web Services (**AWS**) Identity and Access Management

IFP Interface processor

JSON JavaScript Object Notation

MVCC Multi-version Concurrency Control

OLTP On line Transactional Processing

PDE Parallel Database Extension

PE Parsing Engine

OQL Object Query Language

QL Query Language

RDB Relational Database

RDBMS Relational Database Management System

RDS Relational Database Service

RM Relational model

RRS Reduced Redundancy Storage

S3 Simple Storage Service

SCLA Strongly Consistent Loosely Available

SLA Service Level Agreement

SMS SQL to MapReduce to SQL

SQL Structured Query language

TCL Transaction Control

TDP Teradata Director Program

UD User Defined

XML Extensible Markup Language

YMB Yahoo! Message Broker

ZK Zookeeper

Chapter 1

Introduction

Cloud computing is a new technology that promises to change the IT world by providing resources as an elastic pool of services in a pay-as-you-go model. Just like the electric grid freed corporations from worrying about generating their own electricity. The cloud promises to free corporations from worrying about their IT resources to focus on their business logic. Whether it is storage space, computational power, or software delivery, corporations can get these resources over the network from one of the cloud providers such as Amazon, and Google.

To meet the storage needs of cloud applications, new data management systems have been developed. Design decisions were made by analyzing the applications workloads and technical environment. It was realized that traditional DBMS with their centralized architecture, strong consistency, and relational model do not fit well. New data models such as key-value store with its variations of row oriented, document oriented, and wide column are used widely in the cloud. Different architectures with a variety of data partitioning schemes and replica placement strategies are developed.

1.1 Motivation

This thesis investigates data storage approaches in the cloud. There are some scientific research publications that highlights certain approaches. However, there are only very few publications that provide a survey of these approaches. This thesis aims at classifying, comparing, and summarizing data management research approaches in the cloud.

1.2 Structure

This thesis will continue as follows:

- Chapter 2 Foundations and Related Topics: Describes the basics of DBMSs and goes into details of traditional relational DBMSs and Parallel DBMSs. It also introduces cloud computing basics such as design principles, deployment models and enabling technologies.

- Chapter 3 Amazon Storage for the Cloud: Examines the Amazon cloud data management. It starts with Amazon's storage building block: Dynamo. Then goes into more details of the data model, query model, consistency model, and access control of S3, SimpleDB, and RDS.
- Chapter 4 Google Storage for the Cloud: Examines the Google cloud data management. It starts with Google's storage building block: GFS. Then goes into more details of the data model, query model, consistency model, and access control of Bigtable and Google Storage for Developers.
- Chapter 5 Hadoop Storage for the Cloud: Examines Hadoop Framework data management. It starts with Hadoop's building block: HDFS. Then goes into more details of the data model, query model, consistency model, and access control of Hive, HadoopDB, and HBase.
- Chapter 6 Yahoo!'s PNUTS: Examines the system architecture of PNUTS and goes in details of its data model, query model, and consistency model.
- Chapter 7 Cassandra: Examines the system architecture of Cassandra and goes in details of its data model, query model, consistency model, and access control.
- Chapter 8 CouchDB: Examines the system architecture of CouchDB and goes in details of its data model, query model, consistency model, and access control.
- Chapter 9 Comparison and Classification of Current Approaches: Compares the examined approaches and provides a classification based on architecture, partitioning, replication, data model, CAP, and query model.
- Chapter 10 Conclusion: Summarizes the results and gives an outlook of important research directions.

Chapter 2

Foundations and Related Topics

In this chapter, we provide the background for the rest of the thesis. We describe traditional and parallel DBMS. Then we introduce the basics of cloud computing.

2.1 Database Systems

In this section we introduce the requirements of a DBMS. We go into details of data model, consistency model, query model, and access control. Then, we examine some aspects of parallel DBMS which shares some aspects with cloud storage systems.

2.1.1 Traditional DBMS

A DBMS is a suite of software designed to manage databases and run operations on the data that are requested by several clients. A DBMS is expected to be able to provide the following [JDU97]:

- A DBMS should allow users to create new databases. Users define the logical structure (database schema) of the the data that will be stored using a data definition language.
- A DBMS allows users to modify and query the data using a query language or data manipulation language.
- A DBMS is capable of managing large amounts of data while keeping the efficiency of querying and manipulations.
- A DBMS supports durable storage of data. This means recovery from failures and misuse cases.
- A DBMS controls access to data from multiple users with different privileges. It should prevent unexpected interactions between users (isolation) and uncompleted actions on data (atomicity).

Based on these requirements we will investigate the following aspects of a DBMS:

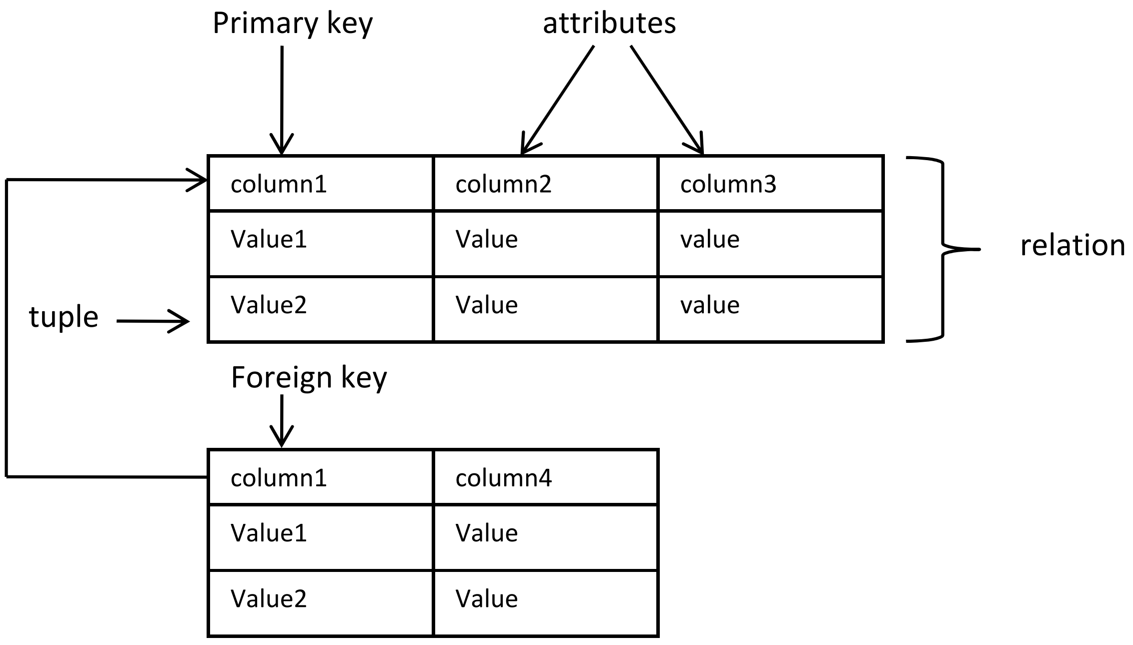

Figure 2.1: Relational data model

Data Model: A data model is a model that describes in an abstract way how data is represented in an information system or in a DBMS. Choosing the data model has a fundamental effect on the other aspects of a database system like the integrity constrains, and data access. Next we list the most important data models:

- Hierarchical model
- Network model
- Relational model (RM)
- Non-first normal form (NF2 and eNF2)
- Object-relational model (ORM)
- Object-oriented database model (OODM)
- Semi-structured and Extensible Markup Language (XML) model

The most commonly used data model is RM. It was developed for classic database applications such as banking systems, airlines reservations, and sales/customers relations. It was implemented by major DBMS like Oracle, IBM DB2, MS SQL, PostgreSQL, etc. In this model, Data is organized in tables(relations) of records(tuples) with columns(attributes). See figure 2.1. A table can have a primary key which is the unique identifier of rows. A primary key can be referenced from another table as a foreign key and forces integrity constrains on the data.

Consistency Model: Consistency ensures that only valid data is written to a database and that transactions transform data from one consistent state to another. There are different consistency models that satisfy the needs of different database applications. Atomicity Consistency Isolation Durability (ACID) properties means strong consistency and are usually implemented in relational databases. With ACID, all users have the same consistent view of data before and after transactions. A lock manager makes sure that data is locked while being modified by a transaction. If another transaction tries to access data while data is locked, it has to wait. Multi-version Concurrency Control (MVCC) is a replacement for lock-based systems and is implemented by some relational databases and most distributed databases. MVCC allows several processes to access data in parallel even if one is modifying the data. Consistency is maintained using some kind of timestamp. Eventual consistency is supported by web scale data management Basically Available Soft-state Eventual consistent (BASE) properties. Eventual consistency does not guarantee that all users see the same version of data item but guarantees that all of them see data (no locking).

Query Model: Data is queried using several languages depending on the data model: Structured Query language (SQL) for the RM, Object Query Language (OQL) for the Object model, XQuery for the XML model. Most DBMSs have sophisticated user interfaces, ad-hoc query support, and JDBC drivers for several languages. The most used query language SQL is based of the operators of relational algebra and is divided to Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), and Transaction Control (TCL). Next, the select statement, is an example of SQL statements:

```
SELECT [ALL | DISTINCT] column1[,column2]

FROM table1[,table2]

[WHERE "conditions"]

[GROUP BY "column-list"]

[HAVING "conditions]

[ORDER BY "column-list" [ASC | DESC] ]
```

Access Control: Database access control deals with controlling who has access to which data in the database. Most of databases have role-based access control model [RSRS98]. We list the most important aspects of this model in the following:

- Assignable privileges: a privilege defines the read, write, and modify rights on the whole database or parts of it (e.g. tables, views, and indices). Privileges can be assigned to a role, a user or group of users.
- User role assignment: initially a user is not assigned any role but can be assigned multiple roles. A role can also be assigned to another role or a group of users.
- Support for role relationships and constraints: the database admin as well as a user who is granted a role with a GRANT OPTION can grant the role to other users. This allows roles nesting.

2.1.2 Parallel Database Systems

Parallel database systems are database systems that parallelize various database operations such as data loading, index building, and query execution to improve performance. Parallel databases usually store data on different nodes. They are built on multi-processor architectures which classified into the following:

- Shared Memory Architecture: processors share direct access to one global memory and to all disks. This architecture is also called shared everything.
- Shared Disk Architecture: each processor has its own memory but can directly access all disks.
- Shared Nothing Architecture: each processor had its own memory and and disks.

Shard memory and shared disk architectures do not scale well for database applications. A major problem for both of them is interference and network traffic. Shared nothing architecture minimizes the need for interference by minimizing the shared resources and thus minimizing the transfered data. It is considered to communicate small amounts of data compared to the other two architecture [DG92]. The shared nothing architecture communicates only questions and their answers. The advantage that this brings is the ability to scale to hundreds of processors that do not interfere. Examples of parallel DBMSs are Teradata, Gamma, Tandem, Oracle RAC, and IBM InfoSphere. Most parallel DBMSs use the RM. Each relation is horizontally partitioned and its tuples are distributed over several nodes of the database cluster. Parallel DBMS provide the following horizontal partitioning techniques [AS10]:

- Round-robin Partitioning: it is considered the simplest strategy. Tuples of a relation are distributed in a round-robin fashion among the nodes. It is not considered good for applications that do associative data access frequently.
- Hash Partitioning: tuples of a relation are distributed among disks by applying a hash function to one or more attributes of the relation. This strategy is good for applications with sequential and associative data access. It is implemented in Teradata, Bubba, and Gamma,etc.
- Range Partitioning: in this strategy, tuples of a relation are partitioned to ranges of key values. The partitioning is predefined or dynamic. It is implemented by Oracle, Bubba, and Gamma, etc.

Horizontal partitioning of tables is essential for scalable performance of SQL queries and leads to another important aspect of parallel databases which is parallel query processing. Relational queries are suitable for parallel processing with shared nothing architecture achieving near linear speedup for these queries and specially for On line Transactional Processing (OLTP) [DG92]. As an example we consider sales table partitioned using the round-robin strategy, where rows of a single customer will be distributed on several nodes of a database cluster. To execute a query, the paralle DBMS compiles it into a number of operator pipeline jobs. Then executes the query plan on all the nodes in parallel [SAD$^+$10]. There are two classes of query parallelizing:

- Intra-Query Parallelization: this is applied on the query level where different operators of the query are executed by different processors. Examples are independent and pipelining parallelization.
- intra-operator Parallelism: this kind is applied to each operator within a query. To execute an operator, multiple processors can be employed. Examples are selections from fragments, hash-based decomposition for join, etc.

Parallel DBMSs take into consideration the partitioning strategy that is applied on a table when executing the query. For instance if two tables of a join are hash partitioned on the joining attributes, the query optimizer will automatically and transparently to the user and applications delete the shuffle operator from the compiled query plan [SAD$^+$10].

Teradata: As an example of commercial parallel DBMS, we will examine Teradata shortly. Teradata is a Relational Database Management System (RDBMS) that supports large sizes of data up to hundreds of terabytes. It supports scalability on all database dimensions such as data size, number of users, and complexity of queries. Teradata was founded in the late seventies and is one of the pioneers to adopt the shared nothing architecture. Many other parallel database system use Teradata techniques of horizontally partitioning tables and partitioned query execution [SAD$^+$10]. Figure 2.2 illustrates the components of Teradata system. We describe the most important components in this architecture in the following [ter]:

Call Level Interface (CLI): It is the interface that receives quires and returns results. All SQL queries are transfered in the CLI packet format.

Teradata Director Program (TDP): It routes the packets to the specified Teradata DBMS sever for execution.

Parallel Database Extension (PDE): It is the interface layer on top of the operating system. It is responsible for providing the parallel environment, scheduling sessions, debugging, etc.

Parser Dispatcher: It is responsible for managing sessions, parsing SQL, and communicating with AMPs.

Teradata processors are divided in two groups based on their function: Interface processor (IFP)s and Access Module Processor (AMP)s. The interface processors handle communicating with the host and the access module processors handle partitioning.

As for data partitioning and query execution, Teradata proved to demonstrate near linear speedup and scaleup on relational storage and querying [DG92]. All relations of the database are hash partitioned over AMPs. When inserting a tuple in a relation, a hash function is applied on its key to decide, based on the result, which AMP will store the tuple. Next step, the chosen AMP performs a second hash function to decide where in the specified relation fragments should this tuple be inserted. For query execution in Teradata, joins are executed using a parallel merge-join algorithm rather than pipelined parallel execution.

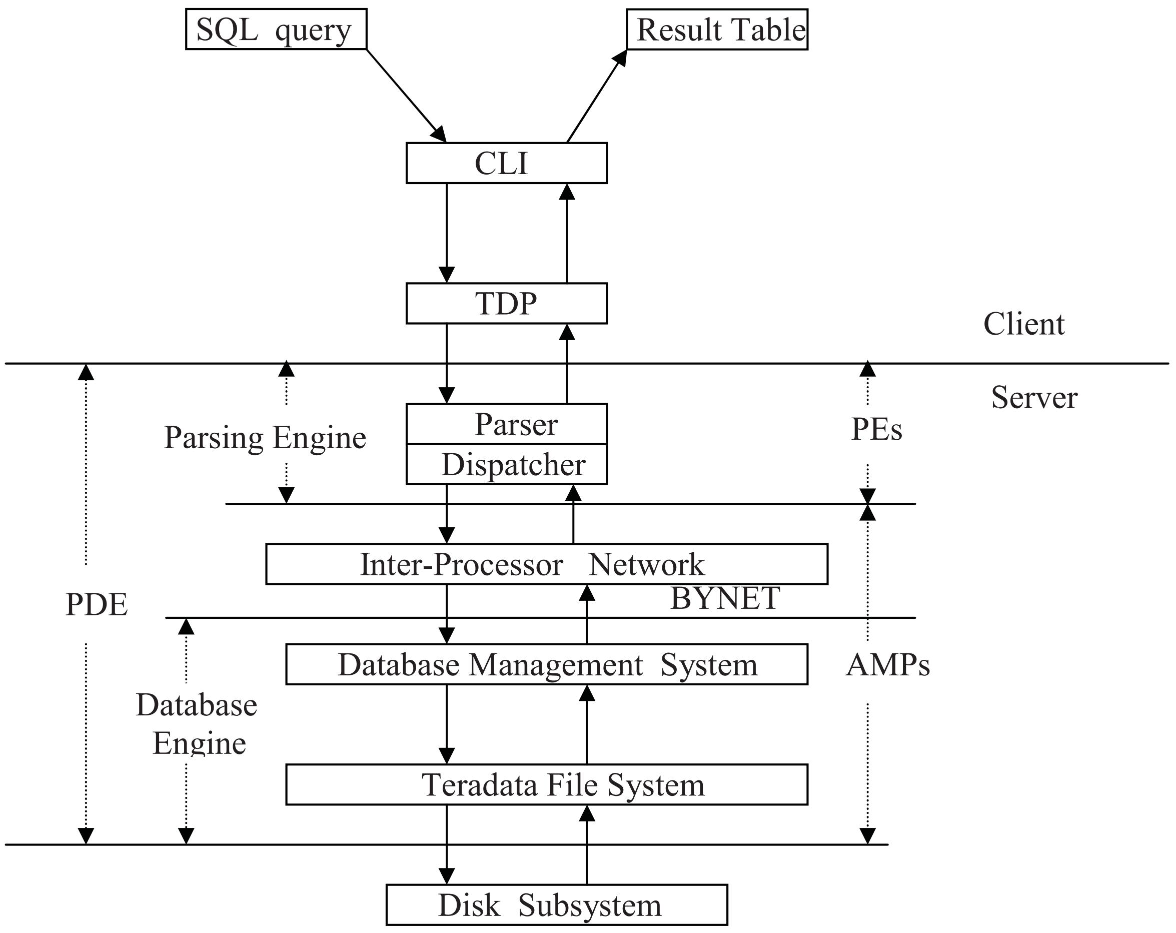

Figure 2.2: Teradata System Architecture adapted from [ter]

2.2 Cloud Computing Overview

There is not a widely accepted definition of cloud computing due to different reasons. One is the involvement of developers and engineers from different fields e.g. grid computing, software engineering and databases in cloud computing research, where each works on it from a different perspective. Another reason is that the technologies, that enable cloud computing such as Web 2.0 and service oriented computing, are still changing and developing. Cloud computing can be defined as a model for ubiquitous, convenient, on-demand network access to a shared pool of computing resources (infrastructure, applications, and platform) that can be provisioned and released with minimal management effort or service provider interaction [MG11]. Figure 2.3 shows the development of computing models from terminal/mainframes to cloud computing.

The essential characteristics and features of cloud computing are described in the following. First of all it must be an on-demand self-service, which means that users of the cloud can automatically provision resources with minimal or no human interference of the cloud service provider. Users access the cloud services via network deploying suitable techniques and protocols with the use of thick or thin clients. Another characteristic is resource pooling. It means that services of the cloud are pooled and serve many consumers using a multi-tenant model.

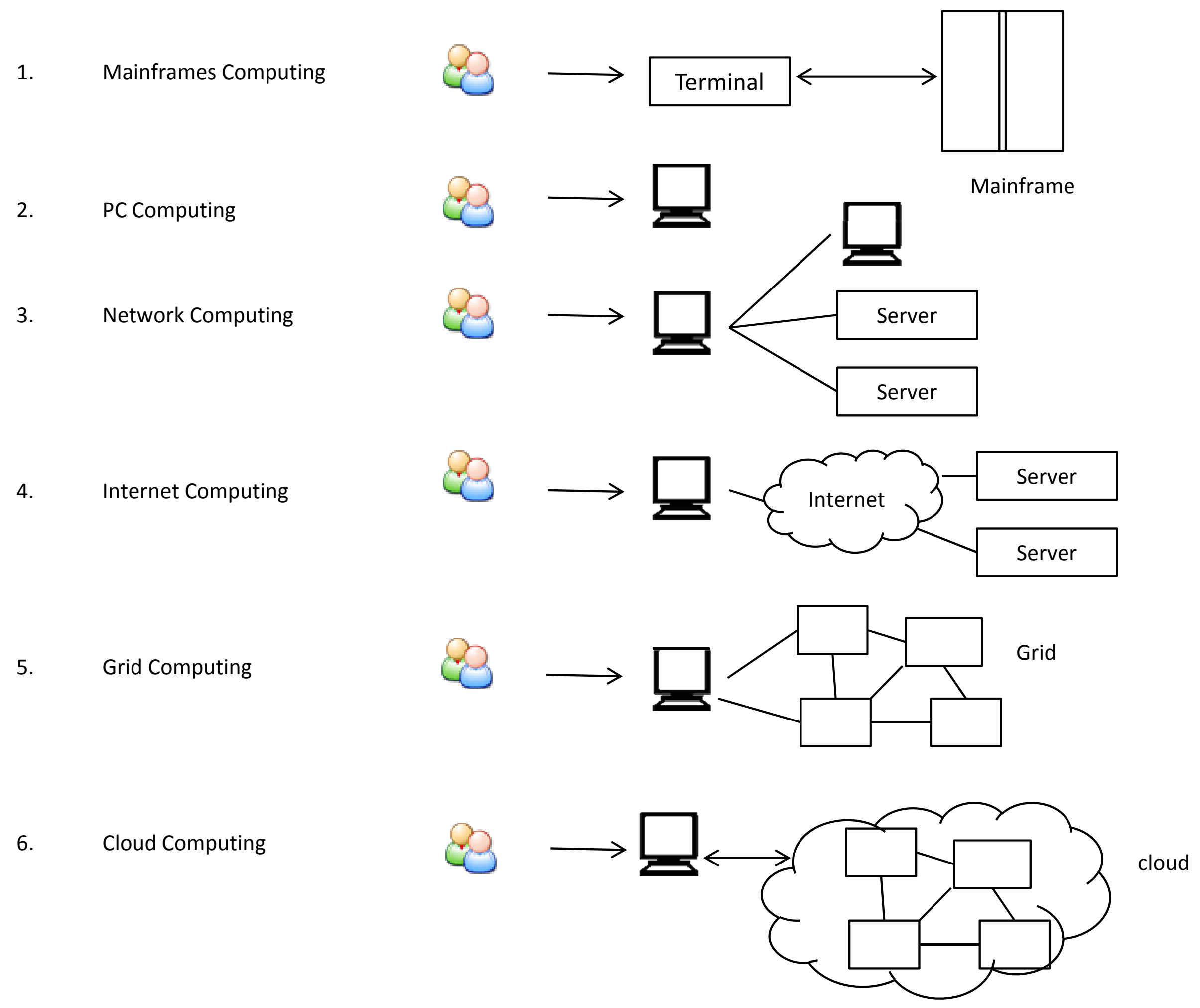

Figure 2.3: From mainframes to cloud computing adapted from [VZ09]

In the cloud, users usually do not know or control the location of the provided services or resources. This is called Location Independence. In some cases or on demand, users know the location on high level like city or data center. They can ask that their data (or meta data about their data) never leaves the EU cloud to US cloud for instance.

Resources should look infinite to the users and according to their needs, resources increase and decrease rapidly and in some cases automatically (rapid elasticity).

Monitoring service usage should be available for both provider and consumer. For example Amazon provides the CloudWatch service that monitors the AWS resources (EC2 and RDS database instances) and can be customized to monitor any metrics of users' applications.

Guaranteed Quality of Service (QoS) is very important in the cloud where users expect certain level of service that should be agreed on. Cloud computing still face many other challenges. Among them are security, privacy, and control issues.

2.2.1 Design Principles and Models of Cloud Computing

Cloud architectures and systems are implemented based on some basic principles that are described in the following [AS311, Hel07]:

Decompose item into small well-understood building blocks: One should not think of making a service that provides everything to everyone. For scalability and availability issues it is better to build small elements which can be used to make other services.

Decentralization: To get rid of any scaling bottlenecks or single points of failures, it is recommended to decentralize the used techniques in the cloud systems

Asynchrony: The system can progress in all situations.

Autonomy: Individual elements of the system make decisions based on local information.

Local responsibility: Elements are peers, each one of them has the responsibility of achieving its consistency.

Controlled concurrency: Activities are planned to make sure that only limited concurrency control is needed or in best case no control is needed at all.

Failure tolerant: Having failures of elements is considered to be the normal case. Operations must continue with minimal or no interruption.

Controlled parallelism: Performance can be enhanced and recovery can be more robust by introducing new elements. Parallelism makes that possible.

Simplicity: The system should be made as simple as possible (but not simpler).

In terms of who manages and owns the cloud services, we can divide its deployment models into the following:

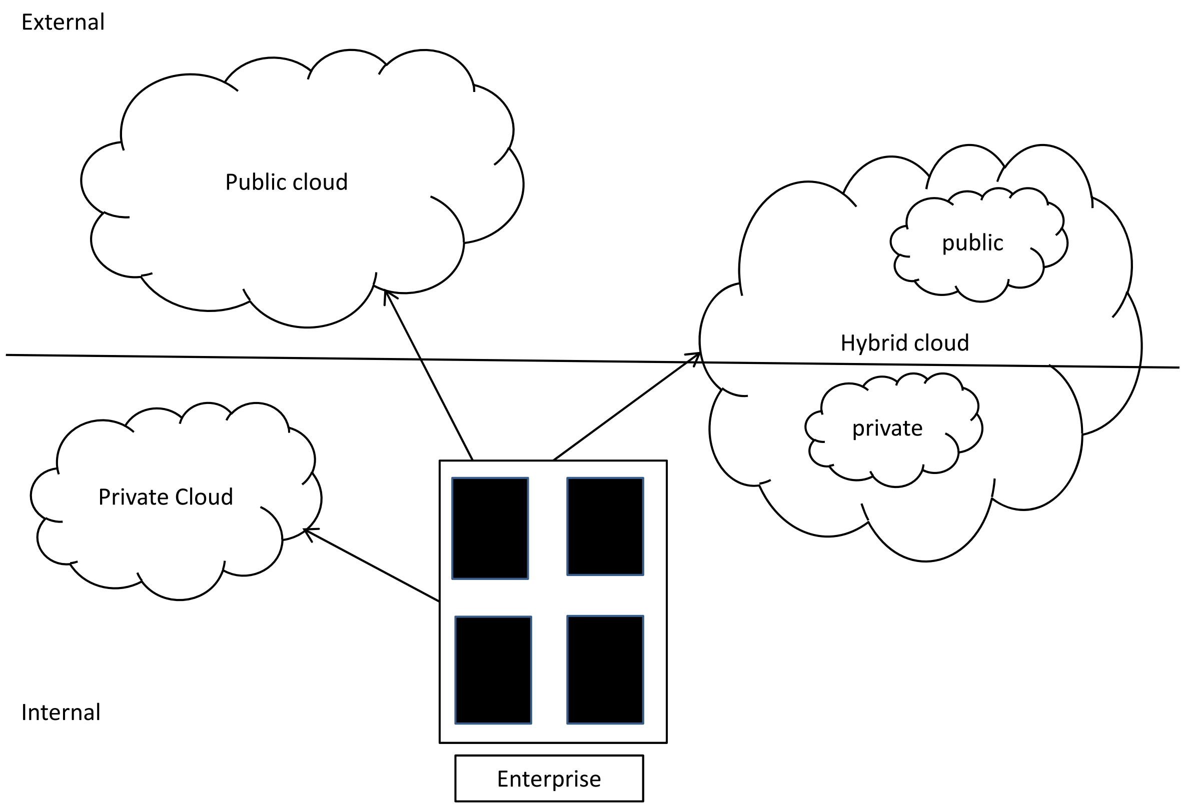

Figure 2.4: cloud computing deployment models adapted from [FE10]

Public clouds: This is the most popular deployment model of cloud computing. In this model, the cloud infrastructure and resources are owned by an enterprise that provide them to individuals or other enterprises in a pay-as -you-go model. Cloud resources are shared between many consumers. Leaders in the market providing cloud services of this model are Google and Amazon. They provide many options that allow users to get the resources with minimal cost and less management efforts. Major concerns are privacy, security, and data control.

Private clouds: Cloud infrastructure operates to serve one organization. Management of cloud is done by a third part or by the organization. This model usually attracts governments and organizations that prefer to keep data in a private environment.

Hybrid clouds: The cloud infrastructure is a combination of private and public clouds. Each of them will still be a single entity connected to another cloud. In this case enterprises can choose to store their data on the private part of the cloud.

Community clouds: Enterprises with the same needs share the cloud infrastructure. Cloud is managed by a third party or by the enterprises that share it.

Figure 2.4 illustrates the different deployment models.

In figure 2.5 we can see the cloud service models. The first model is Software as a Service (SaaS). In this model, thin clients (Internet browsers) are used to access software and applications that operate on the cloud. Users cannot control the underlying infrastructure on which these applications operate (operating systems, networks, servers). In

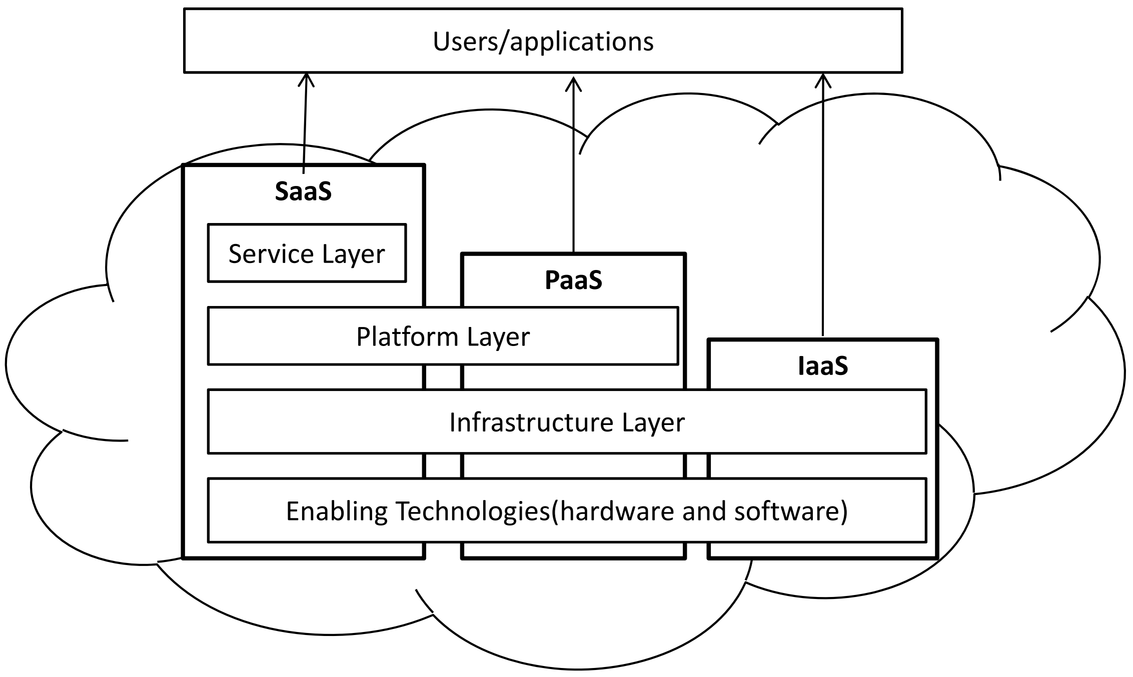

Figure 2.5: Cloud computing service models adapted from [FE10]

some cases, they can modify some application parameters and settings. Example of that is salesforce. The second service model is Platform as a Service (PaaS). The service provided in this case is the cloud platform and programming environment on which users can create and run thier own applications. Operating system, network components and hardware are controlled by the service provider not by users. Applications settings and related working environment properties are managed by users. Examples are Microsoft's Azure and Google's Apps Engine. The last service model is Infrastructure as a Service (IaaS): Users can use the cloud basic computing resources like storage, processing units and networking components. They can install operating systems and applications and control their properties. Some network properties like firewall components and load balancers can be can be controlled by the user but not the underlying cloud infrastructure. Examples are Amazon EC2 and Eucalyptus open source cloud computing systems.

2.2.2 Cloud Computing Enabling Technologies and Related Standards

What makes the cloud computing possible is different technologies that came together each with different standards. It is important to have an overall view of them to be able to understand how the cloud really works. It allows developers and enterprises to make informed decisions especially about security and privacy issues and be able to use the full potential of the cloud service models. Here, we list the most important technologies and related standards [FE10]:

Virtualization: It is the key technology behind cloud computing that allows the creation of an abstraction layer of the underlying cloud Infrastructure. Using virtualization, resources (hardware and software) can be shared and utilized while hiding the complexity from the cloud users.

Web services and Service Oriented Architecture (SOA): Cloud services are designed as web services following the standards of WSDl, SOAP and UDDI.

Web 2.0: The technology of using the World Wide Web to enhance information sharing and collaboration. This usually includes techniques such as CSS, semantic web technologies, mash up, and syndication.

Hypertext Transfer Protocol (HTTP): A protocol used for distributed, collaborative, hypermedia information systems. It is stateless but a stateful session can be created with the use of e.g. cookies that carry the state information in the requests and responses between the user agent and the server. HTTP authentication includes Basic access Authentication scheme which is considered not secure for user authentication where the user name and password are transferred over the network as clear text. To make this scheme more secure, external secure systems like SSL can be used. The second authentication scheme which is considered to be better is Digest Access Authentication where user authentication can be done without sending the password as clear text. The HTTP protocol is a request and response protocol. Client sends a request to the server in a protocol version, or URI, or a request method. The request is followed by a MIME like message that includes the client information, and the request modifiers. The response from the server is a status line that includes the message's protocol version and the success or error code. A MIME like message comes next with holding the server information, entity meta data and sometimes entity body content. A feature of HTTP is that it allows applications to be built regardless of the data being transferred. HTTP is used usually in the IaaS and SaaS layers in the cloud.

HTML5: Adopted by W3C but still is a work in progress. One aim behind this standard is to reduce the need for proprietary plug-in based technologies such as Adobe Flash, Microsoft Silver light, and Sun JavaFX. HTML5 will be the new standard for HTML, XHTML, and the HTML DOM. Another aim is allowing web applications to work off line HTML5 contains features to support this such as SQL-based database API, Off line application caching APIs, on line/off line events. HTML5 is used in the SaaS layer of the cloud.

XML and JavaScript Object Notation (JSON): XML is used to represent arbitrary data of web services. It has been used for data interchange over the Internet. However, it is not considered very well suited for the job since it does not match the data model of most programming languages [FE10]. Here comes JSON. It is considered more suitable than XML for different reasons. First of all it is easy to read and write by humans and easy to parse and generate by machines. It is language independent but uses convections that are familiar to people who work with C++, C#, Java, Python, Java Script, Perl. XML and JSON are used in the PaaS and SaaS layers of the cloud.

Asynchronous JavaScript and XML (AJAX): AJAX is implemented on the client side for creating interactive web applications. It allows them to exchange data with the web server without interfering with the display and the behavior of the existing web page. AJAX is used in the SaaS layer of the cloud.

Web Syndication: It is a form of syndication where website content is made available to multiple sites. Usually a summary of the website's recently added material is made available to other sites as web feeds. Types of content that are syndicated are full content, atom content, and RSS. Java script is usually used to for web syndication. Web syndication can be used in the SaaS layer

The Extensible Messaging and Presence Protocol (XMPP): The is an open technology used for real time communication, that powers applications like Instant Messaging(IM), multi-party chat, voice and video calls, content syndication, and generalized routing of XML data. New extensions are added all the time. XMPP can be used in the SaaS layer of the cloud

Representational State Transfer (REST): It is a style of software architecture for distributed hypermedia systems such as the World Wide Web. It is based on a set of principles that describe how networked resources are defined and addressed. Key goals of defining REST are scalability of component interactions,generality of interfaces,independent deployment of components,reduce latency,enforce security ,and encapsulate legacy systems. Rest can be used in the SaaS, PaaS, and IaaS layers of the cloud.

2.2.3 Security and Data Privacy Issues

A cloud computing environment is not necessarily less secure than other IT environments. However, security is the number one objection to cloud computing [AFG$^+$10]. With a new technology, come a long new challenges and risks that need to be addressed. Cloud users face threats from outside and inside the cloud. Attacks from outside the cloud are similar to those attacks for data centers. But the responsibility for protection depends on the cloud service model and service layer. Both service provider and users are responsible for security with different degrees. In the case of SaaS, security is supposed to be enforced and managed by the provider. In PaaS some security features are built in. But in most cases users need to implement an additional security layer to address their security needs . In the third service model, IaaS, operating systems, applications, and content are managed and secured by the cloud users [BM09]. In all cases, cloud providers are responsible for the physical security. A study that involved more than nine hundred IT professionals in Europe and the US shows that participants believed that the most difficult security issues to handle in the cloud are securing the physical location of data assets and restricting privileged user access to sensitive data [Pon10]. Attacks from inside the cloud can be theft and denial of service [AFG$^+$10]. The main defence technique is virtualization. However, the cloud users' careless use is a major cause for crucial security threats and vulnerabilities [Dar11]. Another security risk comes from the lack of organizational control over employees using cloud services [JG11]. To solve this, control measures of the organization can be extended into the cloud by using Trusted Computing (TC) and applied cryptography [CGJ$^+$09].

Another concern that organizations face in the cloud is Compliance [JG11]. Compliance is the conformance with standards and regulations. When data of an organization is stored in a different country, compliance becomes complicated and many concerns arise because different laws of privacy are forced in each country. The main compliance concerns in this case include whether the laws in the country where the data was collected permit the transfer of data to a different country, whether those laws continue to apply to the data after transfer, and whether the laws at the destination present additional risks or benefits.

2.2.4 Grid Computing vs. Cloud Computing

Grid computing is defined as coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing is necessarily highly controlled with resource providers and users defining carefully what is shared, who is allowed to share, and the conditions under which sharing occurs [IF01]. Grid computing started 10 years before cloud computing and many argue that cloud computing has developed from it [Mye09]. Both Grid and cloud computing promise to harness computing resources to provide users with their needs in a cost effective and timely manner. However, there are many differences between the two in the cost/business model, shared resources, providers and customers, usage patterns, and interaction models. Cloud and grid address different areas of distributed computing problems. The work load in grid computing is composite of distributed jobs that run on highly coupled nodes within a cluster. In the cloud, it is more loosely coupled jobs (e.g. map/reduce). Grid computing provides an easy interface to resources which users have the right to access such as in research and academic institutes. The participating organizations are both consumers and providers. They form a virtual organization under an agreement that specifies the shared resources and all regarded details. Shared resources can be under different administrative domains. Users or clients submit their requests of storage and application processing from their home domain. Requests are handled by a resource broker that selects a domain having the demanded resources. On the other hand, cloud computing provides pay-as-you-go usage of a pool of resources owned by a provider (e.g. Amazon, Google) and is mainly used in industry and businesses. Users become owners of virtual resources with guaranteed QoS. An example of the Cloud usage model is the Amazon EC2 where most clients are individual users, who demand the needed resources via Amazon EC2 API without cross organizations agreement or contract [FE10]. The execution environment in the cloud is virtualized, it can be cloned several times in the cloud. In the Grid case, the execution environment is library based and customized to hardware domains. Library consistency is hard to be ensured across different domains. A case study of the Advanced Gamma Tracking Array (AGATA) project focuses on the cost analysis of storage for their raw data in three cases [VMMS11]: dedicated storage, grid storage, and cloud storage (Amazon S3). The study shows a minimal overall cost for the grid option. While the cloud option shows a minimal cost for the Storage Total Cost of Ownership(TCO), its cost for data transfer is its killing point. It is two times compared to the grid. However by applying an intermediate data storage strategy [YYLC10], the cost can be reduced significantly. Cloud supporters argue that in the cloud we have infinite resources, on-demand scalability, ease of use, and guaranteed QoS. The grid is hard to use, with no guaranteed level of performance [BBG11] and predefined finite allocation of resources. Neither of the two technologies is better or can replace the other. Each is more suitable for a certain area of applications. Integrating them is an ongoing research topic [BBG11, JMF09, KFZ05]

Chapter 3

Amazon Storage for the Cloud

Amazon is considered to be the first industry leader to introduce the cloud term and provide cloud services. Amazon introduced a variety of storage services. For all of them, it promises high availability and low cost. In this chapter we will examine the following Amazon storage services: S3, SimpleDB, and RDS. First, we will examine Dynamo storage, which powers other Amazon services.

3.1 Dynamo

Dynamo is a scalable, highly available key value store developed by Amazon to internally power its web services such as shopping cart, customer preferences, the product catalog and SimpleDB. Dynamo is not intended to be used by the public [Vog11]. Designing dynamo is based on several requirements and assumptions of the targeted work scope. Next we discuss them shortly [DHJ$^+$07]:

Query model: Dynamo targets applications that work with small objects i.e. smaller than 1 MB. Operations are simple reads and writes that do not target more than one object at a time. So relational schema is not needed.

ACID properties: Dynamo targets applications that favor availability and tolerate low consistency. Dynamo does not guarantee any isolation and allows single key updates only.

Efficiency: Dynamo will work on a cluster of commodity hardware. Amazon services that will use Dynamo must be able to configure it so their latency, service level, and throughput requirement are met.

No security requirements: Since Dynamo is intended to be used internally, its surrounding environment is considered friendly. Thus no security related features such as authentication and authorization are needed

Scalability: Dynamo must be able to scale with minimal impact on its performance or on the performance of any service that is using it.

Symmetry and decentralization: Dynamo supports decentralized control techniques where all the nodes that compose it have the same responsibilities of request coordination, membership and failure detection, and data storage.

Heterogeneity: Dynamo will operate properly on heterogeneous infrastructure.

3.1.1 Architecture and Overview

The architecture of Dynamo is based on peer to peer systems. It is a ring of storage hosts called nodes with each one identified by a number that determines its position. Dynamo uses the concept of virtual nodes where each nodes is assigned multiple positions in the ring to tune the partitioning process. Data is automatically partitioned using consistent hashing. The advantage in consistent hashing is that its output is a circular space or a ring (the largest hash value raps around to the smallest hash value). Data is stored as objects and each object is identified by a key. MD5, a cryptographic consistent hash function, is applied to the key to determine its position on the ring. Then, the node on which the data item will be stored is determined by moving clockwise to find the first node on the ring with a position larger than the object's position. A node that is responsible for storing an object is called the coordinator node for that object. The coordinator is responsible for replicating an object N-1 times on other nodes where N is determined per instance. The list of nodes that store replicas of an object is called the preference list for that object. Every node in the system can determine which nodes should be in the preference list for any particular key [DHJ+07]. Any node in dynamo is qualified to receive clients' requests. However, there are two options for clients. First is to route the request using a generic load balancer with the advantage of not having to link any specific Dynamo node in the client's code. The second approach is using a partition-aware library that automatically routes the request to first appropriate node in the reference list with the advantage of lower latency [Vog11]. The system automatically uses Sloppy Quorum approach to handle node failure. An Anti-entropy replica synchronization protocol that uses Merkle trees is adopted by Dynamo to handel perminent failures.

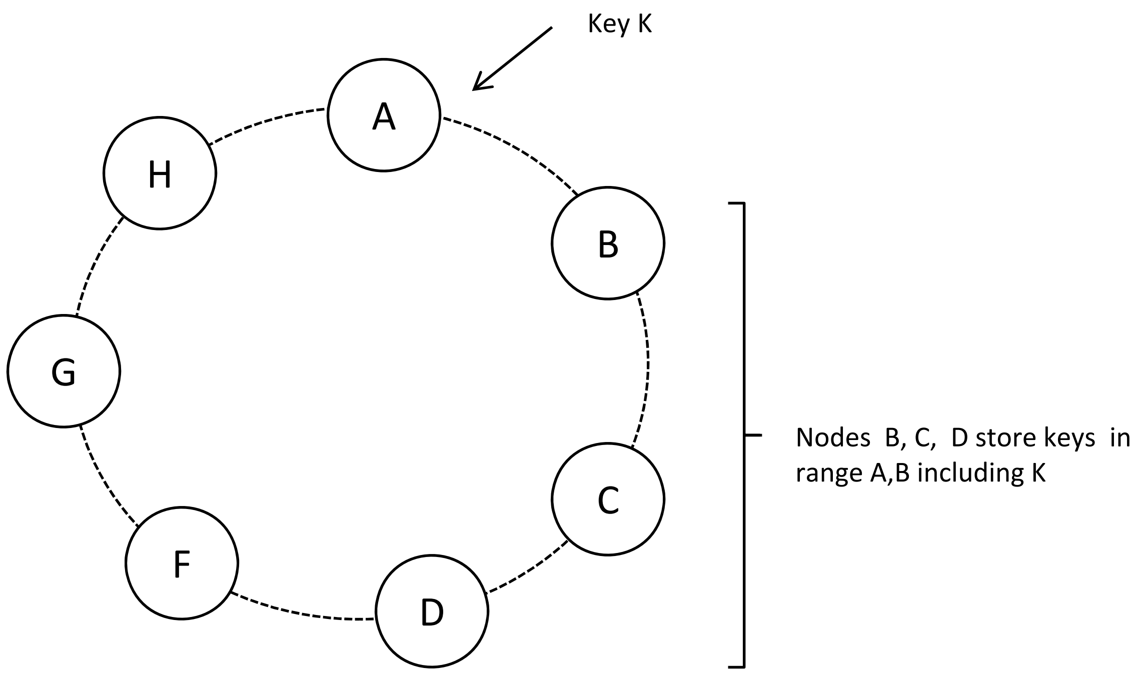

Figure 3.1: Dynamo Ring adapted from [Vog11]

3.1.2 Consistency Model

Dynamo supports weaker consistency to allow higher availability for write operations. Data is eventually consistent An update request can return a success before it has be been committed on all nodes. Read requests may return different versions of the same object. Vector clocks are used for conflict reconciliation. A vector clock is a list of pairs: node and counter that is associated with every version of the data object. Vector clock information is stored in what is called the object's context.

3.1.3 Query Model

Dynamo data is stored using its simple interface that allows two operations put(key, context, object) and get(key) [DHJ$^+$07]. The put(key, context, object) request writes an object associated with the specified key. The context information determines which version will be updated. The get(key) request returns an object or a list of objects with conflicting replicas and a context.

3.2 Amazon S3

Amazon S3 provides a simple web services interface that can be used to store and retrieve data. It gives developers access to the same highly scalable, reliable, secure, fast, inexpensive infrastructure that Amazon uses to run its own global network of web sites. Common use cases include data backup (used to store backup of Amazon RDS), software delivery (host software applications that customers can download) and media hosting (videos, photos, music uploads and downloads e.g. Imagenet: uses S3 to store medical image data) [Ado11b, PC10].

3.2.1 Data Model

Object is the fundamental entity that can be stored in S3. It contains the object data and its metadata. Metadata is a set of name-value pairs that describes the object. There are some default system metadata pairs such as the time the object is stored, the date of last modification of the object, data type and data length. A Bucket is a container for objects. Each object must be contained in one Bucket. A bucket can be configured so that each time an object is added to it, a unique version ID is assigned to the object. So an object is uniquely identified within a bucket by a key (name) and the version ID. Objects can be addressed by a combination of bucket name, key, and optionally version ID. To optimize latency and minimize the cost and for compliance issues, Amazon offers the option to choose the geographical region where buckets are stored (US East, US West-Northern California, EU-Ireland, Asia Pacific-Singapore, Asia Pacific-Tokyo). Each of these regions offers different consistency models as we will explain later. Objects stored in a region and metadata about them never leave the region unless by explicit transfer made by the owner. Folders can be used to organize objects within buckets. They are not part of the S3 API but are available in the AWS management console. Objects are redundantly stored on multiple devices in different facilities of one S3 region and the PUT object operation synchronously stores data across multiple facilities before sending

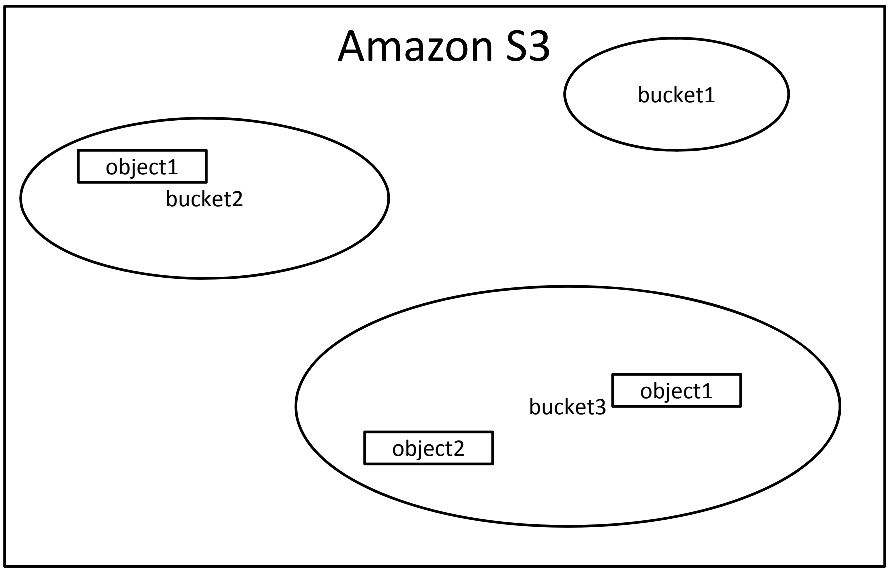

Figure 3.2: S3 data model adapted from [PC10]

back a success message. S3 also provides Reduced Redundancy Storage (RRS). Choosing this option means data will be stored at a lower level of redundancy than Amazons S3 standard storage. It can be used for storing non critical data to reduce the cost. However amazon claims it still provides 99.99% durability of objects over a year and 400 times the durability of a typical disk drive. Integrity is verified using checksums. Amazon S3 regularly verifies integrity and repairs any corrupted data. Checksums are also calculated on all network traffic to detect corrupted data packets when storing or retrieving data.

3.2.2 Consistency Model

The write operation is atomic on the object level. It is not possible to make atomic write operations on several keys. There is no locking mechanism and in the case of concurrent write operations the one with the last time stamp wins. Amazon uses different methods to achieve high availability that involves replicating data across multiple servers within Amazon's datacenters. Getting a success message means data is stored. But since changes might not be immediately replicated across different copies, a subsequent read might return old data. Corrupted or partial data is never written and will not be returned. In Amazon S3, when a process writes a new object and immediately tries to read it or lists the keys of the same bucket or folder, a "key does not exist" error might appear or the new object might not be listed. And it is the same for deleting a key (it will still be listed and accessed by processes until the change has been propagated to all replicas). Read after write consistency ensures the immediate visibility of new objects. The US Region provides eventual consistency for all requests. The EU-Ireland, US-Northern California, and Asia Pacific-Singapore Regions provide read-after-write consistency for PUTs of new objects and eventual consistency for overwrite PUTs and DELETEs.

3.2.3 Query Model

Data can be accessed using the AWS console, REST and SOAP APIs. Amazon also provides language-specific APIs (Java, PHP, Ruby, .Net) that provide basic functions that are not in the SOAP or REST APIs such as error handling and request retries.

Create Bucket: Must specify the geographical region in which data will be stored (default if not specified is the US region).

Delete Bucket: Deletes a bucket. All of the objects in it must be deleted before.

Create Folder: Creates folder within bucket. Available only through the AWS console.

Delete Folder: Deletes a folder. A folder can be deleted without deleting its objects. Available only through the AWS console.

List Keys: Views objects that are stored in a certain bucket or folder. Keys can be filtered based on a prefix.

Read Object: Reads data belonging to a key from a bucket with two options: view it in a browser or download and save it locally.

Write Object: An Object can be created in a certain bucket, data file is uploaded and metadata can be added at creation time and manipulated later.

Delete Object: Deletes a key and its data and metadata.

Edit object metadata: System metadata are sometimes processed by Amazon. User metadata are edited by user only and never processed by Amazon. The maximum size for metadata is 2 KB.

3.2.4 Access Control

Data access is controlled using bucket policies that specify users' permissions or Access Control List (ACL) that define permissions associated with each bucket or object. By default object owner has full privileges and can grant permissions to other accounts. Bucket and object permissions are independent i.e. an object does not inherit the permissions from its bucket (each has a separate ACL)

Bucket Policies: A bucket owner can write a bucket policy or use AWS policy generator tool to define access rights for S3 buckets and objects. Policies are written in JSON and use the access policy language. They are created, retrieved, and deleted by the bucket owner and can be used for the following:

- allow/deny bucket level permissions.
- deny permissions on objects in a bucket.
- allow permissions on objects in a bucket (only if the bucket owner is the object owner as well). If the object is owned by a user other than the bucket owner then permissions can be controlled using ACL.

Access Control List (ACL): Each object and bucket has an ACL attached to it. Amazon creates a default ACL granting the owner full control over the resource whether it is an object or a bucket. ACL states the AWS accounts or groups that have access to a resource and which kind of access they have (read, write, full control). ACL can be created and managed using the AWS console or using supported APIs(REST, Java, .Net, etc.)

Query String Authentication: It can be used when a third party browser access to resources on S3 is required. Owner must specify an expiration date of the query, add a signature, place the data in an HTTP request and distribute it to users or embed on a web page.

3.3 Amazon SimpleDB

It is a web service that provides structured data storage in the cloud. SimpleDB automatically creates and manages multiple geographically distributed replicas of users' data to enable high availability and data durability. SimpleDB is widely used to store data for online games. Another use case is to index Amazon S3 object metadata (store pointers to S3 object locations and detailed information about the objects (metadata). SimpleDB was created to complement S3 and EC2 [Ado11c].

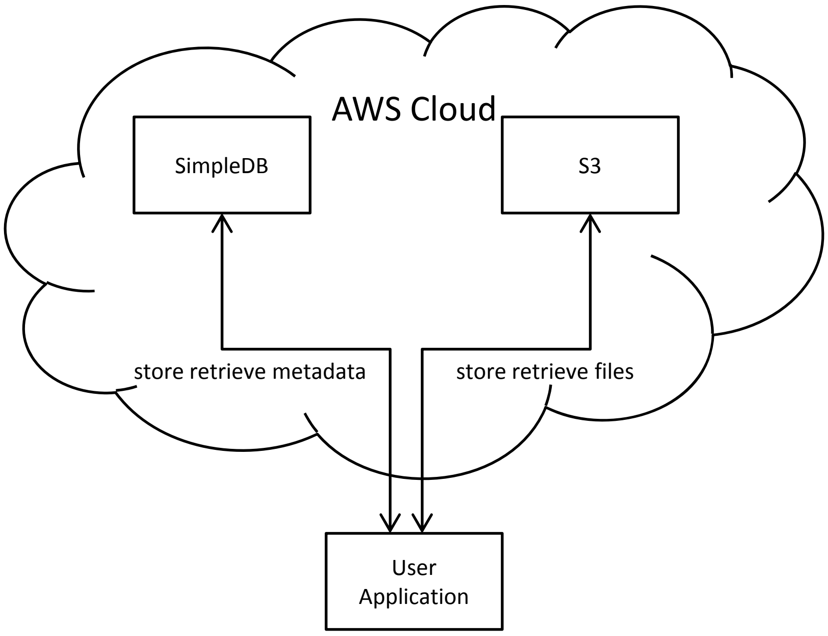

Figure 3.3: Typical way of using SimpleDB and S3 adapted from [PC10]

3.3.1 Data Model

It is non-relational schema-less data model. There are no data types and all values are considered variable length character data [Ado11c]. It is organized into domains compared to tables in the RM. There are no configuration options to set except for the Domain name. Domains are composed of items (rows in RM) that represent objects. Items have attributes (name-value pairs). It is allowed for each item within a domain to have different attributes. Different domains have different attributes and each attribute can have multiple values. Amazon offers the option to choose the geographical region where domains are stored (US East, US West-Northern California, EU-Ireland, Asia Pacific-Singapore, Asia Pacific-Tokyo). SimpleDB automatically indexes data for quick and accurate retrieval. Domains are also automatically replicated in different data centers within a region to achieve high availability. Figure 3.4 illustrates the SimplaDB data model.

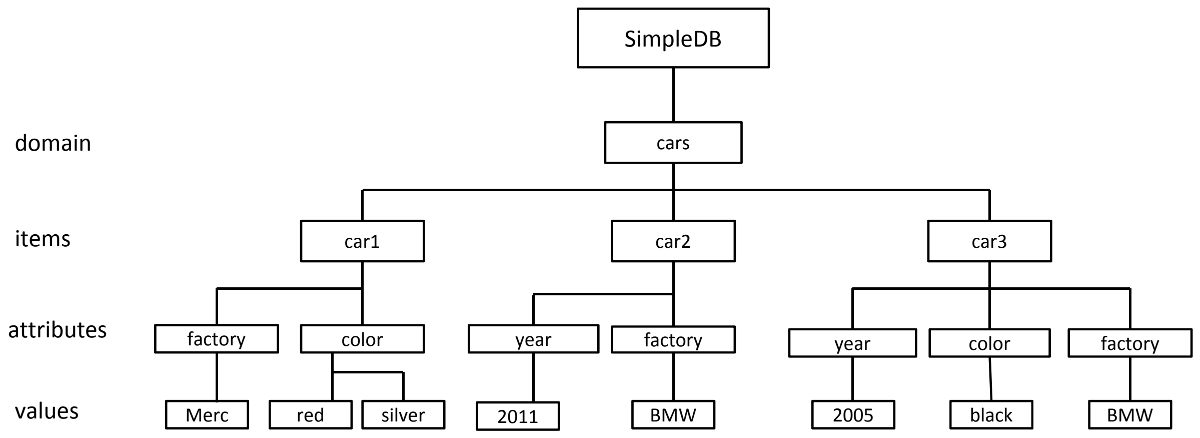

Figure 3.4: SimpleDB data model adapted from [PC10]

3.3.2 Consistency Model

SimpleDB provides the following consistency options for each read: eventual consistency and strong consistency.

Eventually Consistent Read: Read operation might not return the final update of the stored values. Consistency across all copies of data is reached after seconds. In that time frame of seconds before getting the update across all replicas, a read operation could return an old value. Repeating a read operation should return the updated value.

Consistent Read: Read operation returns a value that reflects the latest update on data. It reflects all writes that have received a success message before the read operation was issued.

3.3.3 Query Model

Data in SimpleDB can be accessed using Amazon API, REST API, SOAP API, and several language specific APIs (Java, PHP, Ruby, .Net).

CreateDomain: Creates domains to contain user's data.

DeleteDomain: Deletes any of the domains within user's account.

ListDomains: Lists all domains within user's account.

PutAttributes: Adds, modifies, or removes data within user's domains.

BatchPutAttributes: Executes multiple put operations in a single call.

DeleteAttributes: Removes items, attributes, or attribute values from user's domain.

BatchDeleteAttributes: Executes multiple delete operations in a single call.

GetAttributes: Retrieves the attributes and values of any specified item ID.

DomainMetadata: Views information about the domain, such as the creation date, number of items, and number of attributes.

Select: A SQL-like SELECT expression used to query data. User can execute queries against a domain but not across different domains. Using of foreign keys is not possible (joins are not allowed). Comparison can be applied to a single attribute in the query between the attribute and a constant value and is lexicographical in nature [Hab11]. The syntax of this statement is explained in the following:

```
SELECT output-list
FROM domain-name
[WHERE expression]
[sort-instructions]
[LIMIT limit]
```

Where

- ouput-list can be one of the following: A wildcard that refers to all of the attributes *, itemName(), count(*), an explicit list of attributes (attribute1,..., attributeN). domain-name the name of the domain to be searched
- expression: the match the user is seeking. It can be one of the following:
  - <select expression>intersection <select expression>
  - NOT <select expression>
  - (<select expression>)
  - <select expression>or <select expression>
  - <select expression>and <select expression>
  - <simple comparison>
- Sort-instruction: sorts the result based on a single attribute in ascending or descending order using ORDER BY(lexicographical in nature as well).
- Limit: the maximum number of the results to be returned by the query.

3.3.4 Access Control

SimpleDB does not provide its own system for access permissions but it uses AWS Identity and Access Management (IAM). This service allows user or organization to:

- Create users and groups under the organization's AWS account.
- Share AWS account resources between the users in the account.
- Assign unique security credentials to each user.
- Granularly control users' access to services and resources.
- Get a single AWS bill for all users under the AWS account To control access to a certain domain, IAM policy is used. For SimpleDB, domain names are the only resource type name that can be specified in a policy.

3.4 RDS

Amazon RDS allows users to create, delete, and modify Relational Database (RDB) instances (currently MySQL and Oracle). Data, query, and consistency models of relational DBMS still apply in RDS. Choosing CPU and memory capacity, installing operating system and database server, failover management, replication, and backups' scheduling are automatically managed or managed by users using simple web service calls. For example, users are given the option to choose the CPU and Memory capacity of each Database (DB) instance while creating it by choosing the instance class (small, large, extra large, double extra large, quadruple extra large) [PC10]. Users can add more storage to the DB instance at any time or can change the instance type [Ado11a]. Database instances can be accessed using AWS console, Java-based command line tool, and Amazon supported APIs (Java, PHP, .Net, and Python).

3.4.1 Architecture

Amazon does not provide details about the underlying architecture of RDS. In figure 3.5, we try to illustrate the architecture of some of the most important features. These features are explained in the following:

Multi Availability Zone: this feature enhances the availability of data in the case of failure. An availability zone has an independent infrastructure in a separated physical location from other availability zones. When this feature is enabled, a standby master in a different availability zone is synchronously updated [Ado11a]. Unlike asynchronous replication, this standby master always has the same data as the live master. In the case of failure, the status of the standby master changes to live. Then applications can read and write data from this server while a new standby master is created in the original availability zone.

Read Replicas: this feature uses the replication capability of the underlying DBMS and provides a replica of the database for read operations only. Read replicas are created in the same availability zone as the primary database server.

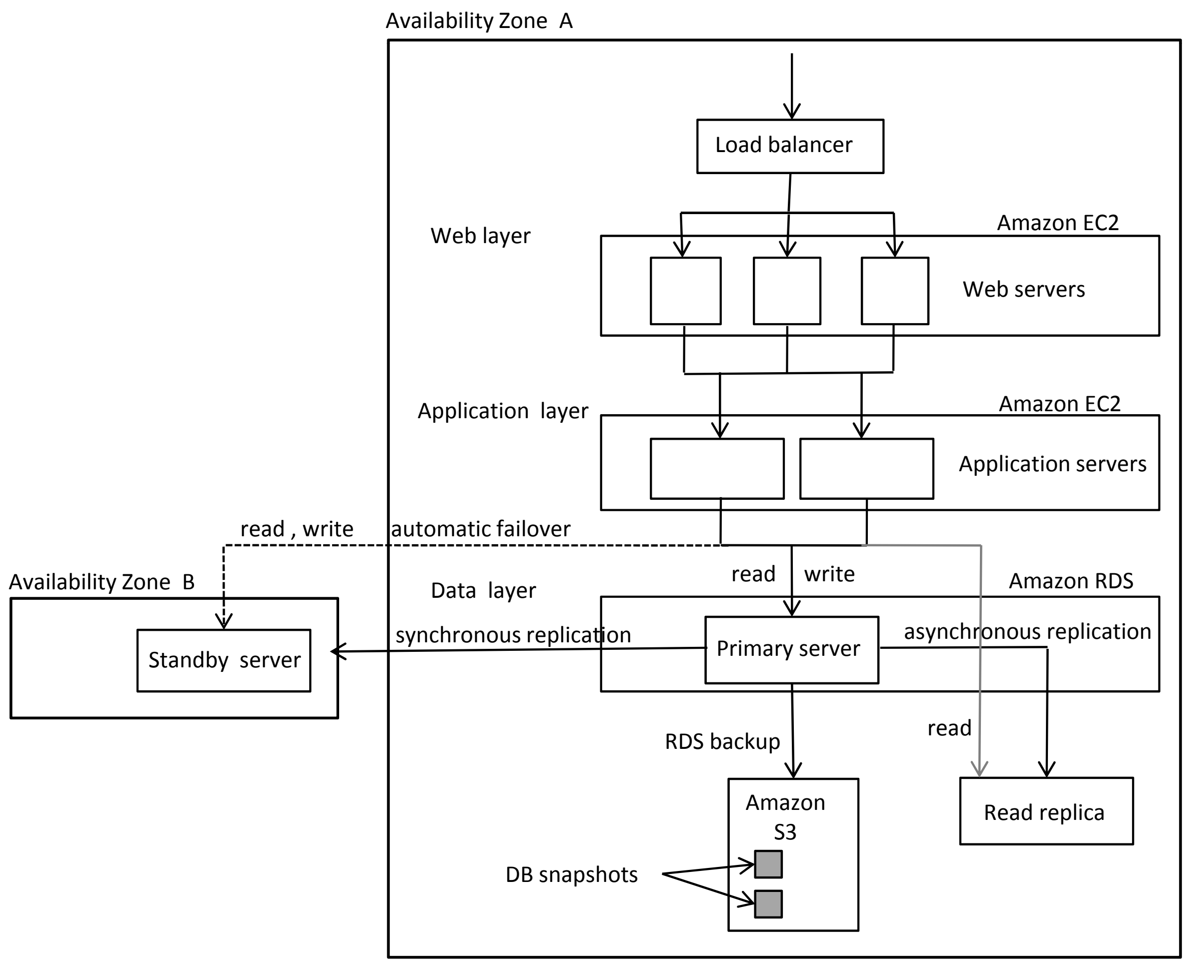

Figure 3.5: Amazon RDS Architecture

Automated Backup and DB Snapshots: RDS uses S3 to store backups and snapshots. Automatic backups perform a full daily backup of data. Data backups are kept for a period of time called retention period after that they are deleted. Snapshots are user initiated. Both automated backup and snapshots are used to restore data in the case of failure using simple AWS API calls.

3.4.2 Data Model

RDS supports the relational data model. Data is organized in tables. Relations between tables are represented in the integrity constrains of the primary and foreign keys. For more information about the relational model see traditional DBMS section 2.1.1.

3.4.3 Consistency Model

RDS supports strong consistency and ACID principles are applied.

3.4.4 Query Model

With RDS, users can use the same libraries and application that were designed to connect to MySQL and Oracle. Querying data can be done using SQL or using the AWS API calls.

3.4.5 Access Control

In addition to the access control features of the underlying DBMSs such as roles and privileges, Amazon provides more security options using database security groups [rds11]. DB security groups act like a firewall against network access. Access can be granted using IP ranges as well.

Chapter 4

Google Storage for the Cloud

Google contributions to the cloud storage aimed to answer its own needs. Whether it is Google's distributed file system, data model, or processing model it was very successful and inspired many other systems. In this chapter, we will examine GFS, Bigtable, and Google Storage for Developers.

4.1 GFS

The GFS is a distributed file system developed by Google to meet their storage needs. It stores data on large clusters of commodity hardware providing fault tolerance and high performance while serving a large number of clients. The design of this file system is build on the characteristics of Google's application workload and technological environment. We will shortly describe the most important ones in the following list [GGL03]:

Components failure is the normal situation rather than the exception: Google system is composed of hundreds or even thousands of storage components. These are accessed by hundreds or even thousands of client machines. Operating system bugs, application bugs, network failures, disk, memory or even power supplies bugs are expected. Detecting and recovering failures must be integrated in the system.

Files are huge (multi gigabytes): Parameters of I/O operations and block sizes must be adapted to that size of files in order to manage them efficiently.

Data access pattern: Analyzing data access patterns of Google's applications shows that after writing files they are often only read and in a sequential order. The largest number of changes that clients make on files are appending data rather than overwriting. This makes appending data and atomicity the main focus of optimization in GFS.

4.1.1 Architecture and Overview

The GFS cluster is made of one master server and multiple chunk servers. Each file is divided into chunks of fixed size (64 MB default). Chunks are stored as Linux files

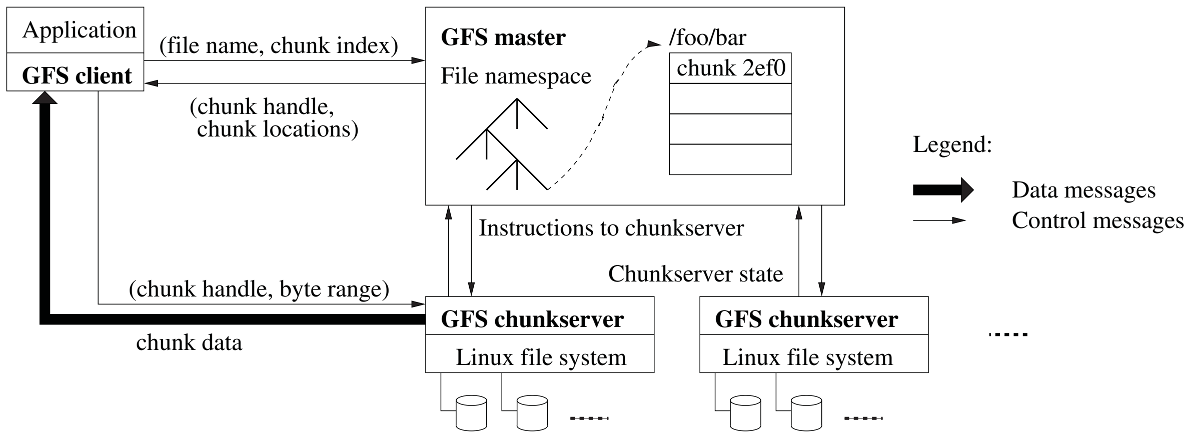

Figure 4.1: Google File System Architecture [GGL03]

and replicated on different chunk servers (3 replicas default). Chunks are identified by chunk handles (assigned by the master at the chunk creation time). The master stores the file system metadata. Metadata in GFS is of three categories: the file and chunk namespace, the file-to-chunk mapping, and the location of each chunks replica. All of the three categories of metadata are kept in master memory and only the first two are stored persistently. GFS has a relaxed consistency model. GFS uses file regions to handle consistency. A file region can be defined consistent, undefined consistent, and undefined inconsistent. The state of a file region depends on the success or failure of the changes made to the chunks and whether it is one operation or concurrent operations. When an update is successful without interference of concurrent writes, the affected region is defined and consistent. When concurrent updates affect one region and they are all successful, the region is undefined but consistent (all clients can see the same data but it is not known which update has written which data). A failed update leaves the affected region inconsistent and undefined.

GFS is not for use outside Google. However to store and access data in GFS, client code must implement the GFS API (interacting with the master and chunk servers) [GGL03]. GFS API supports standard operations such as: create, delete, open, close, read, and write. It also supports snapshot, and record append. Snapshot is used to create a copy of a file or a directory. Record append is used to allow multiple clients to write data to one file concurrently while preserving the atomicity of each client's update [GGL03].

4.2 Bigtable

Bigtsble is a non relational distributed storage system for managing structured data. It is developed by Google to store their data and used by almost 60 Google applications such as web indexing, Google Earth, Google Book Search, and Google Finance. Currently, it is not available to use outside of Google but only available through the Google App Engine. It is based on GFS and uses map reduce as the processing paradigm. A wide range of data is stored in Bigtable from web pages to satellite images. Bigtable can

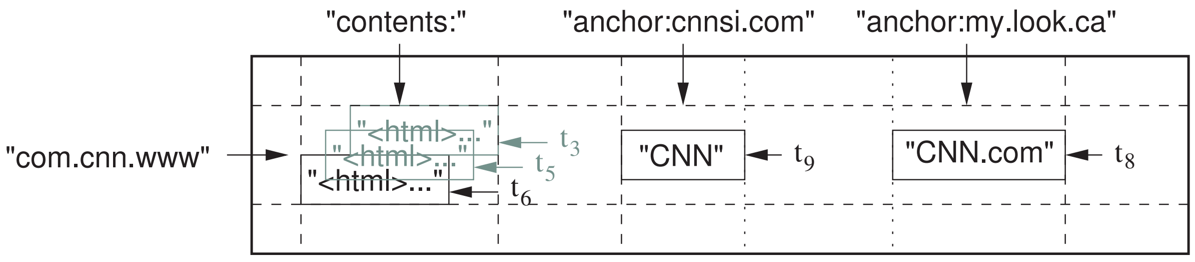

Figure 4.2: Google's Bigtable data model adapted from [CDG+06]

manage large sizes of data with low latency whether it is backend bulk processing or real time data delivery [CDG+06]. It is scalable to petabytes of data across thousands of machines, highly available, and widely applicable.

4.2.1 Data Model

Bigtable provides a simple non relational data model. Data is stored in tables and can be accessed by row or column keys. Each cell in this table can have multiple values with different timestamps (time oriented data). Each row is made of a row-key, a column key, a timestamp and a number of columns. It is similar to a key value store but instead of mapping one key to one value, it maps the row-key, column-key, and timestamp to a value. Rows in Bigtable are maintained in lexicographic order by row-key. Column keys are grouped together to form a column family. This must be done before adding data into the table. The column family is the basic unit of access control in Google's Bigtable and all data in a column family are usually of the same type and compressed together. The number of columns in a table is unbounded but the number of column families is not supposed to exceed hundreds. Access control, memory, and disk accounting are performed at the column family level. The timestamp is 64 bit integer and can be automatically assigned by Bigtable or by the user when adding data. It is used to index the multiple values or versions of data. Different versions are stored in order of the timestamp in a way that the most recent one is read first when accessing them. There is an automatic version garbage collector that can be set on the column family level with one of two options: keep the last n versions of data or keep new-enough versions. Data is indexed by the values of row-key, column key and timestamp.

4.2.2 Bigtable Architecture

The Bigtable cluster depends on a Chubby service and consists of one master server and multiple tablet servers. First of all, each table is partitioned by row boundaries into several parts called tablets and stored in SStables. The Partitioning is based on the row range of each table and aims to have 100-200 MB of data for each tablet. The tablet is considered to be the unit of distribution and load balancing in Bigtable. The Chubby service is a highly available, persistent, and distributed lock service that is used for electing the master server. Bigtable uses Chubby for discovering tablet servers, to store Bigtable schema (column family information) and ACL. The Chubby service is a single point of failure in Bigtable. If it is unavailable, the Bigtable instance goes

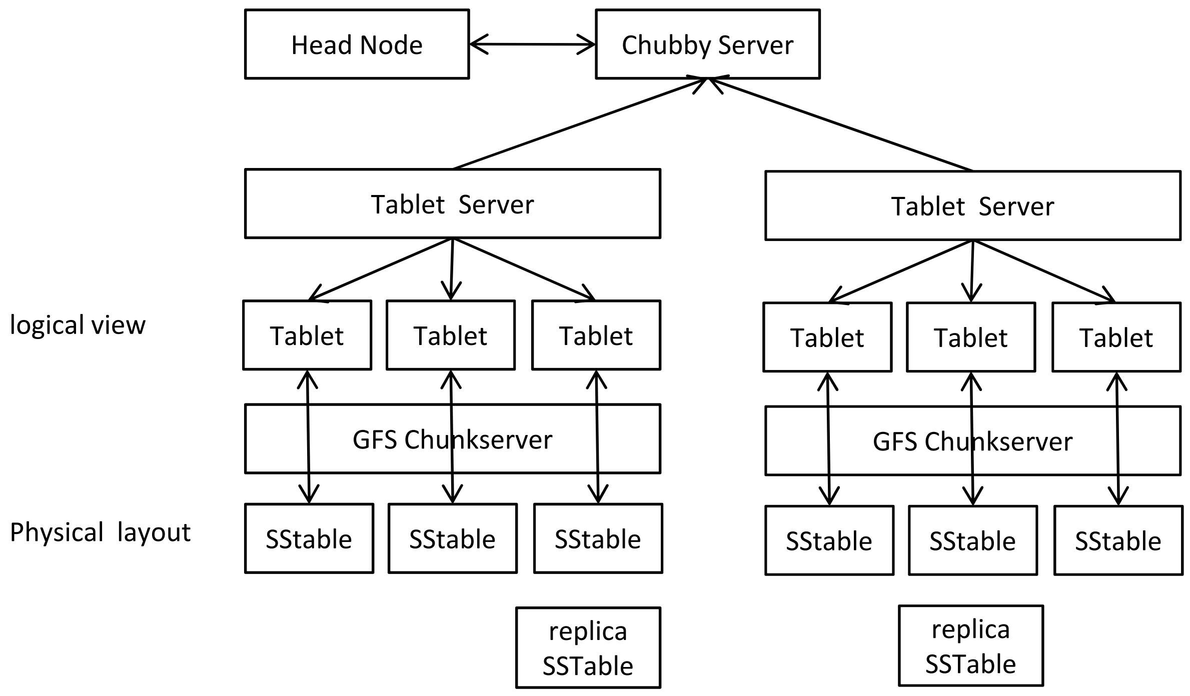

Figure 4.3: Google's Bigtable Architecture adapted from [CDG+06]

down. However, only 0.0047% of Bigtable unavailability is caused by the Chubby service unavailability [CDG+06]. The master server assigns each tablet to one tablet server. Since Bigtable stores tablets on GFS, replication is managed by GFS on the file level and tablets are not assigned to several tablet servers. Meta information about tablet assignments is stored in the master and cached in clients libraries to avoid bottlenecks.

4.2.3 Consistency model

Bigtable supports strong consistency. Data is not replicated on the Database level rather on the file system level (GFS). This means that SStable parts of a tablet are replicated in GFS. On the Database level Chubby service guarantees that one master is active and that requests are routed to the tablet server that has the specific tablet.

4.2.4 Query model

Data can be accessed using the Bigtable API and MapReduce. Each request can work with one table (i.e. no joins) and perform these operations:

- Create/delete tables and column families
- Modify cluster, table, and column family metadata
- Write or delete values in Bigtable
- Search table for a key value match
- Iterate over a subset of the data in a table

The MapReduce is used for generating and modifying data stored in Bigtable [big11]. A set of wrappers were introduced by Google to allow a Bigtable to be used as an input and as an output target for MapReduce jobs [DG04].

4.2.5 Access control

The Column Family is the basic unit of access control rights [CDG+06]. The Bigtable API can be used to manage access control rights using ACL. The Chubby service is used to store ACLs.

4.3 Google Storage for Developers

Google storage for developers is a cloud based binary-object storage service for storing and accessing data on Google's Infrastructure. It enables users to benefit from the performance, scalability, and sharing capabilities of Google's cloud [DE10]. It is used by many Google services (YouTube) and outside Google (the US Navy Visual News Service). Data is automatically replicated to multiple data centers (currently only inside US).

4.3.1 Data Model

Data is stored as objects inside buckets and must belong to a certain project. A project consists of users, authentications, billing, and APIs. An Object is the individual piece of data that can be stored in GSfD and has a data component (a file of any type or size) and a metadata component which is a collection of key-value pairs (size, last modified, share-publically). Objects are not physically stored in hierarchal structure. A folder can be created inside a bucket and can be used to organize objects. Google Storage Manager uses the $folder$ suffix to identify an object as a folder (Google Storage System and GSUtil recognize a folder as an object with the name name_$folder$). Folders can be nested and only one folder suffix is used for each folder hierarchy. The user can specify each bucket's location as the data center location which can be in the US or the EU.

4.3.2 Query Model

Google provides the Google Storage Manger, REST API, and GSUtil to access data stored in the GSfD. Google Storage Manager is available as a service in the Google APIs Console with a simple user interface. To work with REST API, users can use Google Storage simple RESTful programming interface to apply standard HTTP methods (PUT, GET, HEAD, POST, and DELETE). GSUtil is a Python application that allows users to access Google storage from a command line. The following requests on GSfD data are supported:

- Create and delete buckets
- Create and delete folders (not available through GSUtil)

- Upload download and delete objects
- List buckets and objects
- Moving copying and renaming objects
- Setting bucket and object ACLs

4.3.3 Consistency model

GSfD supports strong consistency for read-after-write, delete, and ACL update operations. When a user successfully writes something to Google Storage, any subsequent read will reach a snapshot of the newly written data no matter what replica it is communicating with. Once a read operation sees a snapshot of new data (even if it appeared to have failed to be written) any other future read will see the same data. The list operation is eventually consistent. When a user uploads an object, it can be queried immediately. But it is possible that this object will not appear in the result of an immediate list operation on the containing bucket.

4.3.4 Access Control

Data access in GSfD is controlled using ACLs. ACLs can be applied to buckets and objects. There are two ways to apply ACL: using an ACL query string parameter (for applying the ACL for a specific scope) or using the x-goog-acl request header (for applying the predefined ACL which is known as the canned ACL). When using an ACL query string parameter an xml document that contains the specified ACL must be attached to the body of the request. An ACL scope can be one of the following: Google Storage ID, Google account email address, Google Apps domain, AllAuthenticatedUsers (i.e. anyone with Google account), and AllUsers (i.e. anyone who is on the internet). For the second case of canned ACLs, there is a list of predefined ACLs that can be used with x-goog-acl header: project-private, private, public-read, public-read-write, etc. Full control is automatically granted to the project owners of all buckets inside the project. A user can not apply an ACL that removes full control of the bucket or object owners.

Chapter 5

Hadoop Storage for the Cloud

The Hadoop project was developed by Apache to support scalable distributed computing. The core of the project is a framework that enables the processing of large and distributed sets of data and detects and handles failures. Hadoop includes many subprojects among them HDFS, Hive, HBase, ZooKeeper, and others.

5.1 Hadoop

Hadoop is a framework written in Java that supports distributed processing of large data sets across clusters of machines. Each machine/node must offer local computation and storage. The Hadoop framework is highly scalable and is designed to detect and handle failures. Hadoop consists of two layers: the data storage layer or HDFS and the data processing layer or MapReduce.

5.1.1 Architecture and Overview

A small Hadoop cluster consists of a master node and slave (worker) nodes. A master node consists of jobtracker, tasktracker, NameNode, and DataNodes. A slave node consists of a tasktracker and a DataNode. It is possible to have a slave node with either a tasktracker (compute-only slave node) or a DataNode (data-only slave node) but this would be considered nonstandard. The NameNode and DataNodes are part of the data storage layer. In a large Hadoop cluster, there is a dedicated server for managing the underlaying file system that contains the NameNode. Hadoop can be deployed on any distributed file system(typically HDFS). There are alternatives such as Amazon S3, CloudStore, and FTP file system. The jobtracker and tasktracker are parts of the data processing layer. The jobtracker receives clients' requests and directs them to available tasktracker nodes. Hadoop uses FIFO to schedule jobs. Starting from version 0.19, alternative schedulers can be used such as fair Scheduler and capacity scheduler. Figure 5.1 illustrates the Hadoop Architecture.

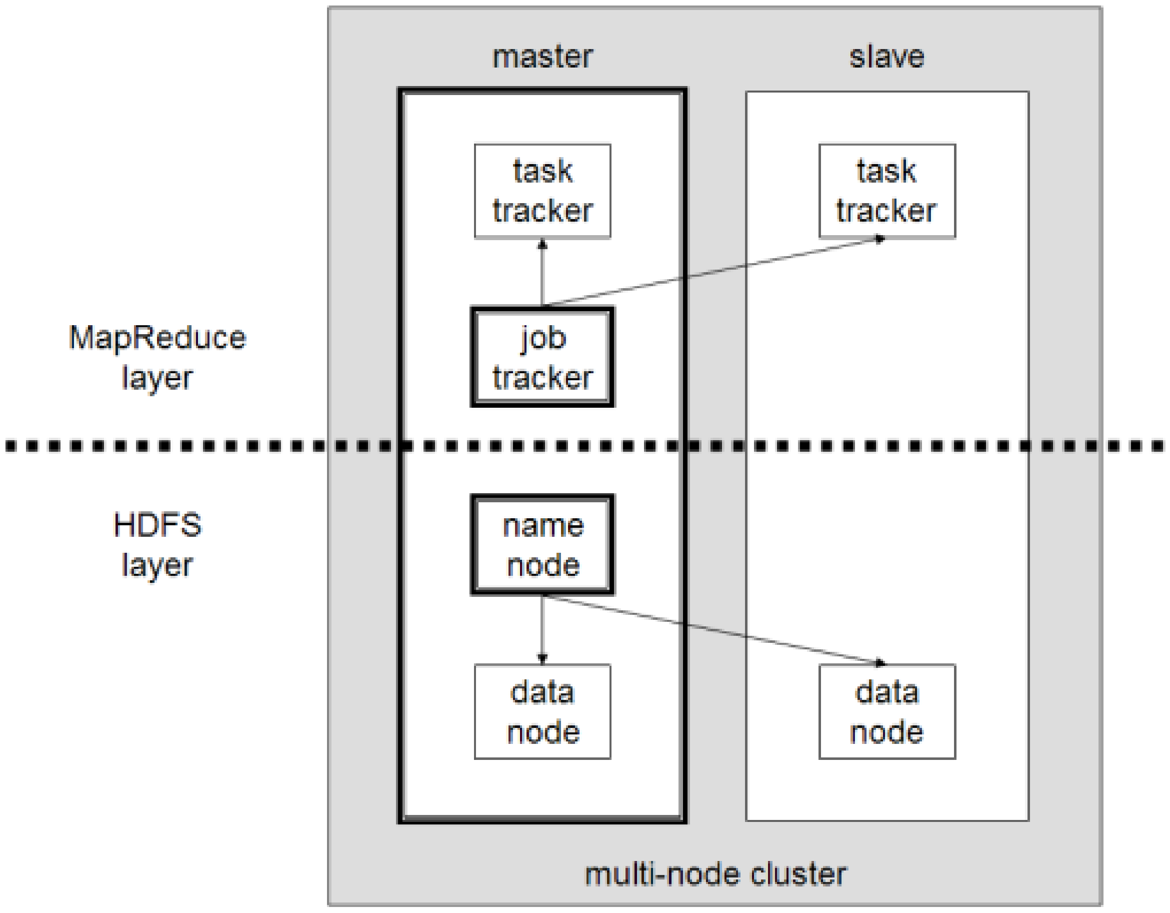

Figure 5.1: Hadoop Cluster Architecture adapted from [Nol11]

5.2 HDFS

HDFS runs on commodity hardware and provides high throughput access to large data sets. It is inspired by the GFS paper with similar assumptions and goals such as hardware failure and large data sets. The following concepts are added [Bor07]:

Simple coherency model: An application that accesses data on HDFS uses a Write-once-read-many access model. It is considered that when the file is written it is closed and needs not to be changed. So it is not supported to append writes to a file in HDFS. This makes high throughput easier to achieve. Examples of applications that use such access model are web crawlers or MapReduce applications.

Streaming data access: HDFS is designed for batch processing of large workloads rather than interactive data access. POSIX standards have been relaxed to improve the throughput rate.

Moving computation is cheaper than moving data: HDFS provides interfaces for applications to move themselves closer to where the data which will be processed is located. This is very efficient especially with large data sets. It decreases network traffic and increases throughput.

Portability across heterogeneous hardware and software platforms: HDFS should be easily portable across different platforms.

5.2.1 Architecture and Overview

HDFS is a slave master architecture composed of one master server, one NameNode several DataNodes [Bor07]. The master sever manages the file system name space and the clients file access. DataNodes serve the read and write requests of the client and perform create, delete and replicate block requests issued by the NameNode. Each file is split into several blocks and stored on a set of DataNodes. The NameNode stores the file system meta data and is responsible for opening, closing, and renaming files and directories. All blocks of a file are of the same size (typically 64MB) except for the last block. Blocks are replicated for fault tolerance. The number of blocks and replicas is configured for each file. Replica placement is optimized by implementing a rack-aware policy to achieve reliability, availability, and utilize network use. Meta data in HDFS are stored in two files: EditLog and FsImage. EditLog is a transaction log that stores meta data such as new file creations and replication factor. The FsImage stores data like file system namespace and the mapping of the blocks to the files. The NameNode keeps a version of the FsImage in memory. When the NameNode starts, it performs a checkpoint. First, it reads the FsImage and the EditLog. Then applies the changes that are recorded in the EditLog to the FsImage. The new version of the FsImage is stored back to disk and the old EditLog is truncated. A checkpoint is performed once at startup (making periodic checkpoints is supported). The NameNode receives periodic heartbeat message from the DataNodes. When it does not receive heartbeat from a node it is marked as dead and no more access requests are directed to it. When the number of replicas falls below the defined number, new replicas are made. The NameNode is a single point of failure in HDFS. There is no automatic restart and NameNode failures are

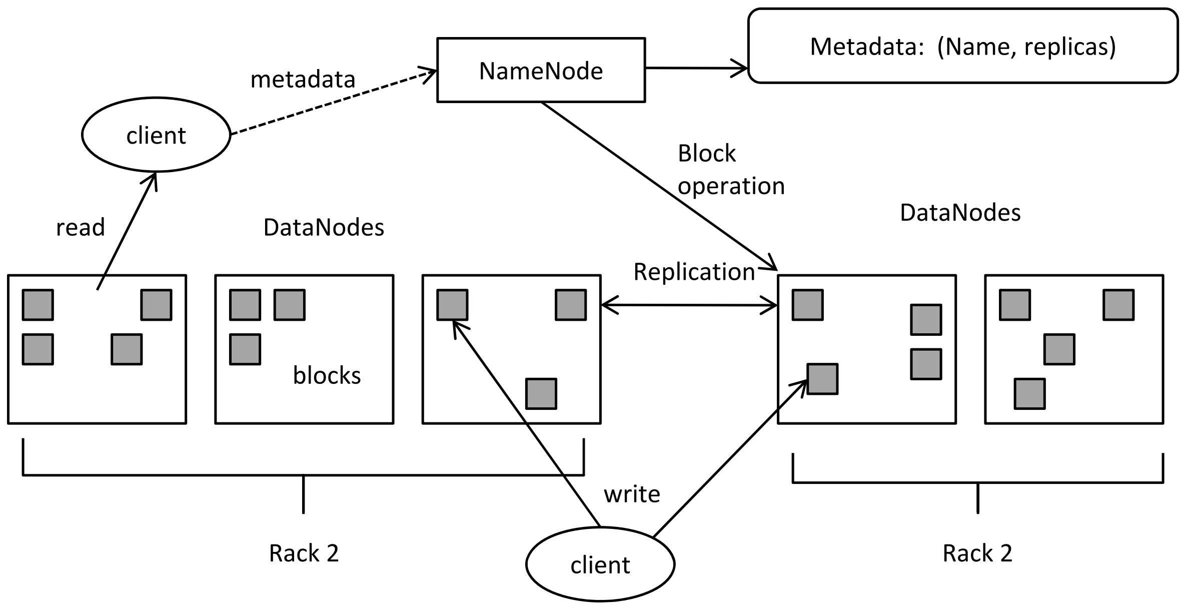

Figure 5.2: Hadoop Distributed File System Architecture adapted from [Bor08]

handled manually. Snapshots of data are not supported. When a client sends a create-file operation, it is not directed at the NameNode until a chunk-size data is cached on a local file in the client. The request then is directed to the NameNode and data is flushed to the specified DataNode. When the file is closed by the client, all remaining data is sent to the DataNode and the NameNode commits the file creation. If the NameNode fails before the file is closed, the file is lost. This approach of caching data on the client has a great impact on the throughput speed.

HDFS provides a Java API and a C language wrapper. Data can also be accessed using an HTTP browser and an FS shell (command line interface). There is a DFS Admin Command Set which is used only by the HDFS administrator.

5.3 Hive

Hive is a data warehouse system built on top of Hadoop. It started at Facebook to overcome some limitations such as Hadoop's lack of command line interface for end users, ad-hoc query support, and schema information. It is now a part of the Apache Hadoop project. Hive use cases include log processing, text mining, document indexing and predictive modeling.

5.3.1 Architecture

Hive is made of the following components which are illustrated in figure 5.3:

External Interface: allows interactive querying. Hive provides command line interface, web interface, and JDBC clients.

Thrift Server: exposes a simple API for executing queries written in the Hive Query Language (HiveQL).

Driver: is the core of Hive. It takes the HiveQL statements from the CLI or JDBC client and hands it to the transformer. Then it fetches results and returns them. It is a cross language framework. The server can be in one language and clients in different languages.

Compiler: parses queries and creates query plans. It uses the database schema and HDFS accesses are used to optimize the query plan.

Execution Engine: uses MapReduce jobs. However, it can read from HDFS without MapReduce (e.g. when reading rows from a table without filtering).

Meta Store: stores properties of tables and partitions such as schema, SerDe library, table location on HDFS, logical partitioning keys and types, and partition level metadata.

An important library is the SerDe Library (Serializer/Deserializer). It describes how to load data from HDFS files into a table and how to write it back out to HDFS in any custom format. Hive implements the Thrift DDL based SerDe interface for IO. This interface handles serialization, deserialization and the interpretation of data into individual fields. Users can write their own SerDe for their own data formats. The Meta store is outside HDFS in a Relational Database e.g. MySQL and Derby.

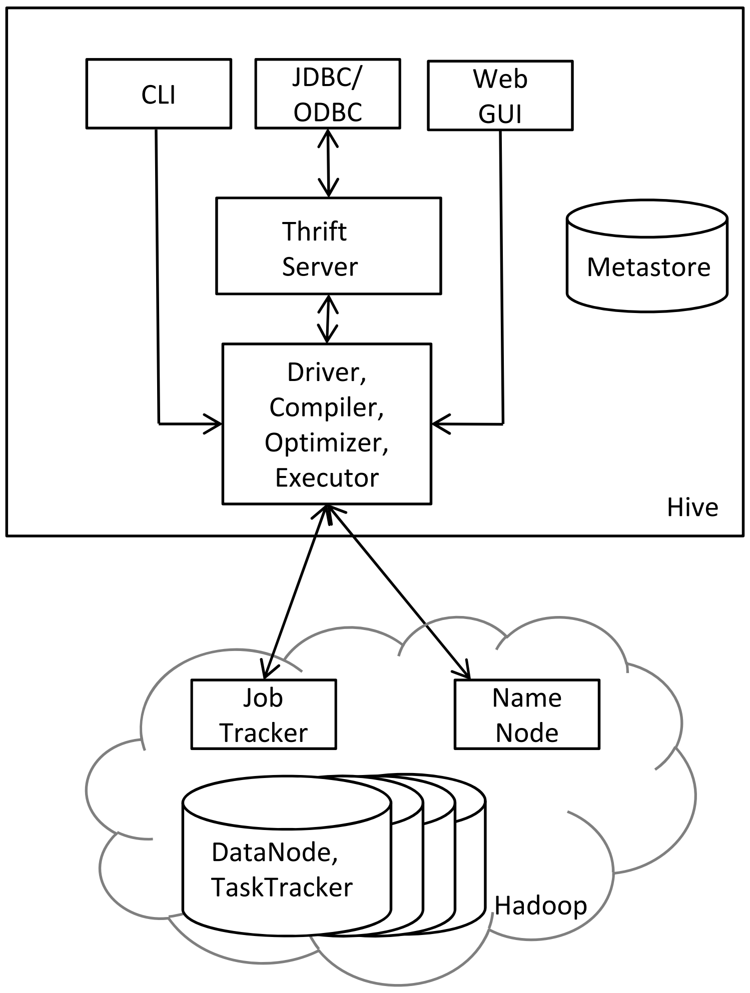

Figure 5.3: Hive Architecture adapted from [TSJ+09]

5.3.2 Data Model

Hive data model is a table-based model. Columns of a table can be of basic SQL types or composite types. Supported SQL types are: integers, floating point numbers, generic strings, dates, and boolean. Composite types include map and list that allow loading semi structured data (e.g. JSON files). Each table has a HDFS directory. Data of a table is serialized and stored as files in the corresponding directory. A table can be partitioned based on columns. Each unique value of the partition-key determines a partition of the table. Partitioning columns are not stored with the data rather than define subdirectories to store data. Each partition of a table is a subdirectory within the table's directory. For example a table books is stored in directory /userName/books. If books is partitioned by language and subject, then data with a particular language value English and particular subject value education are stored in files within directory /userName/books/language=eng/subject=educ. A partition can be hash-partitioned into buckets. Each bucket is stored in a file inside the partition subdirectory. Bucket information is used for optimization by the query planner especially for join optimization [TSJ+09].

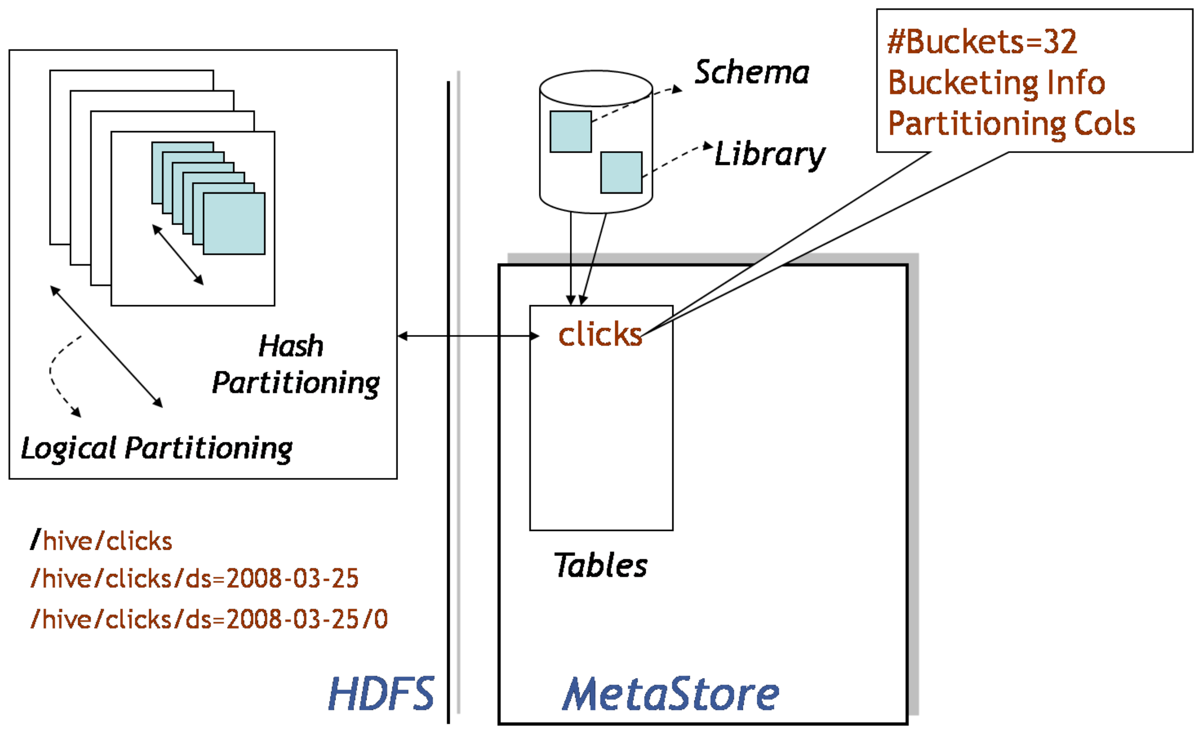

Figure 5.4: Hive Data Model adapted from [Tea09]

5.3.3 Query Model

Hive provides a command line shell for interactive querying. It also provides web interface and JDBC/ODBC client access. Data is queried using a SQL-based query language HiveQL [HLM11]. It supports basic SQL operations such as select, join, union, aggregation, and sub queries. The Creation of primary/foreign keys is not supported. However, the creation of indices has been added [Hiv11]. We will discuss the basic operations and the most important differences in the following:

DDL: Unlike other database systems, Hive gives users control over the way data is stored by deciding partitions and buckets while creating tables. The partitioned by clause defines the partitioning columns. Clustered by clause defines buckets.

```
  CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...)
[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
[LOCATION HDFS_path]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement]
```

Data is loaded into tables using several ways. User can create a data file in HDFS using MapReduce and then using the following command to load data into a table:

```
LOAD DATA INPATH ‘‘path’’
INTO TABLE tableName;
```

Meta Data Access: HiveQL allows querying the Hive metastore:

```
SHOW TABLES;
SHOW TABLES 'xxx.*';
SHOW PARTITIONS tableName;
DESCRIBE EXTENDED tableName;
```

Select and DML Support: HiveQL provides select statement with the following syntax:

```
      SELECT [ALL | DISTINCT] select_expr, , ...
  FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_list]
  [CLUSTER BY col_list
  | [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT number]
```

Where DISTRIBUTE BY and SORT BY are used to partition and sort the output of a query. CLUSTER By can replace both when data is partitioned and sorted by the same columns.

In HiveQL there is no insert, update, or delete row support. Hive tables are based on HDFS files. As we already discussed when a file is closed in HDFS, no new data can be added to it. However new data can be added to the table by adding new file to the table folder and using the bulk data command. Hive can be configured to automatically load data when files are added to a table's directory. Deletes work in a similar way by deleting the underlying HDFS files and using bulk delete.

Joins: Multiple tables can be joined in a single HiveQL statement. Only outer, equi, and left semi joins are supported. Hive converts a join into a MapReduce job if a single column is used in the join.

```
    table_reference JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference join_condition
  | table_reference LEFT SEMI JOIN table_reference join_condition
```

Where

- table-reference can be a table-factor or join-table
- table-factor can be one of the following: table name alias, table-subquery alias, or table-reference
- join condition can be

```
equality-expression ( AND equality-expression )*
```

Using MapReduce inside SQL statement : MapReduce code can be invoked from inside HiveQL using the following syntax where TRANSFORM can replace both MAP and REDUCE keywords. The map and reduce script can be written in any language. Next is an example of how they can be used in Hive:

```
ADD FILE 'map-script.py'
ADD FILE 'reduce-script.py'
FROM (
    FROM facebookStatus
    MAP statusID, statusTime, statusText
    USING 'map-script.py'
    AS word, count
    CLUSTER BY word) map-output
  INSERT OVERWRITE TABLE wordCount
    REDUCE map-output.word, map-output.count
    USING 'reduce-script.py'
    AS word, count;
```

The previous code illustrates the simple task of counting frequencies of each word in facebookStatus table using MapReduce. First, the map and reduce script files are added to Hive. The next step would be to run the map script on facebookStatus table. The third and final step is to run the reduce script against the output of the map face and save the results to the wordCount table.

Jaql can also be used to query data in Hive. Jaql is a functional, declarative query language developed at IBM research center Almaden. It is used to process large data sets. Jaql was inspired by many programming and query languages that include: Lisp, SQL, XQuery, and PigLatin [IBM11]. To benefit from parallelism, Jaql high-level queries are compiled into low level queries that consist of MapReduce jobs that are evaluated using the Apache Hadoop. Users can change in the low level queries adding functionalities that are missing in MapReduce jobs such as hashed joins and indexed access [IBM11].

5.3.4 Access Control

Data access in Hive is organized by granting privileges to users, roles, and groups . By default authorization grants of a table owner are set to null. This results in the owner of the table not being able to access it [HLM11]. To solve this, two properties in hive-site.xml must be changed. First, enable the security authorization of Hive. Second, set the grants of table's owner to ALL. The following code explains the changes:

```
<property>
  <name>hive.security.authorization.enabled</name>
  <value>true</value>
  <description>enable or disable the hive client authorization</description>
</property>
<property>
  <name>hive.security.authorization.createtable.owner.grants</name>
  <value>ALL</value>
  <description>the privileges automatically granted
  to the owner whenever a table gets created.
   An example like "select,drop" will grant select
   and drop privilege to the owner of the table</description>
</property>
```

After this, privileges such as create, update, drop, select, etc. can be granted to users, groups or roles using the following syntax:

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    [ON object_type]
    TO principal_specification [, principal_specification] ...
    [WITH GRANT OPTION]

REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    [ON object_type priv_level]
    FROM principal_specification [, principal_specification] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...
```

where

- principal-specification can be user, group, or role.
- object-type can be TABLE or DATABASE.
- priv-level can be tableName or databaseName.

5.4 HadoopDB

HadoopDB is an architectural hybrid of MapReduce and DBMS technologies. It achieves its best performance when used for analytical workloads. HadoopDB was developed by the database research group at Yale university and is now commercialized into Hadapt [had11].

5.4.1 Architecture and overview

HadoopDB idea is to have MapReduce as the communication layer between several nodes that run single-node instances of DBMS. Queries are given in SQL and translated into MapReduce tasks. All components of the system are open source (Hadoop as the MapReduce framework, PostgreSQL as DBMS, and Hive as the translation layer). HadoopDB has the following components that connect its layers as illustrated in figure 5.5:

Database Connector: receives SQL statements from the MapReduce jobs, then connects to the database systems and fetches the result back.

Catalog: stores metadata information such as table definitions, database location, driver class, replica location, and data partitioning properties. It is used by the jobTracker and taskTracker for task scheduling and data processing.

Data Loader: is responsible for repartitioning data upon loading, partitioning single node data into into smaller chunks, and bulk loading the data chunks into the single node databases. It is made of two parts: Local Hasher and global Hasher. The global hasher partitions data files stored in HDFS depending on the number of nodes in the cluster. Then the local hasher on each node loads a partition and repartition it into chunks depending on predefined maximum chunk size.

SQL to MapReduce to SQL (SMS) Planner: It extends Hive and provides the SQL interface of HadoopDB while pushing most of the data analyzing job to the database layer [ABPH+10]. This improves performance since database system query optimizers are more sophisticated than Hive's simple optimizer [ABPA+09].

HadoopDB supports relational data model and strong consistency. For data access, it provides a parallel database front end to users allowing them to process SQL queries [ABPA+09]. Queries are expressed in HiveQL which was explained in section 5.3.3

5.5 HBase

HBase is an Apache open source, non relational, column oriented, and distributed database. It is based on Google's BigTable and built on top of Apache Hadoop and Apache Zookeeper. HBase is used for random real time read and write accesses to big data [HBA11]. It is suitable for storing sparse and versioned data. The main approaches of using HBase is storing web data e.g. a table of crawled web pages and their attributes. MapReduce jobs can be used for statistics or adding new columns [Whi09].

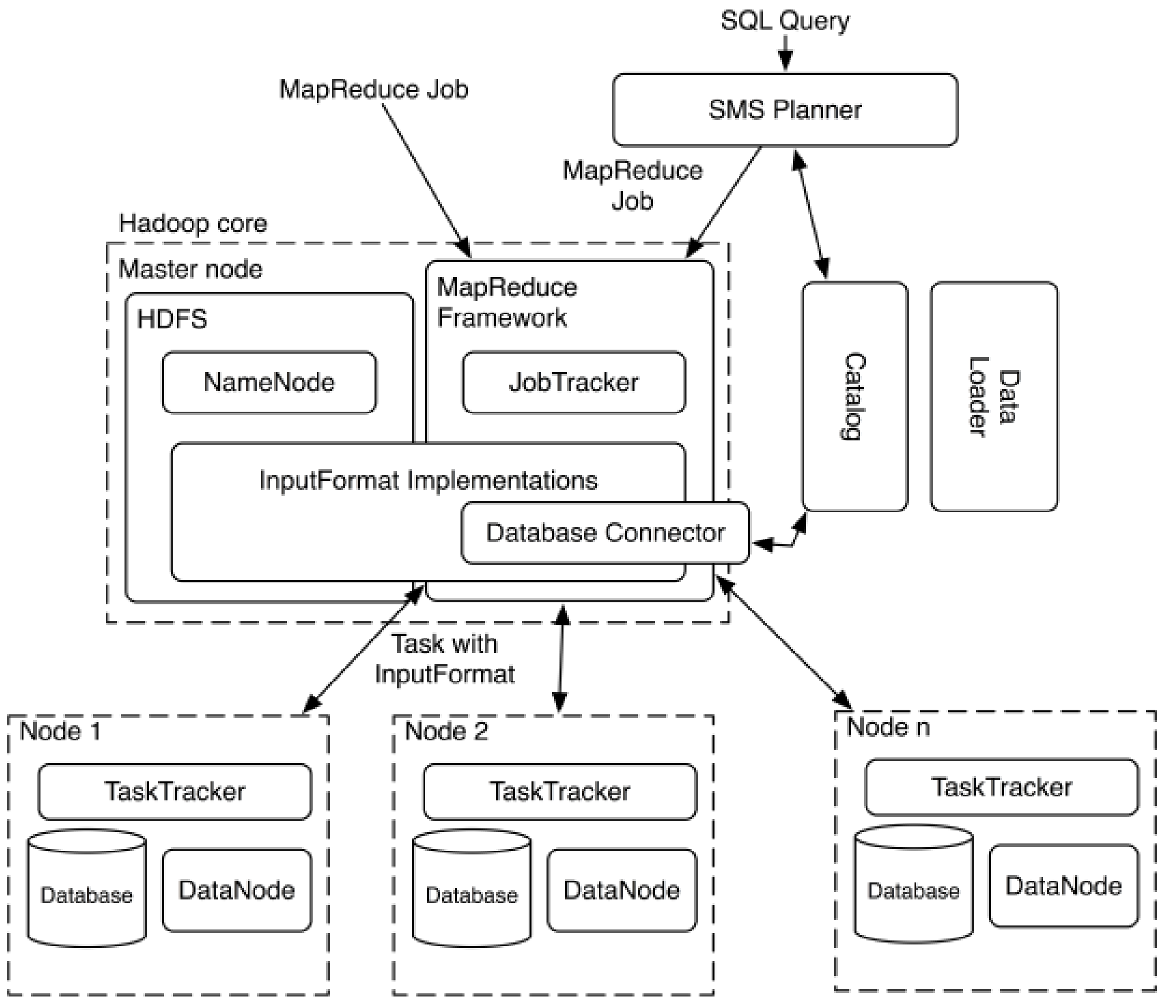

Figure 5.5: HadoopDB Architecture adapted from [ABPA+09]

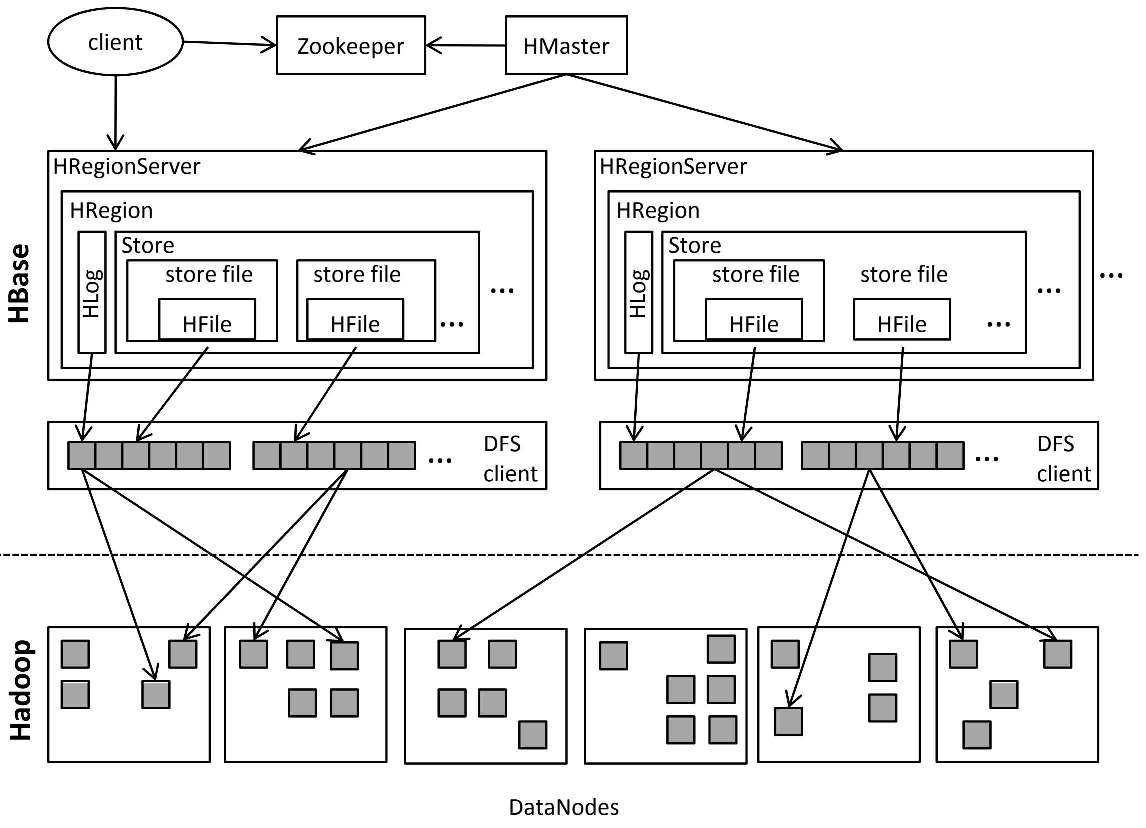

Figure 5.6: HBase Architecture adapted from [Geo11]

5.5.1 HBase Architecture

HBase has a slave/master architecture and is composed of the following components:

HBaseMaster: is responsible for handling table administrative functions e.g. adding and removing column families. It assigns regions to HRegionServers. The first region to be assigned is ROOT which locates the other regions to be assigned.

HRegionServer: stores regions and is responsible for handling clients' read and write requests. It communicates with the HBaseMaster (Heartbeat messages). In a case of HRegionServer failure, the HBasemaster region splits the write ahead log of the specified server for each region. Then It assigns each region with its write ahead log to one of the available alive HRegionServers.

HBase client: each client finds the HRegionServer that stores the region containing the requested rows. The only communication between the client and the HBase-Master is at instantiation to find the ROOT region.

An important underlying Apache service, Zookeeper (ZK), is used by HBase [zoo11]. The master and slave nodes register themselves with the ZK which is responsible for server selection and recovery. Clients connect with ZK to find a cluster. HBase manages the ZK instance. If the ZK is lost, the nodes are lost and a repair operation is needed.

5.5.2 Data Model

In HBase data is stored in sparse tables. Every HBase row-column pair is assigned a timestamp [HBA11]. Table row keys are byte arrays so they can be of any type e.g. string, serlializaed data structure, and binary representation of long [Whi09]. Table rows are sorted by row key. Columns are grouped in column families which are considered the unit of performance tuning. Columns of the same family must have the same prefix. Column families must be defined when defining the schema but columns can be added to a column family at any point. Columns of one family are physically stored together. Tables are horizontally split into regions. Each region is identified by the table name, start-row key, and end-row key.

5.5.3 Consistency Model

HBase focuses slightly less on availability and more on ensuring consistency [Hew10]. It persists data using the underlying HDFS API. Transactions are atomic on the row level and modifications are immediately available. Write operations are appended to a commit log and then added to an in memory memstore. When memsotre is filled its content is flushed to the file system [HBA11].

5.5.4 Query Model

HBase is not relational and does not support joins. Data can be accessed using Avro, Thrift, REST and several language specific APIs. MapReduce jobs can be run against tables [Whi09]. A work in progress project is HBql which aims to provide a SQL like query language for HBase [HBq11]. HBql can be executed using console or JDBC (the driver is not yet finished). It provides functionalities of table management, index management, mapping management, data manipulation (insert and delete), and selects. We will discus some of the important HBql features in the following:

DDL: HBql provides create, drop, alter, enable, disable, split, compact, flush table statements. It also allows user to create, drop, and describe indices. Next is the required syntax for creating a table:

```
CREATE TABLE table_name '(' familyDescription [, ...] ')'  [IF bool_expr]
```

Where familyDescription is:

```
family-name '(' [familyProperty [, ...]] ')'
```

The familyProperty determines properties such as block size and compression type.

Meta Data Access: HBql provides access to the DB schema similar to SQL statements e.g. describe tables and indices:

```
DESCRIBE TABLE table_name
```

Select and DML Support: HBql allows select from a single table

```
SELECT { '*' | selectElement [, ...] } FROM [MAPPING] mapping_name
 [with_clause | with_index_clause]
```

Data manipulation includes inserts and deletes. However, HBql is still is work in progress.

5.5.5 Access Control

HBase does not currently support access control. However there are several projects working on that such as an Apache supported project to implement DAC (Discretionary Access Control) [HBA11] and Secure HBase [HBS11] supported by Trend Micro. At Apache the plan is to have per row and per key value ACL by implementing a new "metacolumn" feature for HBase. The hierarchy for applying ACL [HBA11] is:

```
global -> table -> column family -> column qualifier -> row -> key value
```

It is still a discussion if the row or key value ACL overwrites the higher level ACLs or not. At Secure HBase, access is controlled on the table and column family levels. The current implementation does not support access control on the row or key value levels.

Chapter 6

Yahoo!'s PNUTS

PNUTS is scalable, parallel, distributed database system. It was developed by Yahoo! to meet their applications storage needs and it is used internally by many of their operations such as social network activities and user profiles. It automatically manages load balancing and failure recovery. PNUTS was designed to meet the following requirements [CRS+08]:

Scalability: PNUTS must achieve architectural scalability and it should be able to function during periods of rapid scale. This means that scalability should be achieved with minimal operational effort and minumal effect on the system performance.

Response time and geographic scope: The system will be used internally by other Yahoo! services that are requested by users in wide range of geographic space. It should meet the Service Level Agreement (SLA) of response time and respond to all users with low latency.

High availability, fault tolerance, and consistency guarantees: Targeted applications require high availability of this storage system. They want to be able to read and write data even in the case of failures. This leads to decisions that sacrifice strong consistency such as choosing no referential integrity constrains and no serializable transactions.

6.1 Architecture

PNUTS allows application to store data as ordered or hashed tables. Each table is horizontally divided into tablets (hundreds of Megabytes or few Gigabytes size). Tablets of an ordered table are made based on the primary key intervals. In the case of hashed tables, keys are hashed before dividing the table. Tables are replicated on different regions. Tablets are distributed across servers within regions. Each tablet is stored in one server only within a region so that each region contains only one and a complete copy of a table. Now we will explain the different parts of the system responsible for storing data:

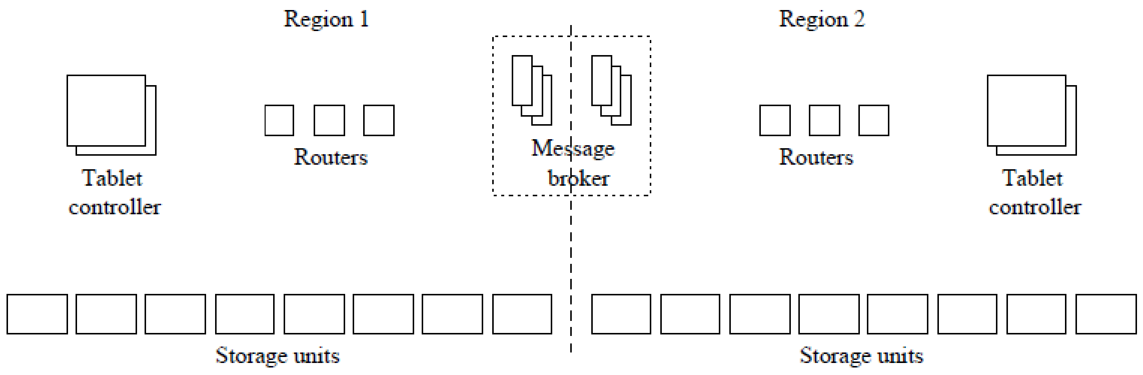

Figure 6.1: PNUTS System Architecture adapted from [CRS+08]

Region: The system is divided into regions which can be geographically distributed. Each region has multiple servers.

Storage Unit: It is responsible for storing tablets, responding to get, set, and scan requests. The storage unit can use any physical storage layer. For hash tables storage, Yahoo! uses a UNIX file system based hash table implementation. For ordered tables, Yahoo! uses MySQL with InnoDB. Records are stored as parsed JSON objects.

Router: It cashes the tablets mapping and is responsible for finding which storage unit contains the records of a client's read or write request. When a router fails a new one starts with no recovery on the failed one.

Tablet Controller: It is responsible for recovery and load balancing between storage units. It decides when to move tablets from one storage unit to another. It is also responsible for tablet splitting when it is bigger than a specified size. It stores the tablets mapping which is cashed in the router.

Yahoo! Message Broker (YMB): It is a topic based publish/subscribe system. Data updates are considered committed once they are published to YMS. YMS is responsible for propagating updates to different copies of data in all different regions. When a region fails, lost tablets are copied from another region. The tablet controller sends a copy request, a checkpoint message is sent to the YMB, and then the specified tablets are copied. Figure 6.1 illustrates the system architecture of Yahoo! PNUTS.

6.2 Data Model

PNUTS provides a simple data model. Data is organized in tables. Supported data types are string, integer, boolean, BLOB, etc. PNUTS allows the change of schema at any stage. Attributes can be added to a table without having to stop queries or updates. Referential integrity constrains are not enforced [CRS+08].

6.3 Consistency Model

PNUTS supports tunable consistency model that is called by Yahoo! Per Record Timeline Consistency. This model supports consistency between general serializability and eventual consistency. Updates are applied to all replicas of data in the same order (managed by YMB). One copy of a record is considered to be the master copy which can be changed according to the work load. All updates of a record are directed and applied to the master copy first. Clients can choose to query at several points in the consistency time line:

Read-any: It gives any valid copy of data history, which can be a stale version. It can be used in cases where performance is more crucial than consistency like in social networking.

Read-critical (required-version): PNUTS returns a version that is newer than or the same as the specified version. Version numbers are known from the output of write requests. Read-critical can be used for example when a user wants to read data immediately after writing it.

Read-latest: It returns a version that reflects all committed operation on data. It has lower latency than the two previous operations.

Write: It gives ACID guarantees but on single operations.

Test-and-set write (required-version): It performs a write operation only when the present version of a record is the same as the required version. It can be used for transactions that needs to first read a record and the manipulate data according to its value.

6.4 Query Model

PNUTS does not provide a complex or add hoc query language. However, it provides a web service RESTful API, and several language specific APIs e.g. PHP, C++, and Java. PNUTS supports selects, updates, and deletes [CRS+08]. Select is supported on a single table only (range and single point). Joins and aggregations are not supported.

6.5 Access Control

There is no published information about it.

Chapter 7

Cassandra

Cassandra is a distributed key value store for storing large sets of data up to Tera bytes. It was developed by Facebook and adopted later by Apache. The aim of starting the Cassandra project was to satisfy the reliability and scalability needs of Facebook. Cassandra brings together Google's BigTable data model and Amazon's Dynamo storage model (distributed hashed table) [LM10]. It promises scalability with no single point of failure.

7.1 Architecture

The Cassandra cluster is made of nodes. One leader node is elected by the ZooKeeper while other nodes are responsible for storage [LM10]. The leader node assigns ranges and replicas to the storage nodes. Data is partitioned using consistent hashing and precisely an order preserving hash function. Each node is assigned a number that represents its position in the ring. Each data item is assigned to a node in the ring by hashing its key and moving clockwise in the ring to find the first node with a larger position. The node that stores a data item is called the coordinator for this item. Each node is the coordinator for any data item that falls in the ring region between the node itself and the previous node. The coordinator stores a copy of the data item and replicates it on different nodes in the ring. Data is replicated on a number of nodes based on a replication factor N and a replica placement strategy. The replication factor is user defined and is set on the instance level. Cassandra supports three placement strategies: Simple Strategy (rack unaware), Old Network Topology Strategy (rack aware), and Network Topology Strategy (data center aware). The leader is responsible for balancing the load over nodes so that no node is responsible for storing more than N-1 replicas. Meta data such as dataItem-to-node mappings are stored in ZK and cached in nodes. Membership issues such as nodes joining (system scaling), failure, and recovery are managed using an anti-entropy Gossip based mechanism [Hew10]. A read/write request for a key routes to any node in the cluster. For write operations, the system routes the update to all replicas of data and waits for a minimal number of replicas to confirm the success of the write. For read operations, user specified consistency determines whether the system sends the request to the closest replicas or to all replicas in the cluster. This will be discussed in more details in the consistency model section.

7.2 Data Model

Data is stored as name-value pairs contained in key space. Key space is made of columns families which are analogues to tables. A column is made of name-value pairs with time stamps. A row is considered the container for columns [Hew10]. Thus columns are referenced by row keys. Time stamp is user defined. However, it can not be queried by users and is used by the server for conflict solving. Columns are grouped together to make column families. Cassandra is considered schema-free because columns can be added to column families at any time. In Cassandra, row keys and column names can be any kind of byte arrays. Column families structure data in four dimensions as follows:

```
key space -> column family -> column family row -> column -> value
```

The address of a value would be a row key that points to a column name that points to the value as illustrated in figure 7.1. The Cassandra data model allows having a column family inside another one, which will be called the super column family. Super column families structure data in five dimension as follows:

```
key space -> super column family -> super column family row ->
super column -> column -> value
```

In this case, the address of a value would be a row key that points to a super column name that points to a column name that points to the value as illustrated in figure 7.2.

7.3 Consistency Model

Cassandra allows users to choose a consistency level that is suitable for their applications requirements. Just like PNUTS, it gives users the choice to make a trade off between latency and consistency for each read and write operation. Consistency in Cassandra is based on the replication factor rather than on the number of nodes in the cluster. Next, we discuss the available choices:

Write Operation the following consistency models are provided:

- ANY: It means that whenever the write is committed on any node, the write returns a successes message.
- ONE: In this case, the write must be committed on at least one node and written to the memory table before it returns as successful.
- QUORUM: Data must be written to (<replication factor>/2)+1 nodes.
- LOCAL-QUORUM: Data must be written to (<replication factor>/2)+1 nodes on the same data center. This option requires NetworkTopologyStrategy.
- EACH-QUORUM: Data must be written to (<replication factor>/2)+1 nodes in each data center. This also requires NetworkTopologyStrategy.
- ALL: Data must be written to all replicas. This sacrifices latency for consistency. If any of the replicas does not respond, the write operation will fails.

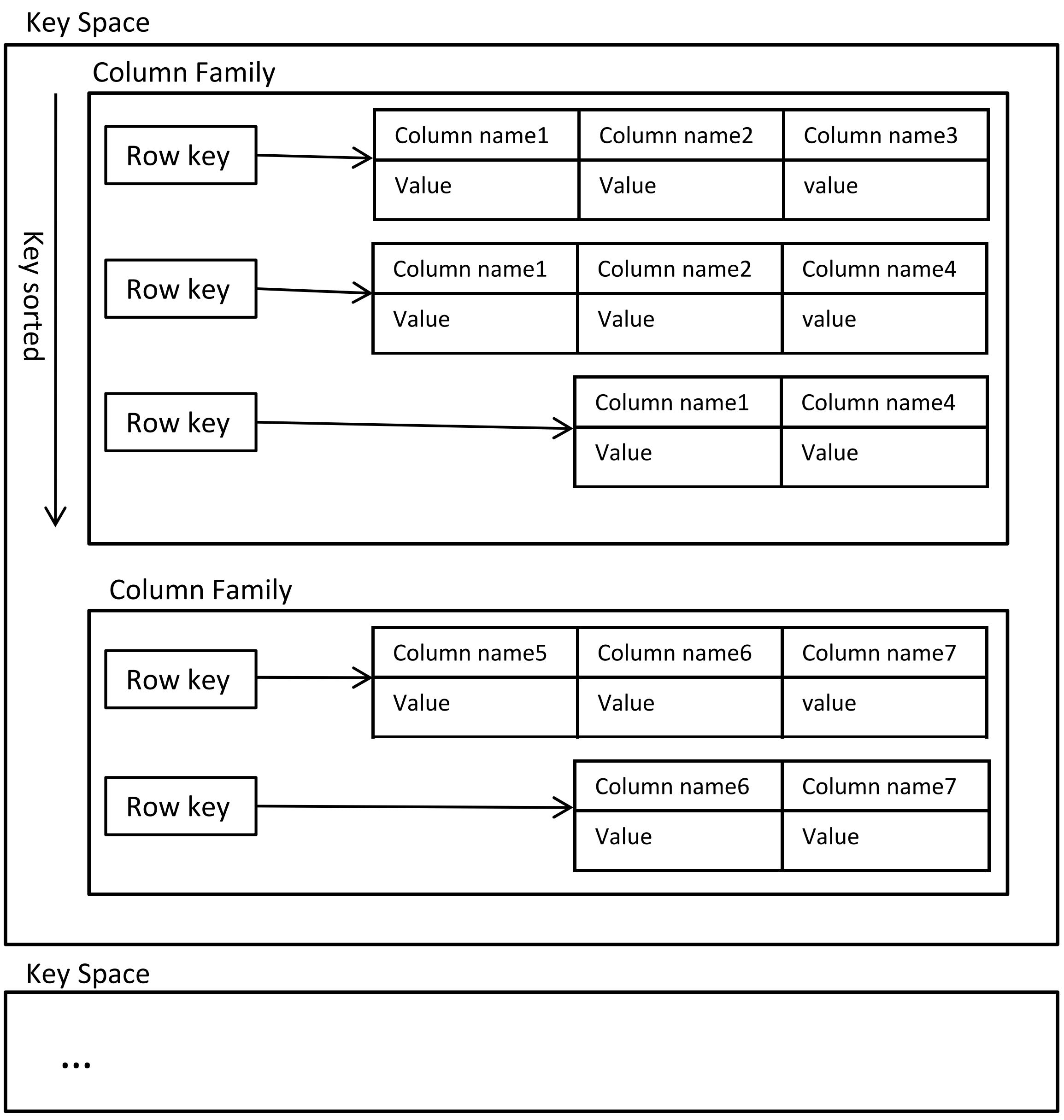

Figure 7.1: Cassandra Data Model adapted from [cas11a]

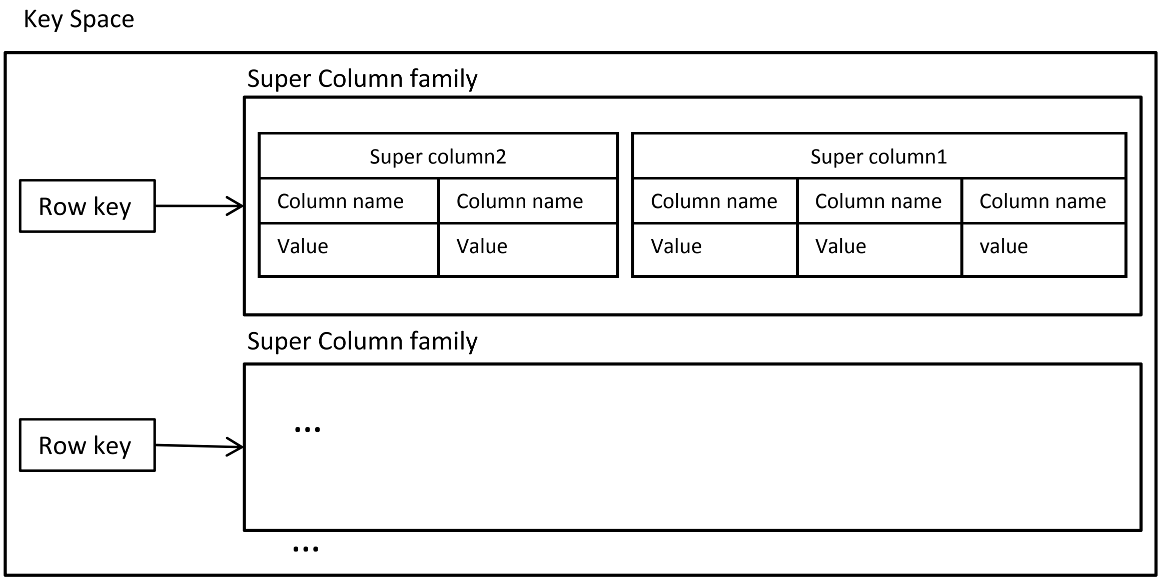

Figure 7.2: Cassandra Data Model: Super Column Family adapted from [Hew10]

Read Operation the following consistency models are provided:

- ONE: In this case, the read operation gives back the value of the first replica to answer the request.
- QUORUM: The read operation gives back the value that has the most recent time stamp after getting responses of (<replication factor>/2)+1 of all replicas.
- LOCAL-QUORUM: The read operation gives back the value that has the most recent time stamp after getting responses of (<replication factor>/2)+1 of all replicas in the same data center
- EACH-QUORUM: The read operation gives back the value that has the most recent time stamp after getting responses of (<replication factor>/2)+1 of all replicas in each data center
- ALL: Read operation gives back the value that has the most recent time stamp off all replicas. If one replica does not respond, the read operation fails.

7.4 Query Model

Cassandra provides an API that allows access through an RPC serialization mechanism, Thrift [Hew10]. There is a list of supported languages-specific APIs such as Java, Ruby, PHP, Python, etc. The thrift API is intended for internal use and for client library developers [Cas11b]. The Hadoop functionalities such as Pig, Hive, and MapReduce can be used with Cassandra DB. Cassandra provides a SQL-like query language called Cassandra Query Language (CQL). Next we discus the supported queries:

DDL: CQL provides create key space, create column family, and create index. Next we explain the create column family statement:

```
CREATE COLUMNFAMILY <COLUMN FAMILY> (KEY <type> PRIMARY KEY
[, name1 type, name2 type, ...])
[WITH keyword1 = arg1 [AND keyword2 = arg2 [AND ...]]];
```

Where

- type: used to specify the type during creation time and is optional. Supported types are:
  - bytea: Arbitrary bytes (no validation)
  - ascii: ASCII character string
  - text: UTF8 encoded string
  - varchar: UTF8 encoded string
  - uuid: Type 1, or type 4 UUID
  - varint: 4-byte integer
  - bigint: 8-byte long
- keyword: used for configuration. Here is a short list of some of possible key words:
  - row-cache-size: Number of rows that will be cached in memory.
  - key-cache-size: Number of keys per SSTable that will be kept in memory.
  - memtable-throughput-in-mb: Maximum size of the memtable before it is flushed.
  - etc.

Select and DML: CQL allows select from a single column family.

```
SELECT [FIRST N] [REVERSED] <SELECT EXPR>
FROM <COLUMN FAMILY> [USING <CONSISTENCY>]
[WHERE-CLAUSE] [LIMIT N];
```

Where

- FIRST N: determines the number of columns that will appear for each row. The default value is 10,000 columns.
- REVERSED: using this means that the sort order of the results will be reversed.
- SELECT EXPR: It specifies the columns that can appear in the result. Two options are offered: a list of column names separated by commas

  ```
  COL1, COL2, COL3,..
  ```

  or a range of columns specified by start and end column names separated by (..)

  ```
  COL1..COLn
  ```

- CONSISTENCY: specifies the consistency level that should be applied. It can be one of the options specified in the previous section 7.3.
- WHERE-CLAUSE: It can filter on Key name or range of keys.

```
WHERE KEY = keyName
WHERE KEY >= startKey and KEY =< endKey AND name1 = value1
WHERE KEY IN (keyname1, keyname2)
```

- LIMIT N: It limits the result to the first N rows. The default value is 10,000.

An INSERT in CQL is used to insert one or more column values to a row.

```
INSERT INTO <COLUMN FAMILY> (KEY, , , ..)
VALUES ( , , , ..)
[USING CONSISTENCY [AND TIMESTAMP ]]
```

A DELETE in CQL is used to remove of one or more columns from one or more rows. When column names are not specified then the entire row is deleted

```
DELETE [COLUMNS]
FROM <COLUMN FAMILY> [USING <CONSISTENCY>]
[WHERE-CLAUSE]
```

7.5 Access Control

The default authority configuration in Cassandra allows access to all resources. Users can specify the permissions for column families or key space by changing org.apache.cassandra.auth.SimpleAuthority.

Chapter 8

CouchDB

CouchDB is a document oriented non relational database written in Erlang. It manages data as JSON documents providing querying and indexing capabilities (using JavaScript in a MapReduce model). It also provides incremental replication with conflict detection and solving [JCAS10].

8.1 Architecture

The CouchDB DB server is a peer based distributed database system [Cou11a]. CouchDB hosts, whether they are servers or off line clients, can have replicas of a database. Replicas are independent and updates are made bi-directionally when a host is back online or on request. CouchDB automatically handles conflicts and copies only new documents and individual fields that changed after the previous replication request. Figure8.1 illustrates the CouchDB system architecture.

Storage Engine: the CouchDB storage engine is B-tree based. It is the core of the system which manages storing internal data, documents and views. Data in CouchDB is accessed by keys or key ranges which map directly to the underlying B-tree operations. This direct mapping improves speed significantly [JCAS10].

View Engine: it is based on Mozilla SpiderMonkey and written in JavaScript. It allows creating adhoc views that are made of MapReduce jobs. Definitions of the views are stored in design documents. When a user reads data in a view, CouchDB makes sure the result is up to date [Len09]. Views can be used to create indices and extract data from documents [JCAS10].

Replicator: It is responsible for replicating data to a local or remote database and synchronizing design documents [Hol11].

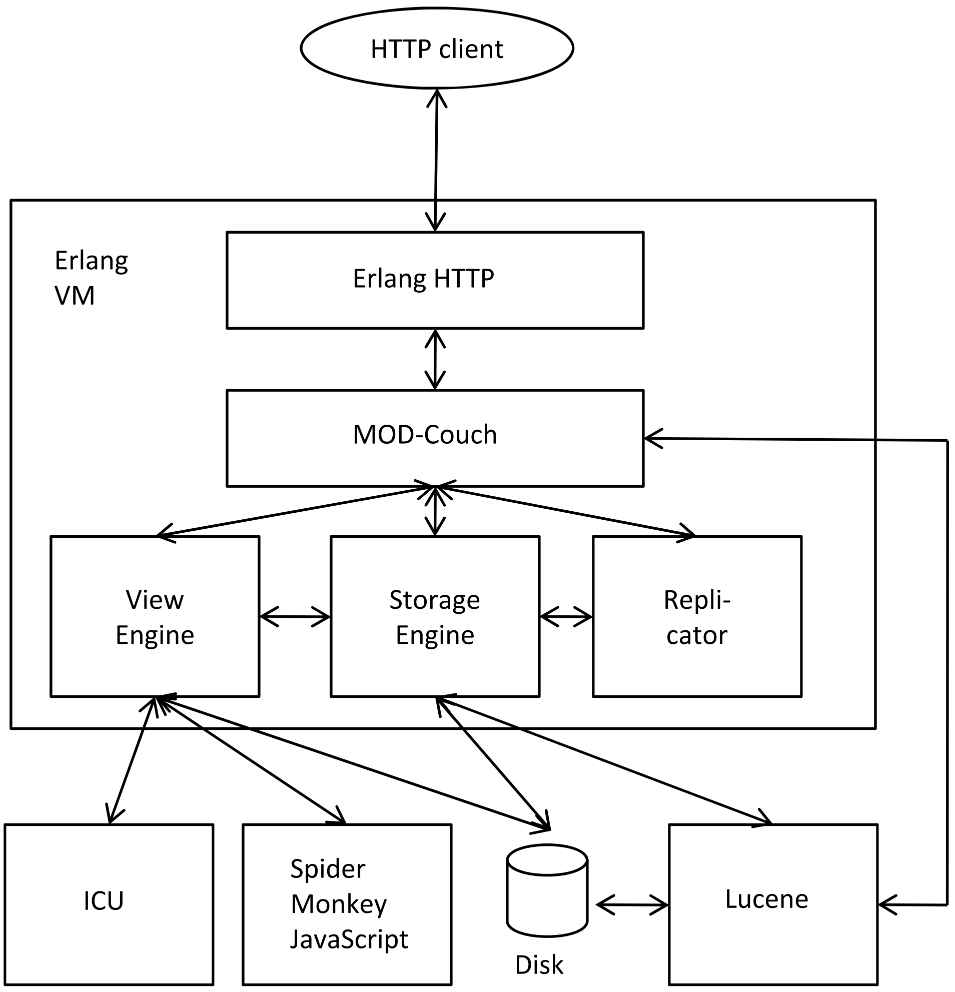

Figure 8.1: CouchDB Architecture adapted from [Cou11a]

8.2 Data Model

CouchDB manages data as a flat, self-contained, schema-less collection of JSON documents. Data is stored in documents as key-value pairs. The Document is the primary unit of data in CouchDB. It has an ID, other fields, and attachments. The document ID is unique per Database and can be any string. Document fields are uniquely named and contain values of varying types (text, date, number, boolean, ordered lists, associated maps, etc). CouchDb puts no limit to the number of fields and elements or the size of the text. An Attachment is identified by a name and includes its content type and the number of bytes it contains. Documents include metadata that is maintained by the database system. Next is an example of data stored in CouchDB:

```
{
  type: 'contact',
    firstname: 'Siba',
    lastname: 'Mohammad',
    email: ['home': 'siba@foobar.net', 'work': 'siba@foobar-working.net'],
    phone: ['home': '+49 00 0000 0000'],
    address: []
}
```

8.3 Consistency Model

CouchDB supports strong consistency where updates are lock-less and optimistic. For write operations, the client loads a copy of the document, makes the changes, and saves it back to the database. Documents are not locked when updating which allows any number of clients to be reading and writing data at the same time. If a document was changed by another client before saving an update, it fails and returns an edit-conflict error. In this case, the client tries to apply the updates to the new version. CouchDB implements MVCC for read operations. Each client sees a consistent snapshot of the database during a read operation.

8.4 Query Model

CouchDB provides a web-based administrator interface for data management, Futon. Futon allows creating/deleting databases, viewing/editing documents and creating/running views. View definitions are saved in design documents and can be replicated like data documents. A view is made of JavaScript MapReduce functions that aggregate, join, and report data. Next is an example of a view in CouchDB that counts documents with attachment adapted from [Cou11c].

```
map: function(doc) {
  if (doc._attachments) {
    emit("with attachment", 1);
  }
  else {
    emit("without attachment", 1);
  }
}
reduce: function(keys, values) {
   return sum(values);
}
```

Data can also be accessed by clients using RESTful HTTP/JSON APIs. The RESTful API allows put, delete, post, and get operations and returns data in the form of JavaScript objects in JSON [Cou11c].

8.5 Access Control

In CouchDB, three types of roles are defined: database reader, database administrator, and server administrator [Cou11b]. We describe them in the following:

Database Reader: This role is defined on the database level and grants reading all documents (data and design). Creating and editing documents is granted but not for design documents.

Database Administrator: This role is defined on the database level and has all privileges of the Database reader plus: creating/editing design documents, adding/removing database admins and readers, setting the database parameters, and executing views on the database.

Server Administrator: This role is defined on the CouchDB instance level. It has the privileges of a Database Administrator plus: creating/ deleting databases.

Roles are defined in the security object of the database (as JSON document). They are not defined on the document level. To get around this, it is suggested to create different databases for different user groups.

Chapter 9

Comparison and Classification of Current Approaches

In earlier chapters, we tried to cover the most important data management systems in the cloud. Next, we compare and classify them based on criteria of cloud computing such as availability and partitioning and DBMS such as data model, consistency, and query language.

9.1 Architecture and Overview

There are two main approaches in providing data storage management in the cloud:

RDBMS as a service: this is the first approach where RDBMS is provided as a service as illustrated in figure 9.1. Examples are RDS that provides MySQL and Oracle, and Microsoft SQL Azure that provides MS SQL server.

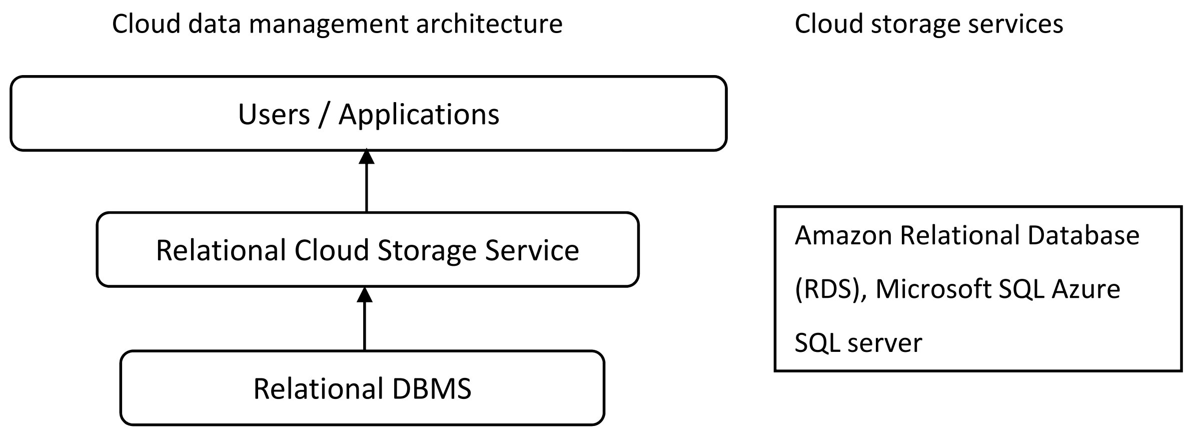
Cloud data management architecture
Cloud storage services
Users / Applications
Relational Cloud Storage Service
Relational DBMS
Amazon Relational Database (RDS), Microsoft SQL Azure SQL server

Figure 9.1: RDBMS as a service in the cloud

Combinations of components: the cloud data management system , in this case, is not a monolithic system but rather a combination of interconnected systems and components that can be replaced according to application needs. Figure 9.2 illustrates this architecture. In the following list, we describe these components and their functionalities:

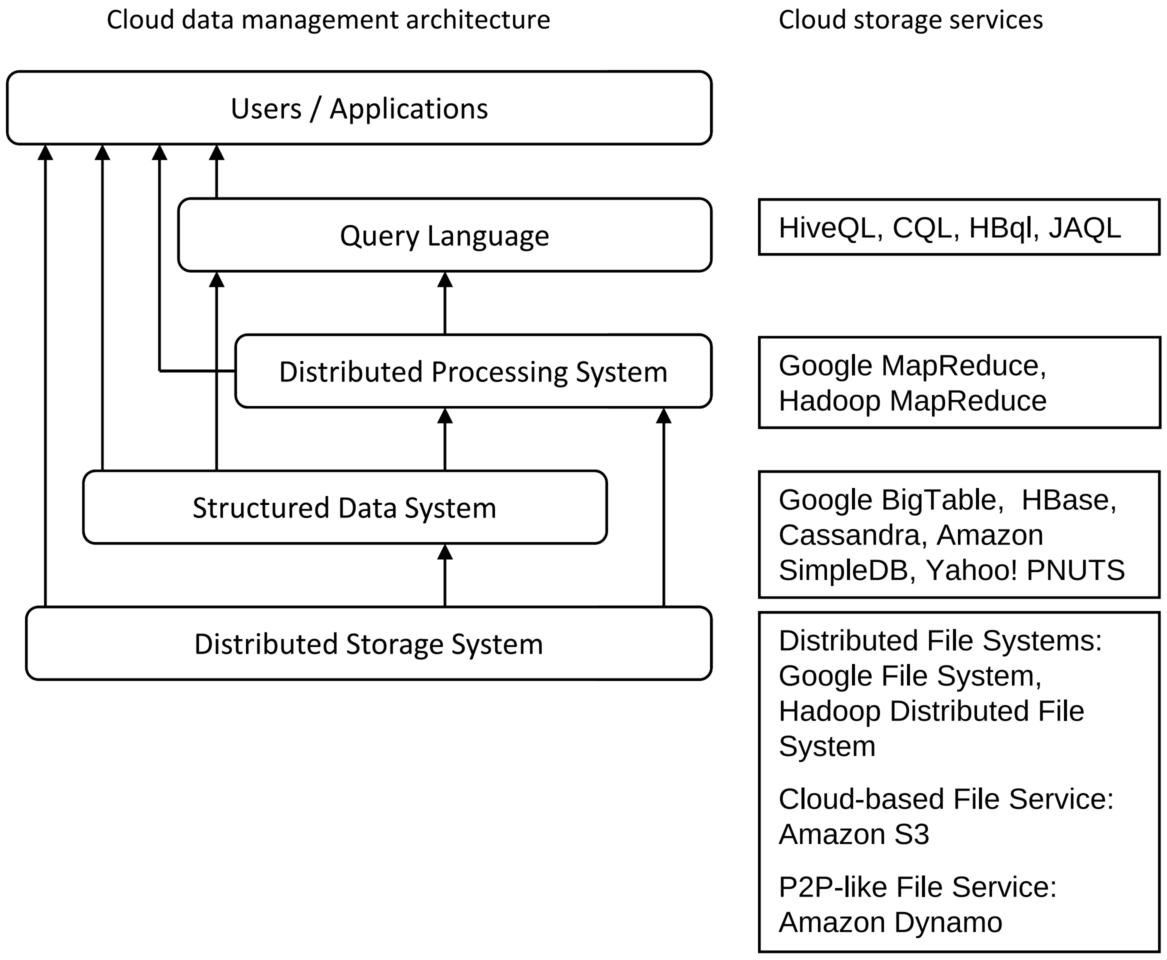
Cloud data management architecture
Cloud storage services
Users / Applications
Query Language
Distributed Processing System
Structured Data System
Distributed Storage System
HiveQL, CQL, HBql, JAQL
Google MapReduce, Hadoop MapReduce
Google BigTable, HBase, Cassandra, Amazon SimpleDB, Yahoo! PNUTS
Distributed File Systems: Google File System, Hadoop Distributed File System
Cloud-based File Service: Amazon S3
P2P-like File Service: Amazon Dynamo

Figure 9.2: Architectural components for data management in the cloud

- Distributed Storage System: it is the essential part of this architecture. Systems that are in this layer are not usually provided to the public as services but they are used internally by their vendors. Examples are Dynamo, GFS, and HDFS. Functionalities provided by systems of this layer are: performance for data access, fault tolerance, availability and scalability.
- Structured Data System: systems of this layer usually support a simple data model such as key value pairs. Examples are SimpleDB and HBase. These systems are provided to end users as web services and support various APIs for data access.
- Distributed Processing System: systems of this layer are used to process and analyze data. Examples are Google's and Hadoop's MapReduce Frameworks. Usually MapReduce is the data processing paradigm used by these systems. They provide high performance for complex data processing operations such as joins and aggregations.

- Query Language (QL)s: SQL is not usually supported. However, developers tried to mimic SQL syntax for simplicity. Examples are HiveQL, SimpleDB select, JAQL, and CQL. Most query languages of the cloud provide access to one domain or one table. Other functionalities such as controlling privileges and user groups, schema creation and meta data access are usually supported.

The previous components complement each other and work together to provide different functionalities of management for the cloud data. Structured data storage systems differ in their support of QL and MapReduce components. One important design consideration that was made fore most QL is not to support joins and aggregations. Instead, MapReduce framework is used to provide the means to perform them in the cloud to take advantage of parallel processing on different nodes. Google pioneered this by providing a MapReduce framework that inspired other systems. For more insight into connections and dependencies between these systems and components, we provide the family tree of the cloud storage systems. See figure 9.3. We use a solid arrow to illustrate that a system uses another one such as Hive using HDFS. We use a dotted arrow to illustrate that a system uses some aspects of another system like the data model or the processing paradigm. Examples are Cassandra using the data model of Bigtable, and CouchDB using the data processing paradigm of Google's MapReduce. In this family tree, we start on the left side with distributed storage systems GFS and HDFS. Then, we have the structured storage systems with API support such as Bigtable. Then comes the systems that support a simple QL such as SimpleDB. Next, we have structured storage systems with support of MapReduce and simple QL such as Cassandra and HBase. Finally we have systems with sophisticated QL and MapReduce support such as Hive and HadoopDB.

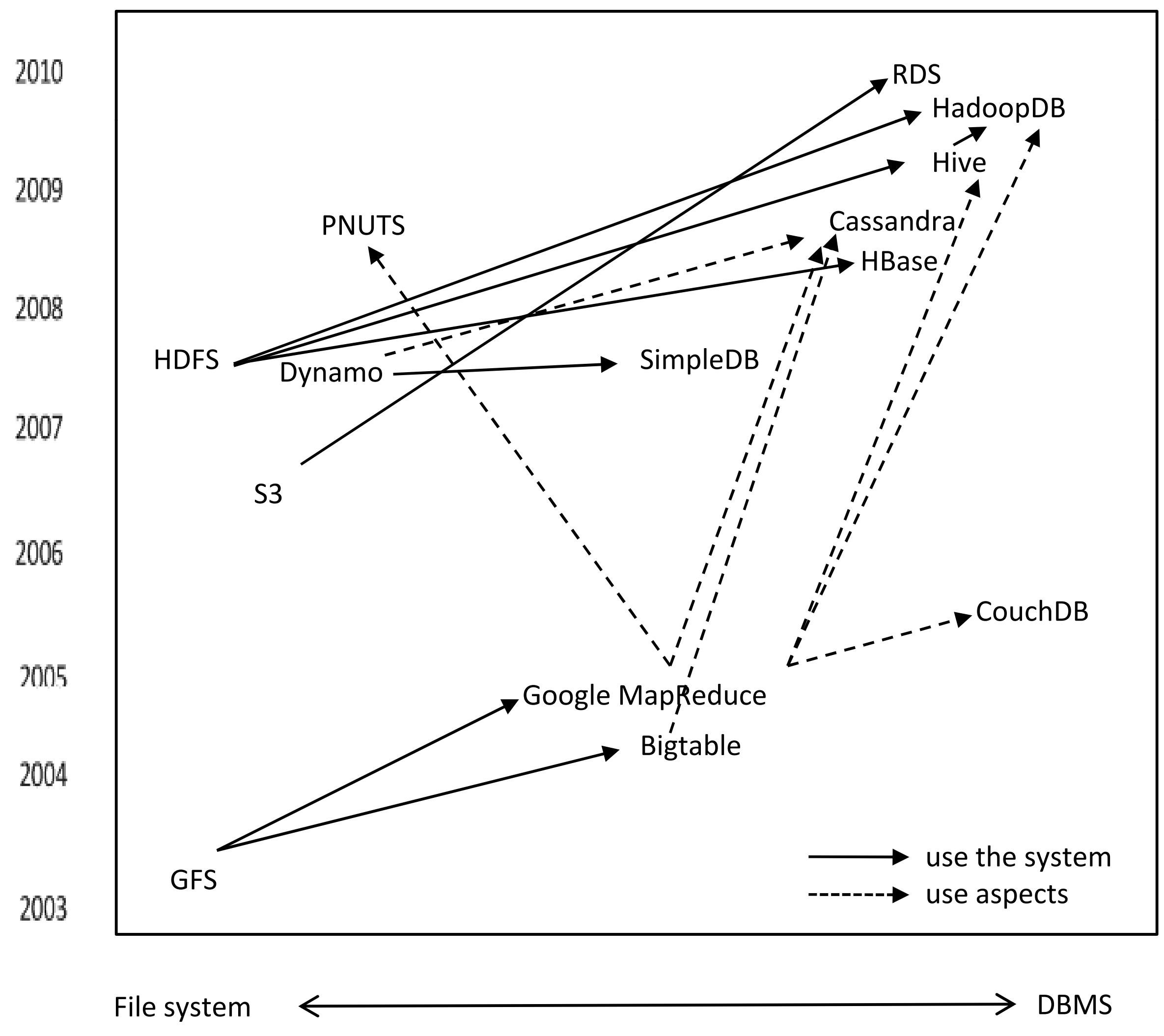

Figure 9.3: Family tree of cloud storage systems

9.2 Partitioning

Partitioning (sharding) is used by databases in the cloud to achieve scalability. There is a variety of partitioning schemes used by different systems on different levels. Some systems partition data on the file level while others partition the key space or tables. Examples of systems partitioning data on the file level are the cloud Distributed File System (DFS)s such as GFS and HDFS which partition each file into fixed sized (typically 64 MB) chunks of data. The second class of systems that partition tables or key space uses one or a composite of the following partitioning schemes [LA02, Par11, AS10] :

List Partitioning: a partition is assigned a list of discrete values. If the key of the inserted tuple has one of these values, the specified partition is selected . Example of a cloud data management system using list as the partitioning scheme is Hive.

Range Partitioning: the range of values belonging to one key is divided into intervals. Each partition is assigned one interval. A partition is selected if the key value of the inserted tuple is inside a certain range. Example of a system using range

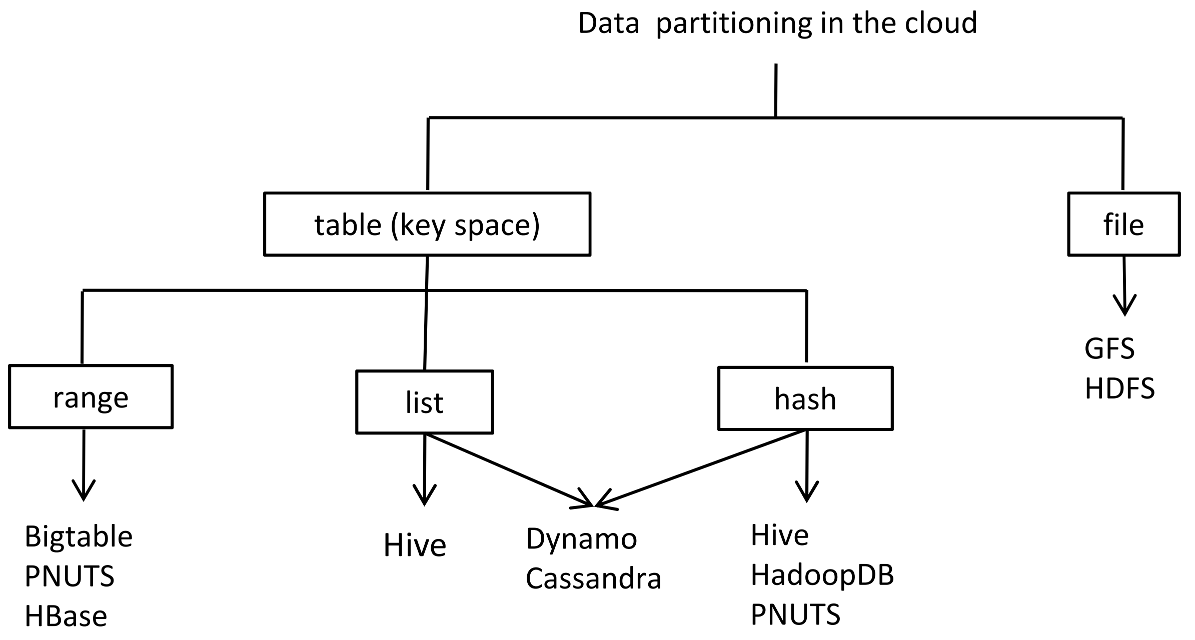

Figure 9.4: Classification based on data partitioning scheme

partitioning is HBase.

Hash Partitioning: the output of a hash function is assigned to different partitions. The hash function is applied on key values to determine the partition. This scheme is used when data does not lend itself to list and range partitioning. Example of a system using hash as the partitioning scheme is PNUTS.

There are some systems that use a composite partitioning scheme. An Example is Dynamo which uses a composite of hash and list schemes (consistent hashing). Some systems allows partitioning data several times using different partitioning schemes each time . An example is Hive where each table is partitioned based on column values. Then each partition can be hash partitioned into buckets which are stored in HDFS.

One disadvantage that comes with data partitioning is load balancing. Since most methods depend on random position assignment of storage nodes and does not take into consideration the diversity of their performance levels. This leads to non uniform distribution of data and workload. Several techniques have been used to achieve load balancing. Cassandra uses a leader node that acts as a load balancer. While Dynamo uses several strategies to maintain a uniform distribution of keys on nodes and provides a load balancer that directs requests based on load information.

One important design consideration to make is whether to choose an order preserving partitioning technique or not. Order preserving partitioning has an advantage of better performance when it comes to range queries. Examples of systems using order preserving partitioning techniques are Bigtable and Cassandra.

system	partitioning	partitioning scheme	partition	architecture
Dynamo	key space	consistent hashing	set of items	peer to peer
S3	not supported	-	-	-
SimpleDB			-	no Info
RDS	not supported	-	-	slave master
GFS	file	fixed sized parts	chunk	slave master
Bigtable	table	ordered	tablet	
GSfD	not supported	-	-	-
HDFS	file	fixed sized parts	chunk	slave master
Hive	table	2 levels: list, hash	bucket	slave master
HBase	table	ordered	region	slave master
HadoopDB	table	2 level: hash	chunk	slave master
Cassandra	table	consistent hashing	set of items	slave master
PNUTS	table	hash or ordered	tablet	peer to peer
CouchDB	not supported	-	-	no Info

Table 9.1: Comparison of data partitioning of cloud storage systems

9.3 Replication

Data replication is used by data management systems in the cloud to achieve high availability. Replication means storing replicas of data on more than one storage node and probably more than one data center. The replica placement strategy affects the efficiency of the system [Hew10]. In the following we describe the replication strategies used by cloud systems [Hew10]:

Rack Aware Strategy: it is also known as the Old Network Topology Strategy which places replicas in more than one data center on different racks within each one.

Data Center Aware Strategy: it is also known as the New Network Topology Strategy. In this strategy, clients/applications specify how replicas are placed across different data centers.

Rack Unaware Strategy: it is also known as the Simple Strategy. It places replicas within one data center using a method that does not configure replica placement on certain racks.

Figure 9.5 illustrates the classification of data management systems in the cloud based on the replication placement strategy.

The replication factor which determines the number of replicas is handled in different ways. Some systems do not reveal it to users e.g. Amazon S3 and SimpleDB. Most systems allow users to set the replication factor. One example is HDFS which allows clients to set the replication factor on the file level. For a complete overview of how different cloud systems manage data replication see table 9.3

Replication improves system robustness against node failures. When a node fails, the system can transparently read data from other replicas. Another gain of replication is increasing read performance with the help of a load balancer that directs requests to a data center close to the user. Replication has a down side when it comes to updating data. The system has to update all replicas. This leads to very important design considerations that impact the availability and consistency of data. The first one is to decide whether to make replicas available during updates or wait until data is consistent across all of them. Most systems in the cloud choose availability over consistency. However, we will discuss this in more detail later in section 9.5. The second design consideration is to decide when to perform replica conflicts resolution i.e. during writes or reads. If conflict resolution is done during write operations, writes could be rejected if the system can not reach all replicas or a specified number of them within a specific time. Example of that is the WRITE ALL operation in Cassandra, where the write fails if the system could not reach all replicas of data. However, some systems in the cloud choose to be always writeable and push conflict resolution to read operations. An Example of that is Dynamo which is used by many Amazon services like the shopping cart service where customer updates should not be rejected.

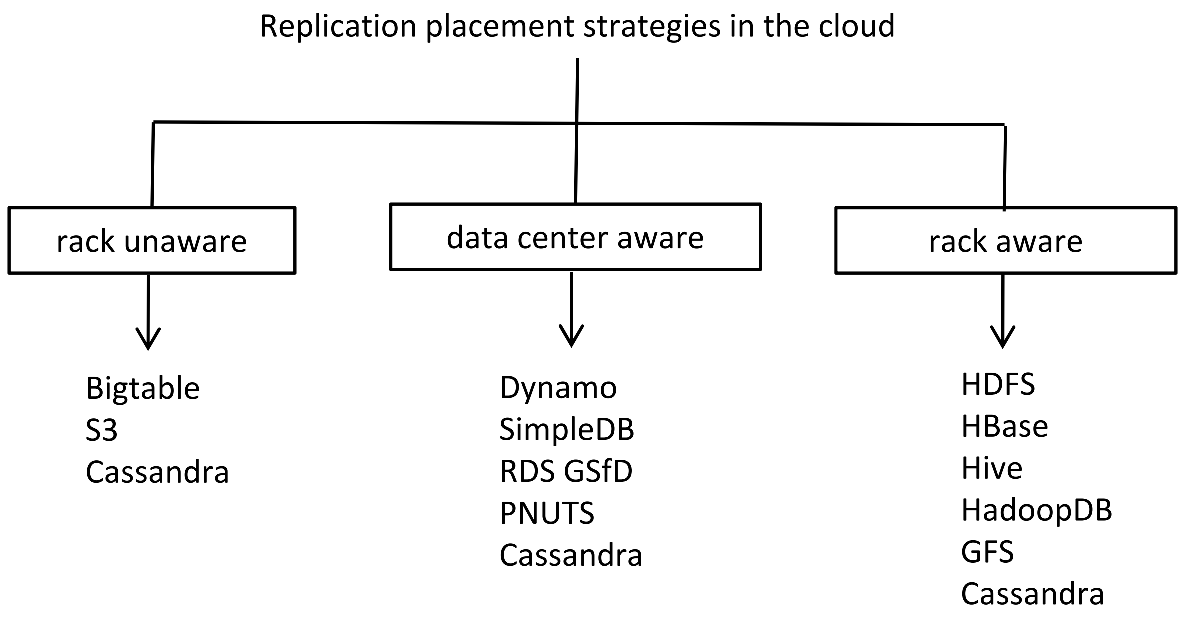

Figure 9.5: Classification based on replica placement strategy

system	replication level	placement strategy	replication factor
Dynamo	item	data center aware	per instance
S3	item	data center unaware	automatic, RRS
SimpleDB	domain	data center aware	automatic
RDS	database	within region in different AZ	per database
GFS	chunk	rack aware	per part of fileNS
Bigtable	sstable	data center unaware	per part of fileNS
GSfD	no Info	data center aware	automatic
HDFS	block	rack aware	per file flexible
Hive	DB file	uses HDFS	defined in HDFS
HBase	DB file	uses HDFS	defined in HDFS
HadoopDB	chunk	uses HDFS	defined in HDFS
Cassandra	per record	rack unaware, rack aware, data center aware	per key space
PNUTS	tablet	data center aware	automatic
CouchDB	database	manual	-

Table 9.2: Comparison of replication of cloud storage systems

9.4 Classification based on Data Model

The main data models used by cloud systems:

Relational Model (RM): the most common data model for traditional DBMS. Examples of cloud storage systems supporting this model are HadoopDB and Amazon's RDS.

Key Value: the most common data model for cloud storage. It has three subcategories:

- Row Oriented: In this model, data is organized as containers of rows that represent objects with different attributes. An Example is SimpleDB where data is organized in domains of rows. In this approach, access control lists are applied on the object(row) or container(set of rows) level.
- Document Oriented: data in this model is organized as a collection of self described JSON documents. Example of a cloud system with document oriented data model is CouchDB. Document is the primarily unit of data which is identified by a unique ID. CouchDB automatically indexes data by document IDs.
- Wide Column: In this model, attributes are grouped together to form a column family. Column family information can be used for query optimization. Some systems perform access control and both disk and memory accounting at the column family level such as Bigtable.

  Systems of wide column data model should not be mistaken with column oriented DB systems. The former deals with data as column families on the conceptual level only. The latter is more on the physical level and stores data by column rather than by row.

See table 9.3 for a comparison of data storage systems' data model, schema, and index support. Figure 9.6 illustrates the classification of data storage systems in the cloud based on the data model.

system	Data model	Schema	Index
S3	key value object, bucket, folder	no	no
SimpleDB	key value items, domain	no	automatic
RDS	relational	yes	yes
Bigtable	key value row, CFamily, table	yes	automatic
GSfD	key value object, bucket, folder	no	no
Hive	table-based	yes	yes
HBase	table CFamily flexiable	yes	yes
HadoopDB	relational	yes	yes
PNUTS	relational	yes flexiable	yes
Cassandra	key value CFamily flexiable	yes flexiable	yes
CouchDB	key value document oriented	no	yes

Table 9.3: Comparison of data model of cloud storage systems

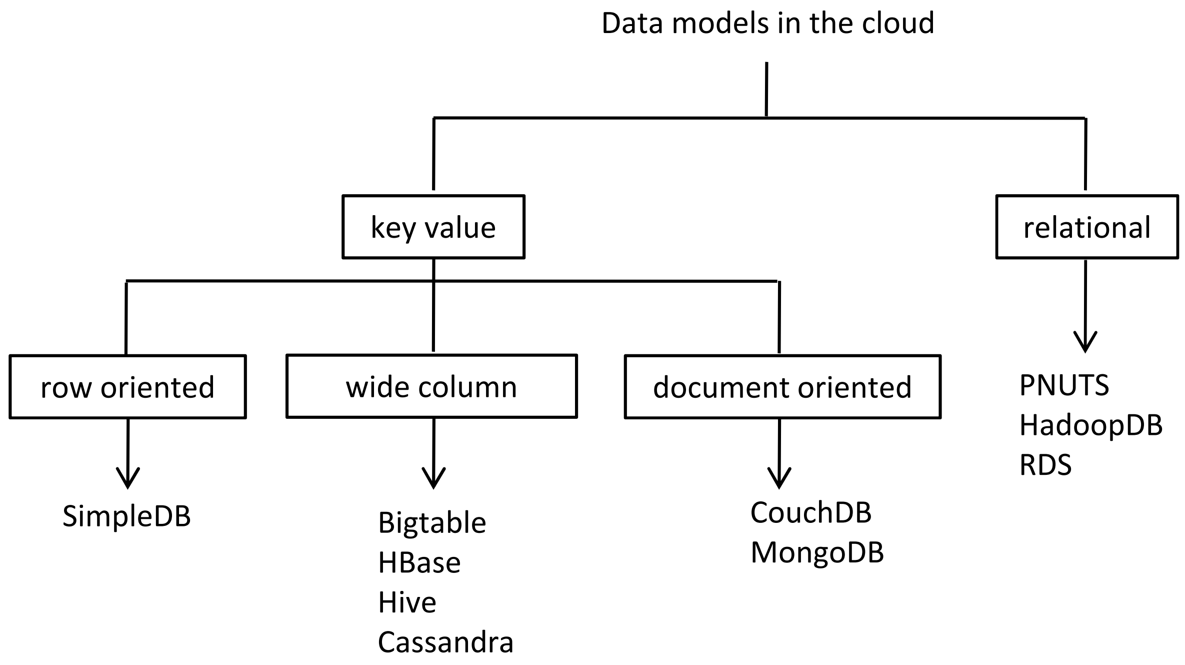

Figure 9.6: Classification based on data models

9.5 Consistency, Availability, and Partition Tolerance

We will compare and classify the cloud data storage systems based on Consistency, Availability, and Partition tolerance (CAP). The CAP theorem [Bro11, GL02] states that consistency, availability, and partition tolerance are systematic requirements for designing and deploying applications for distributed environments. These concepts mean the following:

- Consistency means that modifications on data must be visible to all clients once they are committed. At any given point in time, all clients can read the same data.
- Availability means that all operations on data whether read or write must end with a response within a specified time.
- Partition tolerance means that even in the case of components' failures, operations on the database must continue.

The CAP theorem also states that developers must make trade off decisions between the three to achieve high scalability. So if we want a data storage system that is both strongly consistent and partition tolerant, its availability will be sacrificed . Because the system has to make sure that write operations returns a success message only if data has been committed to all nodes which is not always possible because of network or node failures. In the cloud, there are basically four approaches for systems in dealing with CAP.

Atomicity Consistency Isolation Durability (ACID): with ACID, all users have the same consistent view of data before and after transactions. A transaction is atomic. If one part fails, the whole transaction fails and the state of the data is left unchanged. But once the transaction is committed, it is protected against crashes and errors. Data is locked while being modified by a transaction. If another transaction tries to access data while data is locked, it has to wait. ACID is used by systems which prefer very strong consistency and do not mind losing some availability for that.

Basically Available Soft-state Eventual consistent (BASE): with BASE, the system does not guarantee that all users see the same version of data item but guarantees that all of them see data. It means that data is not locked while a transaction is working on it. BASE is used by systems that choose weaker consistency to have higher availability.

Strongly Consistent Loosely Available (SCLA): this approach provides better availability than ACID and stronger consistency than BASE. It is used by systems that choose higher consistency and sacrifice a little bit of availability. Examples of that are HBase and Bigtable.

Tunable consistency: in this model, consistency is configurable for each read and write request. Users decide the level of consistency in balance with the level of availability. Then the system can work in high consistency or high availability

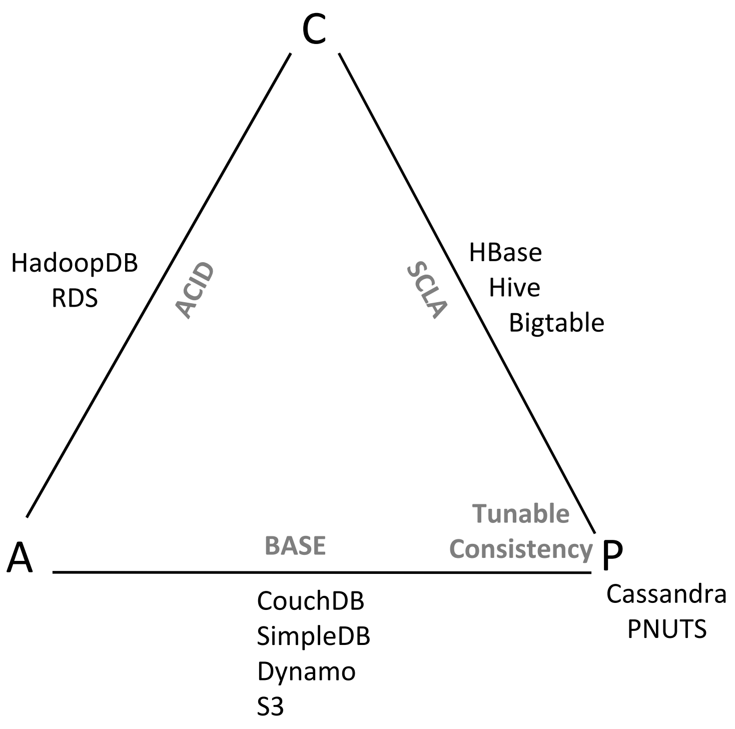

Figure 9.7: Classification based on CAP

mode and other degrees in between. For instance a user can set the consistency level of a write operation to high (means update all replicas of data). In that case, the user gains strong consistency but must wait until all replicas are updated. If any node storing a replica of the specfied data does not respond to the system, the operation fails. For more examples and discussion about this see sections 6.3 and 7.3

9.6 Data Access and Query Model

Data querying capabilities in cloud systems varies a lot. Some systems provide querying options on a single object such as in S3. Other systems provide querying capabilities on a single container of objects (or table of records) such as SimpleDB, HBase, and PNUTS. At the end of the spectrum, systems provide more sophisticated operations like joins and aggregations. Figure 9.8 illustrates the classification of cloud data storage systems according to query capabilities.

Amazon storage services based on Dynamo provide a web based interface, REST API, SOAP API, and language specific APIs such as Java, PHP, C#. They do not support a SQL like query language except for SimpleDB. SimpleDB provides a select statement that allows querying data from one domain. Amazon's RDS provide the full query support of the underlying Oracle and MySQL clusters.

In Google's Bigtable, data can be queried using the Bigtable's API and Google's MapReduce Framework. The API provides operations such as searching for a key value match or iterating over data. MapReduce is used to generate and modify data in Bigtable. Google's Storage for Developers provides GSUtil, a command line interface, and also supports REST API.

Hadoop based Hive provides an SQL like query language, HiveQL. HiveQL supports schema definition and data querying. Outer, equi, and left semi joins are also supported. MapReduce jobs can be used from inside queries. As for client support, Hive provides command line tool and supports JDBC/ODBC and Thrift APIs. HBase provides an SQL like query language called HBql which is still a work in progress. Joins are not allowed. However, HBase supports running MapReduce jobs against tables. Data can be also accessed using Avro, Thrift, REST and several language specific APIs. A system that tries to combine the best of relational and MapReduce worlds is HadoopDB. It provides a parallel DB front end to allow users to query data using HiveQL. Both MapReduce jobs and joins are supported. Yahoo!'s PNUTS does not support any SQL like query language. However data can be accessed using REST API, and many other language specific APIs such as Java and PHP. PNUTS does not support joins or MapReduce jobs. Facebook's Cassandra which is based on Hadoop provides a Thrift API to access data. Other Hadoop functionalities such as Pig, Hive, and MapReduce can be used against Cassandra DB. Cassandra also provides a SQL-like query language called CQL. Joins are not supported in Cassandra. Document oriented CouchDB provides REST and HTTP/JSON APIs. It also supports using MapReduce functions to create indices and perform other operations on data. Table 9.6 summarizes this comparison.

System	QueryL	MapReduce	Join	Languages/API
S3	no	no	no	REST, SOAP Java, Ruby PHP, C#
SimpleDB	SQL-like	no	no	REST, SOAP Java, Ruby PHP, C#
RDS	SQL	no	yes	REST, SOAP Java, Ruby PHP, C#
Bigtable	no	yes	no	GoogleAPI Java, Python
GSfD	no	no	no	REST, GSU-til
Hive	HiveQL Jaql Pig	yes	yes	Thrift JDBC/ODBC
HBase	HBql	yes	no	REST, Avro Thrift, JDBC
HadoopDB	HiveQL	yes	yes	Thrift JDBC/ODBC
PNUTS	no	yes	no	REST, Java C++, PHP
Cassandra	CQL	yes	no	Thrift
CouchDB	no	yes	no	REST HTTP/JSON

Table 9.4: Query models and supported languages and APIs of different cloud storage systems

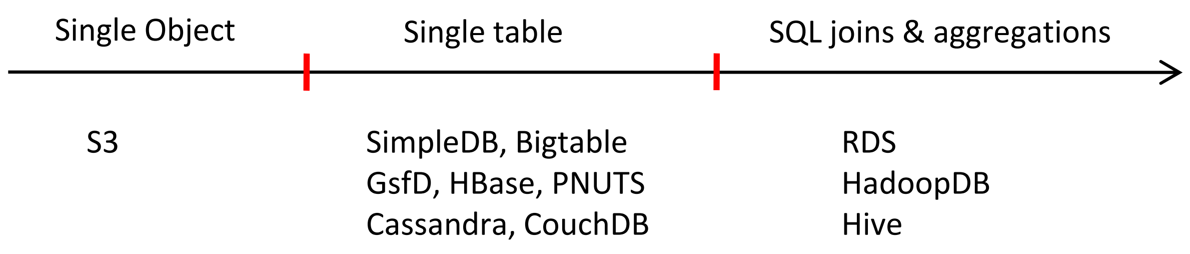

Figure 9.8: Classification based on query model

Chapter 10

Conclusions and Future Work

The landscape of data management systems has changed during the last ten years. A new generation of database applications that exploit the advantages of the cloud is emerging. We presented a comprehensive survey and classification of cloud data management approaches. This work covers a variety of systems developed by industry leaders such as Google's Bigtable or by academia such as HadoopDB, from the database research group at Yale University.

Some data management systems in the cloud are provided as web services or stand alone systems. Other data management systems are made of components that can be deployed and replaced depending on the application needs and workload. In most cases components developed by one vendor complement each other to satisfy one application storage needs. Data partitioning and replication are used by cloud data management systems to achieve scalability and availability. Since data partitioning is a state of fact in the cloud, this leads as explained by the CAP theorem to the trade off between consistency and availability. Consistency is not an all or nothing proposition in the cloud. Several levels of consistency are supported and some systems allow users to tune consistency on the request level. Non relational key value store is the prominent data model in the cloud. This new model provides more flexibility and scalability. It also provides more efficient data processing through MapReduce.

Cloud data management is a work in progress where new strategies, features, and components are still being developed. Next we list some challenges and research directions in cloud data management [Oez10, LYCL11]:

- Query processing and optimization:
  Parallel database techniques such as indices and optimization can be used to improve the performance and support more complex data analysis operations. Another direction is the deployment of non centralized query execution such as peer to peer techniques [JX09].

- Automatic data management:
  This includes automatic management of partitioning and replication to handle load variations and consider the diversity of nodes capacities and their performance levels .

- Data security and privacy:
  One direction is the transparent deployment of data encryption techniques with

the ability to perform operations on encrypted data. Another is the deployment of privacy preserving query processing techniques [HXRC11, TSWZ11]

- Declarative programming languages:
  Distributed data management on large scales using MapReduce is still hard [Oez10]. Ongoing research is to find a replacement that increases the ease of development and keeps the performance benefits that MapReduce already provides. A data centric declarative programming language was provided by Berkeley Orders Of Magnitude project (BOOM) [ACC+10].

Bibliography

[ABPA$^+$09] Abouzeid, A.; Bajda-Pawlikowski, K.; Abadi, D.; Silberschatz, A.; Rasin, A.: Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proc. VLDB Endow.*, Band 2, S. 922–933, August 2009.

[ABPH$^+$10] Abouzied, A.; Bajda-Pawlikowski, K.; Huang, J.; Abadi, D. J.; Silberschatz, A.: Hadoopdb in action: building real world applications. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, S. 1111–1114. ACM, New York, NY, USA, 2010.

[ACC$^+$10] Alvaro, P.; Condie, T.; Conway, N.; Elmeleegy, K.; Hellerstein, J. M.; Sears, R.: Boom analytics: exploring data-centric, declarative programming for the cloud. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, S. 223–236. ACM, New York, NY, USA, 2010.

[Ado11a] Amazon rds documentation, June 2011.

[Ado11b] Amazon s3 technical documentation, May 2011.

[Ado11c] Amazon simpledb technical documentation, June 2011.

[AFG$^+$10] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D. et al.: A view of cloud computing. *Commun. ACM*, Band 53, S. 50–58, April 2010.

[AS10] Avi Silberschatz, S. S., H. F. K.: *Database System Concepts Fifth Edition.* McGraw-Hill, 2010.

[AS311] Amazon s3 design principles, May 2011.

[BBG11] Buyya, R.; Broberg, J.; Gościński, A.: *Cloud Computing: Principles and Paradigms.* John Wiley & Sons, Inc., Hoboken, New Jersey, 2011.

[big11] Bigtable, August 2011.

[BM09] Brunette, G.; Mogull, R.: Security guidance for critical areas of focus in cloud computing v2. 1. Technischer Bericht, Cloud Security Alliance, December 2009.

[Bor07] Borthakur, D.: *The Hadoop Distributed File System: Architecture and Design.* The Apache Software Foundation, 2007.

[Bor08] Borthakur, D.: Hdfs architecture guide. The Apache Software Foundation., 2008.

[Bro11] Browne, J.: Brewer's cap theorem, August 2011.

[cas11a] Apache cassandra quick tour, July 2011.

[Cas11b] Cassandra wiki, July 2011.

[CDG$^+$06] Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W. C.; Wallach, D. A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R. E.: Bigtable: A distributed storage system for structured data. In *IN PROCEEDINGS OF THE 7TH CONFERENCE ON USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION - VOLUME 7*, S. 205–218, 2006.

[CGJ$^+$09] Chow, R.; Golle, P.; Jakobsson, M.; Shi, E.; Staddon, J.; Masuoka, R.; Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, S. 85–90. ACM, New York, NY, USA, 2009.

[Cou11a] The apache couchdb project, July 2011.

[Cou11b] Couchdb security features overview, July 2011.

[Cou11c] Exploring couchdb a document-oriented database for web applications. Technischer Bericht, July 2011.

[CRS$^+$08] Cooper, B. F.; Ramakrishnan, R.; Srivastava, U.; Silberstein, A.; Bohannon, P.; Jacobsen, H.-A.; Puz, N.; Weaver, D. et al.: Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, Band 1, S. 1277–1288, August 2008.

[Dar11] Darmstadt, F. S.: Careless behaviour of cloud users leads to crucial security threats. Technischer Bericht, June 2011.

[DE10] David Erb, M. S.: Google storage for developers, 2010.

[DG92] DeWitt, D.; Gray, J.: Parallel database systems: the future of high performance database systems. *Commun. ACM*, Band 35, S. 85–98, June 1992.

[DG04] Dean, J.; Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, S. 10–10. USENIX Association, Berkeley, CA, USA, 2004.

[DHJ$^+$07] DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Sivasubramanian, S.; Vosshall, P. et al.: Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, Band 41, S. 205–220, October 2007.

[FE10] Furht, B.; Escalante, A. (Hrsg.): *Hand Book of Cloud Computing.* Springer, 2010.

[Geo11] George, L.: Hbase architecture 101 - storage, July 2011.

[GGL03] Ghemawat, S.; Gobioff, H.; Leung, S.-T.: The google file system. *SIGOPS Oper. Syst. Rev.*, Band 37, Nr. 5, S. 29–43, 2003.

[GL02] Gilbert, S.; Lynch, N.: Brewer's conjecture and the feasibility of consistent available partition-tolerant web services. In *In ACM SIGACT News*, S. 2002, 2002.

[Hab11] Habeeb, M.: *A Developers Guide to Amazon SimpleDB.* Addison Wesley, 2011.

[had11] Hadoopdb an architectural hybrid of mapreduce and dbms technologies for analytical workloads., August 2011.

[HBA11] Hbase discretionary access control, July 2011.

[HBq11] Hbase query language hbql, July 2011.

[HBS11] Secure hbase: Access controls, July 2011.

[Hel07] Helland, P.: Life beyond distributed transactions: an apostate's opinion. In *CIDR*, S. 132–141, 2007.

[Hew10] Hewitt, E.: *Cassandra The Definitive Guide.* O Reilly Media, Inc, Novemober 2010.

[Hiv11] Hive indices, August 2011.

[HLM11] Hive language manual, July 2011.

[Hol11] Holt, B.: *Scaling CouchDB Replication, Clustering, and Administration.* OReilly Media, Inc, April 2011.

[HXRC11] Hu, H.; Xu, J.; Ren, C.; Choi, B.: Processing private queries over untrusted data cloud through privacy homomorphism. *Data Engineering, International Conference on*, Band 0, S. 601–612, 2011.

[IBM11] Jaql overview, August 2011.

[IF01] Ian Foster, S. T., C. K.: The anatomy of the grid enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, S. 200–222, 2001.

[JCAS10] J. Chris Anderson, J. L.; Slater, N.: *CouchDB The Definitive Guide.* OReilly Media, Inc, 2010.

[JDU97] Jeffrey D. Ullman, J. W.: *A first course in database systems.* Prentice Hall, 1997.

[JG11] Jansen, W.; Grance, T.: Guidelines on security and privacy in public cloud computing. Technischer Bericht, NIST, January 2011. Draft Special Publication 800-144. Available at `http://csrc.nist.gov/publications/drafts/800-144/Draft-SP-800-144_cloud-computing.pdf`.

[JMF09] Jha, S.; Merzky, A.; Fox, G.: Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes. *Concurr. Comput. : Pract. Exper.*, Band 21, S. 1087–1108, June 2009.

[JX09] Jurczyk, P.; Xiong, L.: Dynamic query processing for p2p data services in the cloud. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications*, DEXA '09, S. 396–411. Springer-Verlag, Berlin, Heidelberg, 2009.

[KFZ05] Keahey, K.; Foster, I.; Zhang, T. F. X.: Virtual workspaces: Achieving quality of service and quality. In *of Life in the Grid. Scientific Programming Journal*, S. 265–276, 2005.

[LA02] Lance Ashdown, S. B. e. a., C. B.: *Oracle9i Database Concepts.* Oracle Corporation., 2002.

[Len09] Lennon, J.: *Beginning CouchDB.* Springer-Verlag New York, Inc, 2009.

[LM10] Lakshman, A.; Malik, P.: Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, Band 44, S. 35–40, April 2010.

[LYCL11] Li, M.; Yu, S.; Cao, N.; Lou, W.: Authorized private keyword search over encrypted data in cloud computing. In *ICDCS*, S. 383–392, 2011.

[MG11] Mell, P.; Grance, T.: The nist definition of cloud computing. Technischer Bericht, NIST, January 2011. Draft Special Publication 800-145. Available at `http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf`.

[Mye09] Myerson, J. M.: Cloud computing versus grid computing. *IBMdeveloper-Works*, 2009.

[Nol11] Noll, M. G.: Running hadoop on ubuntu linux (multi-node cluster), August 2011.

[Oez10] Oezsu, V. P., M. T.: *Principles of Distributed Database Systems, Third Edition.* Prentice Hall, 2010.

[Par11] Partition (database), August 2011.

[PC10] Prabhakar Chaganti, R. H.: *Amazon SimpleDB Developer Guide Scale your application's database on the cloud using Amazon SimpleDB.* Packt Publishing Ltd, 2010.

[Pon10] Ponemon, L.: Security of cloud computing users a study of practitioners in the us & europe. Technischer Bericht, Ponemon Institute, May 2010.

[rds11] Amazon relational database service getting started guide, April 2011.

[RSRS98] Ramaswamy, C.; Sandhu, R.; Ramaswamy, R.; S, R.: Role-based access control features in commercial database management systems. In *In Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, S. 503–511, 1998.

[SAD$^+$10] Stonebraker, M.; Abadi, D. J.; DeWitt, D. J.; Madden, S.; Paulson, E.; Pavlo, A.; Rasin, A.: Mapreduce and parallel dbmss: Friends or foes? CACM, 53(1), January 2010.

[Tea09] Team, F. D.: Data warehousing & analytics on hadoop, 2009.

[ter] *Manual: Introduction to Teradata RDBMS.*

[TSJ$^+$09] Thusoo, A.; Sarma, J. S.; Jain, N.; Shao, Z.; Chakka, P.; Anthony, S.; Liu, H.; Wyckoff, P. et al.: Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, Band 2, S. 1626–1629, August 2009.

[TSWZ11] Tian, X.; Sha, C.; Wang, X.; Zhou, A.: Privacy preserving query processing on secret share based data storage. In Yu, J.; Kim, M.; Unland, R. (Hrsg.): *Database Systems for Advanced Applications*, Lecture Notes in Computer Science, Band 6587, S. 108–122. Springer Berlin / Heidelberg, 2011.

[VMMS11] Victor Mendez Munoz, A. G., M. K.; Salt, J.: On the economics of huge requirements of the mass storage a case study of the agata project. *CLOSER International Conference on Cloud Computing and Services Science*, April 2011.

[Vog11] Vogels, W.: All things distributed amazon's dynamo, August 2011.

[VZ09] Voas, J.; Zhang, J.: Cloud computing: New wine or just a new bottle? *IT Professional*, Band 11, Nr. 2, S. 15 –17, march-april 2009.

[Whi09] White, T.: *Hadoop: The Definitive Guide.* OReilly Media, Inc, 2009.

[YYLC10] Yuan, D.; Yang, Y.; Liu, X.; Chen, J.: A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *IPDPS*, S. 1–12, 2010.

[zoo11] Apache zookeeper, July 2011.

Deceleration of Authorship

I hereby declare that I am the sole author of this thesis and used nothing but the specified resources and means.

Magdeburg, den September 23, 2011

Siba Mohammad