https://doi.org/10.31449/inf.v43i1.2650 Informatica **43** (2019) 313–319 **313**

Consistency in Cloud-based Database Systems

Zohra Mahfoud
USTHB University, Algeria
E-mail: mahfoud.zohra@yahoo.fr

Nadia Nouali-Taboudjemat
CERIST Research Center, Algeria
E-mail: nnouali@cerist.dz

Keywords: cloud computing, consistency, distributed databases, relational databases, No-SQL, CAP

Received: July 15, 2019

Cloud computing covers the large spectrum of services available on the internet. Cloud services use replication to ensure high availability. Within database replication, various copies of the same data item are stored in different sites, this situation requires managing the consistency of the multiple copies. In fact, the requirement for consistency level can be different according to application natures and other metrics; a delay of some minutes in visualizing latest posts in social networks can be tolerated, while some seconds can make a loss of a bid in an auction system. Wide variety of database management systems are used actually by cloud services, they support different levels of consistency to meet the diversity of needs.

This paper draws a presentation of the main characteristics of cloud computing and data management systems and describes different consistency models. Then it discusses the most famous cloud-based database management systems from the point of view of their data and consistency models.

Povzetek: Prispevek analizira podatkovna skladišča v oblakih predvsem s stališča konsistentnosti.

1 Introduction

Cloud computing refers to the large spectrum of services available on the internet. These services manage big quantities of data with high availability, scalability and elasticity. Providing availability requires databases replication. Replication permits the creation and the management of various copies of data items stored in different sites.

Consistency concerns the freshness of data and indicates if copies are the same in the different sites and witch version of data is returned by queries. In fact, consistency does not have the same importance for all the applications and the users. In social networks, a delay of minutes or even hours in visualizing the posts may not be a problem. Whilst for an auction system, a delay of few seconds can cause the loss of a bid.

Various systems are proposed to manage data for cloud services; they provide a variety of consistency models and use different data models which are based either on the classical relational model or on No-SQL models.

This paper discusses consistency in cloud-based database management systems. The reminder of this paper is organized as follows. Section 2 presents the main characteristics of cloud computing. Section 3 presents databases models in cloud. Section 4 explains the concept of consistency and the dilemma posed by the CAP theorem; it presents also the different levels and models of consistency. Section 5 presents some famous cloud systems and describes the implemented models of data and consistency. Section 6 concludes the paper.

2 Cloud computing

Cloud computing includes all forms of services available on the Internet; that are classified as software, platform or infrastructure as a service. Cloud services attract increasingly individuals, startups and big companies by the fascinating characteristics offered such as Availability, Scalability and Elasticity [1-4].

2.1 Availability

Queries must be answered within a reasonable time even there is a huge load of work or under any type of failures.

Availability is guaranteed by **replicating** databases, i.e. creating multiple replicas (copies) of the database and storing them at different sites. Replication can be full when it concerns the entire database or partial when it concerns just a part of the database (one or more tables, one or more partition) [5, 6].

Typically, Replicas are used to increase the **availability** of the system. They permit i) to decrease the latency by distributing queries on different replicas, ii) to cache site failure by accessing other sites, and iii) to recover site failure as backups [7].

Synchronous replication control algorithms assume that replicas are the same all the time. But this is not possible physically, so outdated replicas are made not

accessible until they are synchronized. In contrary, asynchronous algorithms allow accessing to divergent replicas that will finally converge [8, 9].

2.2 Scalability

This property is related to the capacity of providing large databases and managing their growing. Scalability is ensured by **partitioning** the database, i.e. devising the database into several disjoint partitions (fragments) that can be stored in different sites. Partitioning database offers the possibility of incrementing infinitely the capacity of storage by adding new hardware [6].

Partitioning has two general types: **Vertical** and **Horizontal**. In vertical type each partition contains a set of columns of the database; while in horizontal type (called communally sharding) the database is divided into sets of rows. The two types of partitioning can be combined to obtain a better strategy [10].

2.3 Elasticity

Elasticity called also elastic scalability refers to the flexibility of scaling up and down quickly in order to support the change of the requirements. Elasticity is the most important property that attracts companies to the cloud as it permits to pay accurately according to use.

3 Database models in cloud computing

Data storage in the cloud uses both of the classical relational model and the new No-SQL architectures.

Relational Databases: These databases respect the classical relational model proposed by E.F.Codd [11]. Relational databases structure data into tables composed of columns and rows, with a unique primary key and possible foreign keys. They provide the CRUD (**C**reate, **R**ead, **U**pdate and **D**elete) basic operations, and also operations across several tables.

Relational databases dominate the market of databases for more than twenty years; this success is due to its stability and consistency. These characteristics are guaranteed via transactional mechanisms that are implemented by the ACID (Atomicity, Consistency, Isolation and Durability) properties [12].

SQL (Structured Query Language) is the most used for requesting and maintaining relational databases.

Database Management Systems (DBMS) are responsible to store, retrieve, secure, replicate and realize backups of databases. The most famous Relational Databases Management Systems (RDBMS) are: Oracle, MySQL, Microsoft SQL Server, Postgres.

No-SQL Databases:

No-SQL databases ('Not only SQL' or 'Not relational') is a family of databases or more appropriate data stores that support all schemas of data characterized as structured, semi-structured and unstructured. No-SQL databases provide a high level of availability, scalability and elasticity. These features make No-SQL databases increasingly used for big data and qualified as the databases for the next-generation of web applications [13, 14].

Unlike the relational databases, No-SQL databases do not have a unified data model. Also, the level of operations is different; some systems provide only simple read-write operations, while others support more advanced operations. These differences lead to more than one hundred No-SQL databases which are principally classified into four categories [2, 3, 15, 23]:

i. **Key-value Databases:** this model permits to store all schemas of data, as (key, value) pairs. A unique key is assigned to every value and permits to access the value. The value can be a simple data item, or a set of key-value pairs.
   Example of key-value databases are: App Engine Data Store, Redis, Riak, etc.
ii. **Column-oriented Databases:** this model holds structured data in tables that are organized in rows like in relational databases. The difference is that columns can be different from one row to another. Also, a column can also regroup a set of columns. In other hand, operations across tables are not supported.
   Examples of column-oriented databases are: Google BigTable, Cassandra, etc.
iii. **Document-based Databases:** this model is used to store unstructured data, where keys addressed generally XML (eXtensible Markup Language) or JSON (JavaScript Object Notation) documents. No restrictions on data type or documents length are imposed.
   Examples of Document-based stores are: CoucheDB, MongoDB, RavenDB, etc.
iv. **Graph Databases:** this model allows storing data and relationships between them using graphs; nodes store data and arcs store relationships. The support of dynamic relationship makes this model the most appropriate for social networks.
   Examples of graph databases are: Neo4j, HyperGraphDB, Infinite Graph, etc.

We stress that all No-SQL architectures are basically based on the key-value model.

4 Consistency

Mutual consistency or simply consistency refers how to propagate updates between the different copies of replicated items. It concerns the state of data items in different sites; if they are the same or not. Also, how users see data items, if they see the same value, or they are allowed to see different values [6, 22].

Figure 01 shows a cloud system where the item X is duplicated in three sites. In an ideal situation, all copies of X have the same value (V1=V2=V3), this classical level of consistency is the most suitable, but it is hard to implement in distributed systems as it is proved by the CAP theorem as explained below.

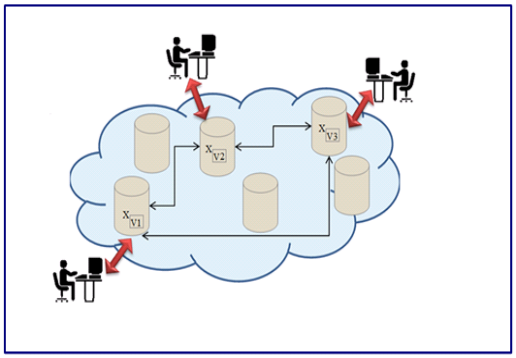

Figure 1: Distributed system with replication.

We mention here that the cloud is considered as a large geo-distributed system; data is largely replicated to ensure availability in the case of concurrent queries and recovery in case of failure. The different replicas can be located in the same datacenter or over different geo-distributed datacenters that can located in different continents; in this case the communication between replicas is very expensive.

4.1 CAP theorem

The CAP theorem (Figure 02) states that shared-data systems can ensure at most two of three properties: Consistency, Availability, and Partition tolerance at the same time [16, 17].

Figure 2: CAP theorem.

Choosing two properties between Availability, Partitioning tolerance and Consistency in the cloud is not easy; Availability and Partitioning are primordial and Consistency is vital for reliability. Cloud systems do not avoid absolutely one of the three properties, and propose generally a compromise between the three properties, which leads to support degraded levels of each one. A description of consistency levels is presented by the next section.

PACELC [18] extends CAP and states that the compromise is not all the time between Availability and Partitioning and Consistency; during network Partition (P) the compromise is between Availability (A) and Consistency (C). Else (E), the compromise is between Latency (L) and Consistency (C). The latency measures the delay of getting a reply.

4.2 Consistency levels

Consistency levels are influenced by the type of replication control protocol; i) Synchronous protocols propagate updates to all the replicas at the same time and in the same order. These protocols present strong consistency (immediate consistency). ii) Asynchronous protocols allow updating one replica while other outdated replicas are still accessible. iii) Hybrid protocols propagate updates synchronously between some replicas. Asynchronous and hybrid protocols present different levels of consistency according to which replicas are accessible, and the number of replicas that must be written and read before replying to queries [18, 19]. Quorum-based systems are proposed to achieve strong consistency by using the majority of replicas; Paxos is the most known protocol in this area [47].

The level of consistency is chosen according to the system nature and user's needs. Transactional systems like they proposed to book a flight ticket, buy an item, or send a bid are cases where data must be treated with strong consistency; an inconsistency of few seconds may make a loss. Social networks are examples of applications that tolerate weak consistency; a delay in visualizing the latest posts can be accepted.

4.3 Consistency models

A variety of consistency models degraded from strong to weak consistency are proposed in the literature, the main models are [19, 20, 21, 22, 40]:

Strict consistency (Atomic consistency, Linearizability), is the strictest model of consistency; updates are propagated between replicas at the same order according to the real time. Also, reads return the last written values.

Sequential consistency (Serializability): updates are ordered according to a logical order applied by all the replicas, this order can be different from the real order. Reads return the last values written according to the logical order.

The eventual consistency model ensures that all replicas will eventually become consistent even if requests can read inconsistent values. Different variants of this model are distinguished according to the techniques used to manage the inconsistent window:

Causal Consistency is a variation of the eventual consistency, where only causally related operations are ordered.

Read-your-writes consistency is a case of causal consistency where users access always his updates, or a newer version, and never access an older version.

Session consistency implements the read-your-writes consistency model during the session.

The bounded staleness consistency model tolerates reading stale values under some conditions such as bounding staleness by a specific period of time delta. This condition is satisfied by propagating updates within delta.

In Configurable consistency (Tunable consistency) the user configures the number of replicas accessed synchronously. Here, the consistency level depends on the percentage of the replicas requested synchronously; strong consistency is reached if the number of replicas for read (R) and write (W) overlap (R+W>=N), N is the total number of replicas.

5 Consistency levels in cloud systems

Wide variety of database management systems are used actually by cloud services. This section presents the most famous of them from the point of view of their data and consistency models [24].

5.1 Amazon propositions

Amazon has several propositions: Simple Storage Service (S3) [25,29], SimpleDB [26] and DynamoDB [27, 28] are No-SQL databases that provide high availability and scalability. Amazon Aurora [30, 31] is a relational databases management system that provides strong consistency.

S3 is designed to store large data in buckets: a bucket is organized as a key-value store, values are generally objects that represent data files or folders used to organize data files, folders can be arranged hierarchically. S3 offers simple operations to create, write, read and delete buckets, keys and objects. S3 uses automatic Cross-region replication that allows asynchronous copying of objects across buckets in different Regions. This strategy provides eventual consistency model.

SimpleDB arranges structured data in domains which consist of items; items are composed of pairs of (attribute, value); value can contain multiple data. SimpleDB offers operations for creating, writing, reading and deleting a domain or an attribute. Operations manipulate one or various items of the same domain. Eventual consistency is proposed by default; however it is possible to choose the strong consistency.

Dynamo uses tables of items, each item contains one or more attributes. An attribute is composed of (key, value) pairs. Dynamo provides several operations to create, write, read and dele table, item and attribute; which permit to manipulate one or various items of the same table. Initially, dynamo offers eventual consistency; a quorum that preserves availability and scalability is addressed to fulfill operations. However, dynamo makes it is possible to achieve strong consistency by configuring the number of requested replicas.

Data models in simpleDB and Dynamo have the structure of tables. Although, they are not classified as column-family store because they have simple columns and not super column families.

Amazon Aurora is a cloud-based relational databases management system proposed by Amazon Relational Database Service (RDS). Aurora is built on a MySQL engine and it is compatible with PostgreSQL. It provides better availability and scalability comparing to classical databases engines on RDS. Aurora guarantees strong consistency by supporting a quorum protocol.

5.2 Google propositions

In its turn, Google published several cloud-based systems [32] like BigTable [33], Megastore [34], Spanner [35], Cloud SQL [36], and Cloud datastore [37].

Bigtable stores data in massive tables. Each table is organized in rows that are accessed by primary keys and they contain a set of column-families which can differ from a row to another. A column-family regroups related columns and each column contains a single value for a row. This model allows storing versioned data in columns regrouped in a column family. Operations concern atomic single-row and a quorum protocol based on Paxos algorithm is implemented to provide strong consistency for write operations, read operations can get stale data if an update is on progress.

Bigtable is designed to store very large amounts of data; Google uses it in many applications like: Google Analytics, Earth, Map and Personalized Search.

Megastore uses schemas of tables to organize data; a table contains a set of entities that are characterized by a set of properties. Megastore defines entity groups that are sets of related tables based on Bigtable. Megastore provides transactions with full ACID semantics that can concern data through several tables of the same entity group, not just data of the same table like the majority of No-SQL databases. Like Bigtable, Megastore uses Paxos protocol to provide strong consistency; for each write operation, a majority of replicas across geographically distributed datacenters is requested; this strategy increases the system latency.

Megastore is proposed to build interactive applications; it is used by well-known Google applications as: AppEngine, Gmail, Calendar and Android Market.

Spanner is a key-value database created to fix the weaknesses of megastore in term of latency. Like megastore, spanner organizes data in schematized semi-relational tables, uses timestamp for versioning data and use a like SQL-based query language. Spanner propose an excellent support of transactions with full ACID properties, it provide strong consistency for distributed transactions across geographically replicated datacenters; this is achieved by executing a combination of the two-phase-commit protocol and Paxos protocol. Spanner is largely used within Google's datacenters infrastructures.

Cloud SQL is a RDBMS based on MySQL that provides classically immediate consistency.

Cloud Datastore is a Document store that organizes data on kinds of entities; each entity is accessed by a key and composed of a set of properties storing values that can have different types even for the same properties. Cloud Datastore use Multi-Master replication based on Paxos. Queries are configured to obtain immediate or eventual consistency.

5.3 Microsoft propositions

Microsoft has also several propositions: Microsoft Azure Table storage [38], Microsoft Azure DocumentDB [39] and Microsoft Azure SQL Database [41].

Microsoft Azure Table storage is a key-value store that stocks large amounts of data in tables. Each table contains a set of entities: an entity is composed of a primary key and a set of properties. Table storage provides strong consistency, and permits to achieve transactions with ACID properties across tables of the same partition.

Microsoft Azure Cosmos DB gathers multiple data models that include key-value, table, columnar, document and graph data models. It offers a configurable consistency model that presents five levels: strong, bounded-staleness, session, consistent prefix, and eventual. Strong consistency is associated only with one Azure region; it uses a linearizability based on a majority of replicas. The other levels are designed to reinforce avalability across different regions.

Microsoft Azure SQL Database is a RDBMS in the cloud built on the Microsoft SQL Server engine that supports full ACID properties of relational databases and uses a quorum-based algorithm that provides an acceptable consistency level with high availability.

5.4 Others solutions

5.4.1 Cassandra

Cassandra [42, 43] is an open source column family store proposed by Facebook for managing massive amounts of data. Cassandra is inspired from Google BigTable and Amazon DynamoDB.

The data model of Cassandra uses column families (tables) that regroup rows; each row in a table is composed of a key and a list of columns or super columns. A column is composed of a key, a value and a timestamp. A super column is a column family that regroups columns.

Cassandra proposes panoply of consistency models that can be configured at operation level. These levels are differentiated according to the requested replicas and theirs locations; the level ALL involves all the replicas of the cluster. The levels: One, TWO and THREE involve at least one, two and three replica (s), respectively. The level QUORUM involves a quorum of replicas of the cluster. According to the nodes locations, the following levels are defined: EACH_QUORUM requires a quorum of replicas in all data centers. LOCAL_QUORUM requires a quorum of replicas in the same data center. And, LOCAL_ONE requires one replica at least in the local data center. In addition, Cassandra proposes the SERIAL level that uses linearizable consistency for achieving lightweight transactions.

LOCAL_SERIAL concerns one datacenter. The levels listed above are common to read and write operations. The ANY level is specific only to write operations; it permits to execute a write operation even if no required replica is available; the operation writes hints for downed nodes on others nodes. The changes will be sent to downed nodes when they recovered.

The consistency level is determined by the number of replicas solicited for the read (R) and write (W) operations; if it overlaps the total number of replicas (N) the consistency is strong (R+W>=N), otherwise the consistency is weak.

5.4.2 PNUTS

PNUTS [44, 45] proposed by Yahoo! exposes a simple relational model with flexible schema. PNUTS organizes data into tables of records with attributes that can store any type of data. PNUTS offers various operations like Update, delete, selection of one or more items from a single table.

PNUTS proposes a per-record timeline consistency model that offers a consistent view of data to the user; a master replica is nominated to each record, this replica receives all the updates concerning the record and propagates the updates to other replicas in the same order.

This consistency can be configured; the weak level is ensured by the options: Read-any, Read-critical (required version), Test-and-set-write. However, the options: Read-latest ensures strong consistency.

5.4.3 Neo4j

Neo4j [46] is a graph based No-SQL databases that models data using nodes and relationships. Nodes are used to represent entities, they can be labeled and contain properties. Relationships present relations between nodes and can also contain properties.

Neo4j supports full ACID properties and implements causal consistency to provide an acceptable level of consistency.

6 Conclusion

Availability, scalability and elasticity are the success keys of cloud computing. At the storage level, these properties are guaranteed by partitioning and replicating databases.

Initially, cloud systems used the relational model that dominated the market of databases for more than twenty years. This model is known by its stability and consistency, which are guaranteed using transactional mechanisms. However, these mechanisms make the relational model very rigid and lack required availability and scalability. In order to meet the cloud needs, a new generation of relational cloud-based systems that supports more availability and scalability appeared. Several applications in cloud prefer No-SQL models that are proposed initially as simple key-value pairs that avoid all types of constraints. Bit by bit, No-SQL Databases use more organized models and integrate some transactional mechanism. Nevertheless, they still more flexible comparing to relational model.

In the consistency side and as it is difficult to ensure availability with strong consistency in large geo-distributed systems, cloud systems implement different consistency models to ensure the best compromise between availability and consistency. In addition, a lot of systems propose a tunable consistency that offers the possibility to choose between numerous proposed models.

7 References

[1] S. Sakr, A. Liu, D. Batista, M. Alomari (2011). “A Survey of Large Scale Data Management Approaches in Cloud Environments”. IEEE

Communications Surveys and Tutorials. 13(3): 311- 336, https://doi.org/10.1109/SURV.2011.032211.00087.

[2] A. Elzeiny, A. Abo Elfetouh ,and A Riad (2013). "Cloud Storage: A Survey". International Journal of Emerging Trends & Technology in Computer Science. Vol. 2, Issue 4, ISSN 2278-6856: 342-349.

[3] M. Siba, S. Breß, and E. Schallehn (2012). "Cloud Data Management: A Short Overview and Comparison of Current Approaches". Grundlagen von Datenbanken.

[4] D. Kossmann, T. Kraska, S. Loesing (2010). "An evaluation of alternative architectures for transaction processing in the cloud". SIGMOD Conference : 579-590. https://doi.org/10.1145/1807167.1807231.

[5] Saeed K. Rahimi , By (author) Frank S. Haug (2010). "Distributed Database Management Systems A Practical Approach". Wiley-IEEE Computer Society. https://doi.org/10.1002/9780470602379.

[6] M.T Özsu, P. Valduriez (2011). "Principles of Distributed Database Systems". Springer Science+ Business Media, 3rd ed. https://doi.org/10.1007/978-1-4419-8834-8.

[7] V.K. Pallaw (2010). "Concept of Database Management Systems". Asian Books Pvt. Ltd. ISBN : 978-81-8412-119-3.

[8] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso (2000). "Understanding Replication in Databases and Distributed Systems". IEEE International Conference on Distributed Computing Systems: 464-474.

[9] M. Wiesmann, F. Pedone, A. Schiper (2000). "Database Replication Techniques: a Three Parameter Classification". The 19th IEEE Symposium on Reliable Distributed Systems: 206-215.

[10] SH. Navathe, S. Ceri, G. Wiederhold, J. Dou (1984). "Vertical Partitioning Algorithms for Database Design". ACM Transactions on Database Systems, Vol. 9, No.4. https://doi.org/10.1145/1994.2209.

[11] Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM. 13 (6): 377–387. https://doi.org/10.1145/362384.362685.

[12] J. Gray (1981). "The Transaction Concept: Virtues and Limitations". The 7th VLDB, Cannes: 144-154.

[13] F. Bugiotti, L. Cabibbo, P. Atzeni, R. Torlone (2014). "Database Design for NoSQL Systems". 223-231.

[14] P. J. Sadalage and M. J. Fowler (2012). "NoSQL Distilled". Addison-Wesley.

[15] G. Harrison (2015). "Next Generation Databases: NoSQL, NewSQL, and Big Data". Apress, ISBN(e): 978-1-4842-1329-2.

[16] E. A. Brewer (2000). "Towards Robust Distributed Systems". PODC (Invited Talk) :7.

[17] N. Lynch and S. Gilbert (2002). "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". ACM SIGACT News, Vol. 33 Issue 2: 51-59. https://doi.org/10.1145/564585.564601.

[18] Daniel J. Abadi (2012). "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story". Journal of computer, 45(2):37–42. https://doi.org/10.1109/MC.2012.33.

[19] S.P. Kumar (2016). "Adaptive Consistency Protocols for Replicated Data in Modern Storage Systems with a High Degree of Elasticity". PHD thesis, Conservatoire national des arts et métiers, Paris, France.

[20] D. Mosberger (1993). "Memory Consistency Models". ACM SIGOPS Operating Systems Review Homepage archive. Vol. 27, Issue 1 : 18-26 https://doi.org/10.1145/160551.160553.

[21] Adve, Sarita V and Gharachorloo, Kourosh (1996). "Shared Memory Consistency Models: A Tutorial". Journal of Computer, Vol. 29, Issue 12: 66-76. https://doi.org/10.1109/2.546611

[22] W. Vogels (2009). "Eventually consistent". Communications of the ACM, Vol. 52, n.1: 40-44, https://doi.org/10.1145/1435417.1435432.

[23] IGI Global publications (2016). "Big Data: Concepts, Methodologies, Tools, and Applications". ISBN: 9781466698406.

[24] "DB-Engines Ranking", Available Online [Aug2018]: http://db-engines.com/en/ranking/.

[25] "Amazon Simple Storage Service Documentation". Available Online [Aug2018]: https://aws.amazon.com/documentation/s3/.

[26] "Amazon SimpleDB Documentation". Available Online [Aug2018]: https://aws.amazon.com/documentation/simpledb/.

[27] "Amazon DynamoDB Documentation". Available Online [Aug2018]: https://aws.amazon.com/documentation/dynamodb/.

[28] G. DeCandia, D. Hastorun, M. Jampani, et al. (2007). "Dynamo: Amazon's highly available key-value store". SOSP:205–220. https://doi.org/10.1145/1294261.1294281.

[29] D. Bermbach and S. Tai (2011). "Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior". The 6th Workshop on Middleware for Service Oriented Computing. ACM. https://doi.org/10.1145/2093185.2093186.

[30] "Amazon Amazon Aurora". Available Online [Aug2018]: https://aws.amazon.com/rds/aurora/.

[31] "Amazon Relational Database Service Documentation". Available Online [Feb2017]: https://aws.amazon.com/documentation/rds/.

[32] "Google Cloud Platform: Cloud Storage Products". Available Online [Aug2018]: https://cloud.google.com/products/storage.

[33] F. Chang, J. Dean, S. Ghemawat, et al. (2008). "Bigtable: A Distributed Storage System for Structured Data". ACM TOCS 26.2, 4:1–4:26. https://doi.org/10.1145/1365815.1365816.

[34] J. Baker, C. Bond, J. Corbett et al. (2011). "Megastore: Providing Scalable, Highly Available Storage for Interactive Services". CIDR: 223–234.
[35] J. Corbett, J. Dean, M. Epstein, et al. (2012). "Spanner: Google's globally-distributed database". OSDI:251–264. DOI: 10.1145/2491245.
[36] "CLOUD SQL". Available Online [Feb2017]: https://cloud.google.com/sql/.
[37] "Google Cloud Datastore Documentation". Available Online [Aug2018]: https://cloud.google.com/datastore/docs/.
[38] B. Calder, J. Wang, A. Ogus et al. (2011). "Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency". The 23rd ACM Symposium on Operating Systems Principles: 23-26. Cascais, Portugal.
[39] "Azure Cosmos DB Documentation". Available Online [Aug2018]: https://docs.microsoft.com/en-us/azure/cosmos-db/.
[40] A. Singla, U. Ramachandran, and J. Hodgins (1997). "Temporal Notions of Synchronization and Consistency in Beehive". The 9th Annual ACM Symp. on Parallel Algorithms and Architectures: 211–220. https://doi.org/10.1145/258492.258513.
[41] "Microsoft Azure SQL Database". Available Online [Aug2018]: https://azure.microsoft.com/en-us/services/sql-database/
[42] "Apache Cassandra". Available Online [Aug2018]: http://cassandra.apache.org/
[43] A. Lakshman, P. Malik (2010). "Cassandra: a decentralized structured storage system". Operating Systems Review 44(2): 35-40. https://doi.org/10.1145/1773912.1773922.
[44] B. Cooper, R. Ramakrishnan, U. Srivastava (2008). "Pnuts: Yahoo!'s hosted data serving platform". PVLDB, 1(2):1277–1288. https://doi.org/10.14778/1454159.1454167.
[45] A. Silberstein, J. Chen, D. Lomax et al. (2012). "PNUTS in Flight: Web-Scale Data Serving at Yahoo". IEEE Internet Computing 16(1): 13-23 https://doi.org/10.1109/MIC.2011.142.
[46] "Neo4j". Available Online [Aug2018]: https://neo4j.com/
[47] L. Lamport (2002). "Paxos Made Simple, Fast, and Byzantine". OPODIS: 7-9