Future Generation Computer Systems 52 (2015) 13–21

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

BASE analysis of NoSQL database

Deka Ganesh Chandra

Regional Vocational Training Institute for Women, Tura, Meghalaya
Ministry of Skill Development and Entrepreneurship, Directorate General of Training, Government of India

HIGHLIGHTS

- Discussion on BASE features of some of the popular NoSQL.
- BASE (Basically Available, Soft State, Eventual consistency) analysis of NoSQL.
- ACID (Atomicity, Consistency, Isolation, and Durability) versus BASE.
- Discussion on NRW (Node, Read, Write) analysis of mostly used NoSQL.
- Discussion on Yahoo! Cloud Serving Benchmark (YCSB) and SandStorm Cloud benchmarking with a focus on NoSQL.

ARTICLE INFO

Article history:
Received 19 November 2012
Received in revised form
15 February 2015
Accepted 4 May 2015
Available online 18 May 2015

Keywords:
ACID
BASE
NoSQL
CAP theorem
Soft state

ABSTRACT

NoSQL databases are designed to address performance and scalability requirements of web based application which cannot be addressed by traditional relational databases. Due to their contrast in priorities and architecture to conventional relational databases using SQL, these databases are referred as "NoSQL" databases since they incorporate lots of additional features in addition to the features of conventional databases. The relational databases strongly follow the ACID (Atomicity, Consistency, Isolation, and Durability) properties while the NoSQL databases follow BASE (Basically Available, Soft State, Eventual consistency) principles. This survey paper is an analytical study on BASE features of some of NoSQL databases.

1. Introduction

NoSQL databases are commercial as well as open source database management software supporting massive data storage across distributed servers. Some of the NoSQL support SQL or SQL-like query language, hence to avoid the misunderstanding that SQL cannot be used, NoSQL is redefined as "Not Only SQL".

The following three are the major contributors of the NoSQL movement [1]:

I. BigTable (Google) started in 2004 by Google.
II. Dynamo: Amazon's Highly Available Key-value Store.
III. CAP theorem.

Lots of commercial and open source NoSQL are developed and floated in the market by various vendors. Currently there are more than 150 NoSQL databases.

The common motivations of NoSQL design are:

- Easier deployment.
- Large scale data.
- Meeting the scalability and failover.
- Can be used as a Caching layer for storing the transaction data.

Key features of NoSQL are:

- "Horizontal Scaling/Scaling Out" for "simple operations" by adding more machines to a pool of resources, whereas, "Vertical Scaling/Scaling Up" means adding more CPU, RAM, etc. to existing machines.

Scaling Up by adding new expensive big servers is difficult because of

- Hardware limitations.
- Involves lots of complex Join Operation.
- Requires higher level of skills.
- Not reliable in some cases.
- Replicate/distribute data over many servers.

The RDBMS can "Scale Up", but not "Scale Out", hence suitable for strong Consistency and Availability. In most of NoSQL data is partitioned and replicated across multiple nodes. Inherently, most of them use either Google's MapReduce or Hadoop Distributed File System or Hadoop MapReduce for data aggregation.

E-mail address: ganeshdeka2000@gmail.com.

http://dx.doi.org/10.1016/j.future.2015.05.003
0167-739X/

Variety of NoSQL systems having different features exist. Since it is not possible to do an analysis of BASE features of all the NoSQLs, some of the popular NoSQL are discussed. The systems discussed in this study are more representative rather than comprehensive. This paper is organized into **7** sections. Section 2 is about the characteristics of NoSQL databases. Section 3 is about categories of NoSQL databases. While Section 4 is BASE analysis of NoSQL. Section 5 is a Discussion about the BASE features of various NoSQL. Section 6 is about the benchmarking of Cloud services with a focus on NoSQL. Section 7 is Observations and recommendations for picking up NoSQL for a particular application. Section 8 concludes the paper.

2. NoSQL characteristics

The following characteristics are mostly found in all NoSQLs. (a) *Schema-less*: Relational databases follow the strict schema which every row or tuple must follow. Each attribute within these tuples is atomic having prescribed domain which it must follow and sometimes has constraints placed on such as NOT NULL or UNIQUE. The systems that need to deal with data requiring frequent changes are the best fit for a schema-less data store such as NoSQL [2].

(b) *Shared Nothing architecture*: Instead of using a common storage pool such as SAN each server uses only own local storage, allowing storage to be accessed at local disk speeds instead of network speeds which also permits capacity to be increased by adding more nodes. Costs are also reduced since the commodity hardware can be used with "Shared Nothing" architecture.

(c) *Elasticity*: Elasticity of databases means dynamic expandedability. When a new node is added to the mesh, some subset of data is to be replicated in new nodes.

Elastic databases emphasizing BASE principles necessitate to give-up traditional paradigm of ACID transactions. However, various implementations can prioritize certain requirements of availability over other requirements such as consistency. The following NoSQLs also support elasticity in principle:

- Cassandra.
- Dynomite.
- Amazon's Dynamo.
- Voldemort.
- MapReduce.

(d) *Sharding*: Instead of considering the record storage as a heap of memory locations, records can be partitioned into shards, which are small enough to be managed by a single server. Shards are replicated as per requirement and split when it gets too big. Applications assist in data sharding by assigning each record a partition ID automatically.

(e) *Asynchronous Replication*: Compared to synchronous replication technique such "Mirroring" and/or Striping, NoSQLs employ asynchronous replication, allowing writes to complete faster and smoother as they are independent of network traffic. However, the limitation of asynchronous replication is that data is not immediately replicated and could be lost in certain windows. Further locking is generally not available for protecting copies of a specific unit of data.

(f) *BASE instead of ACID*: The acronym BASE was purposely chosen to contrast to ACID paradigm. NoSQL emphasizes on "Availability" and "Performance". Building a database providing ACID properties is difficult, therefore Consistency and Isolation are often forfeited, resulting in most of the BASE approach application. One of the basic concepts behind BASE is that data consistency is to be taken care by the developer's problem and should not be handled by the database.

In a "Shared something" environment, ACID is wanted for forcing Consistency at the end of the transaction, whereas in a "Shared Nothing" environment, BASE is implemented. The features of ACID and BASE are given in Table 1.

The technical mechanism for "Commit" and "Transactions processing", etc. does not differ much in ACID and BASE, but the usage differs.

ACID comes from a paradigm of one database with "Many users" and that transaction on datasets are made only one at the time having the ability to change a value. For ACID deadlocks are a challenge. Some NoSQL solutions such as CouchDB are fully ACID and some can be configured to behave as ACID [3].

But, BASE comes from the paradigm where data is distributed and synchronization of data is not feasible. In BASE the exact values are not utterly necessary. A challenge for BASE is how to update distributed data while the data takes many unforeseen routes.

Table 2 summarizes the discussion.

Applications specifically financial transactions cannot cope with underlying features provided by BASE. For example, in an online banking application a user who has transferred a certain amount of money to some account must get the confirmation about the successful transaction instantly. However, if the original account's balance is untouched due to "eventual Consistence" the user might end up issuing multiple transfers and eventually transferring cash multiple times. Hence online banking applications in general cannot cope with BASE properties, whereas for applications such as Facebook, reading an old value in an instance of time is acceptable, hence can possibly best fit for BASE.

In Service Oriented Architecture (SOA), because of the nature of the comprised distributed service capabilities and the availability of standards, implementing consistency is difficult to achieve across service boundaries. The BASE approach, especially outside service boundaries is more easily implementable. If we look at task services and certainly orchestrated task services, the BASE is the default approach. Offering BASE is easier, if it is known what is the accepted margin-of-error in the business or project principal, identified during the requirements clarification phase of every service delivery effort. Without proper requirements and business process analysis, it is virtually impossible to determine whether the ACID or BASE is suitable [4].

3. NoSQL categories

NoSQL databases are broadly classified into following 2 categories:

(a) **Aggregate Oriented**.
(b) **Non-aggregate Oriented**.

All the A**ggregate Oriented** NoSQLs collect information from various nodes in the network cluster. Map-Reduce can be used to rearrange the data into different aggregate forms. Aggregate oriented database is useful if the same aggregate is used frequently.

- **Aggregate Oriented**: The NoSQL under this category are:
  - **Key/Value** Databases such as Amazon S3 (Dynamo), Voldemort, Scalaris, etc.
  - **Column family, such** as Cassandra used in Banking, financial industry. Writes are faster than read, so one natural niche is data analysis. HBase (used by Search engines, any application where scanning huge, two-dimensional join-less tables are a requirement).
  - **Document-based** such as MongoDB.

Non-aggregate Oriented NoSQL such as Graph database are a superset of Aggregate Oriented NoSQL such as Key/Value, Column family and Document-based. The Non-aggregate oriented NoSQL uses relations heavily. Non-aggregate databases are more ACID compliant. In case of applications involving lots of complex Queries Non-aggregate database such as Neo4J is the best choice.

Table 1
Features of ACID and BASE.

ACID	Atomicity	All or nothing of the N actions. Commit or Rollback
	Consistency	Transactions never observe or cause inconsistent data
	Isolation	Transactions are not aware of concurrent transactions
	Durability	Acknowledged transactions persist in all events
BASE	Optimistic behavior	Accept temporary database inconsistencies
	Basically Available	Availability by replication
	Soft state	It is the user's application's task to guarantee Consistency
	Eventually consistent	Weakly Consistent, the database will be consistent in the long run; 'stale' data is OK.

Table 2
ACID versus BASE.

ACID [C + A]	BASE [A + P]
Strong consistency	Accomplishes Consistency, Atomicity and Partition tolerance "eventually"
Isolation	Availability first
Focus on "commit"	Best effort
Nested transactions	Approximate answers
Pessimistic: Force consistency at the end of transaction	Optimistic: Accepts temporary database inconsistencies, Eventually Consistent
Difficult evolution	Simpler, Faster, Easier evolution
Suitable for Financial Portals	Suitable for non-financial web-based applications
Safe	Fast
Shared Something (Disk, Memory)	Shared Nothing
Scale UP (limited)	Scale Out (Unlimited)
Simple Code, robust database	Complex code, simple database
Single Machine	A cluster
CA	AP/CA/CP, i.e. any 2 out of 3.
Scale Vertically	Scale Horizontally
SQL	Custom APIs
Full Indexes	Indexing is mostly on Keys

The basic difference between the Aggregate oriented database and Non-aggregate databases is that **Aggregate Oriented** database ***splits relations***.

The common feature is that both the **Aggregate Oriented** and **Non-aggregate Oriented** NoSQL are ***Schema-Less***. Now, the above mentioned NoSQL and their underlying data storage format are briefly discussed.

(a) *Key/value databases*

The key-value stores are commonly known as dictionaries or hash. In NoSQL key-value stores have been adopted because they promote easy scalability and flawless growth at rapid speed. The basic concept is that a globally distributed hash table has keys that lead to the different database servers scattered all over the world. Each data item is converted into a key using some unique formula which is stored in the lookup table or directory. When the data are needed, the key is converted into the location of the data and accordingly data are retrieved.

The majority of the NoSQL databases described in this study are key/value stores at their core habitually providing additional functionality for access by secondary values. Voldemort [5] is a key-value storage system used in LinkedIn. Examples of key/value databases without additional indexing facilities are:

- Berkeley DB.
- MemcacheDB.
- Redis.
- Tokyo Cabinet/Tyrant.
- Riak.

The Pros and Cons of Key/Value Databases are:

Pros:

- Very fast.
- Very scalable.
- Simple model.
- Able to distribute horizontally.

Cons:

- Many data structures (objects), hence cannot be easily modeled as key–value pairs.

(b) *BigTable*

"BigTable" databases are known as Record-oriented or Tabular databases consisting of multiple Tables, each containing a set of addressable rows. Each row consists of a set of values that are considered columns. In addition to Google's BigTable database other examples are:

- Azure Tables (Microsoft).
- Cassandra (Apache).
- HBase (Apache Hadoop project).
- Hypertable.
- SimpleDB (Amazon).
- Voldemort (LinkedIn, now open source).

(c) *Columnar databases*

Columnar databases are a hybrid between NoSQL and relational databases. They provide some row-and-column structure, but do not have the strict rules of relational databases.

Column-oriented databases store and process data by column instead of row. Having its origin in analytics and business intelligence, column-stores can be used to build high-performance applications. Column oriented [6] stores are seen less puristic, subsuming data stores that integrate column and row-orientation. This is faster because most of the tables contain lots of columns which are rarely used simultaneously by queries. Columnar databases only perform I/O on the blocks corresponding to columns that are actually being read/updated. In addition to the smaller I/O overhead, memory is more efficiently utilized. The column-oriented data store very effective in ***block***ing data based on each column's data type such as ***Date, Text,***etc.

(d) *Document databases*

In document oriented databases, data is stored as documents. A set of documents is called collections. Collections may contain

Table 3
NoSQL data model features.

Data model	Performance	Scalability	Flexibility	Complexity	Functionality
Key-Value Stores	High	High	High	Low	Variable (None)
Column Store	High	High	Moderate	Low	Minimal
Document Store	High	Variable (high)	High	Low	Variable (Low)
Graph Database	Variable	Variable	High	High	Defined by Graph theory

any number of documents of any type. This is the key feature of document oriented databases to become schema-less as well as giving them the advantage of porting and storing data of different types easily.

Document databases focus on storage and access optimization for documents as opposed to rows or records. The document database entities are just records with multiple fields. All of the current document databases support documents in JSON (JavaScript Object Notation) format.

Some document databases are BigTable databases, others provide MapReduce processing, while some of them are columnar databases. The inference is that the storage and access implementation is independent of the database's orientation towards document-oriented databases. Some document databases provide added capabilities such as SQL or SQL-like query processing capabilities. For instance, Terrastore document store provides advanced scalability and elasticity features without sacrificing Consistency.

(e) *Graph Databases*

Graph Databases (GD) store information in multi-attribute tuples that reflect relationships in a different way. At the top of GD there may be a key/value store, columnar database, "BigTable" database or combination of these and other architectures.

A graph database might be used to store the "friend" relationships of a social network, with a record merely consisting of two friends who share a relationship. GDs are suitable for handling highly interconnected data and therefore are very efficient in traversing relationships between different entities. In Facebook everything is a Graph.

The features of GD are:

- ACID compliant.
- Inspired by Euler's Graph theory.
- Date model: Nodes, relations, K-V on both.

Graph Databases are a suitable for the following types of applications [3]:

- Location-based Services.
- Recommendation Engines.
- Complex network-based applications such as Social, Information, Technological, and Biological networks.

Table 3 summarizes the salient features of various NoSQL models discussed above.

4. Base analysis

eBay proposed BASE. eBay services run in a single data center on a reliable network, deliberately engineered to use potentially stale or incorrect data, rejecting synchronization in support of faster response with the risk of inconsistencies [7].

Researchers at Amazon have also adopted BASE. BASE is a database design philosophy that prizes Availability over Consistency of operations.

BASE is:

Basically available: The database system always seems to work. **B**asically available indicates that the system *does* guarantee Availability, in terms of the CAP theorem. NoSQL databases spread data across many storage systems with a high degree of replication for high degree of Availability.

Soft state: With Soft start, the copies of a data item may be inconsistent and the state of the system may change over time, even without input because of the "Eventual Consistency".

"The soft state mode should be utilized for applications built on legacy functionality in the backend, particularly when the functionality includes initial loading/caching of large sums of information for a single request. By using soft state, the resources/functionality which has been loaded during the initial load can be reused for the subsequent requests of the service" [8].

Eventually consistent: The updates propagate, when there are no updates over a certain period of time defied for a particular application. Hence, the systems where BASE is the key requirement for reliability, the potential of the data changes essentially slows down.

A lot of databases on websites prefer Speed, Performance and Scalability instead of pure Consistency and integrity of data. Availability of BASE system is ensured through accepting partial partitions. Protocols such as Gossip can be used to replicate data optimistically and later heal any problems that arise [9]. "Updates are applied in the first-tier, but then passed to inner-tier services which might apply them in different orders" [9].

The elasticity of storage and server resources is at the crux of BASE paradigm. BASE databases use strategies to have Consistency, Atomicity and Partition tolerance "eventually". BASE does not flout CAP theorem, but works around it. If users are partitioned across five database servers, BASE design encourages crafting operations in such a way that a user database failure impacts only the 20% of the users on that particular host.

The following subsections discuss about the BASE properties of some of the mostly used NoSQL.

4.1. Basically available

Means data are available most of the time. A failure could cause some data not to be available, but only catastrophic failures cause everything down.

(1) Dynamo is highly focused on availability achieved by replication. This feature has deeply influenced the customer experience, which translates into revenue from commercial transactions. It aims to achieve an "always writable" data store because rejecting customer updates could result in a poor customer experience and the conflicts are resolved during the read operation. Dynamo powers Amazon Web Services such as Amazon S3 for cloud computing, including Dropbox and Ubuntu One [10,11].

(2) GFS (Google File System) maintains high availability through faster recovery, chunk and master replication. Both the master and chunk server can restore their states and restart in seconds. Every chunk of data is replicated on multiple chunk servers. Operation log and checkpoints are owned by the master, and are replicated on multiple machines. GFS supported BigTable provides the data storage backbone for several services such as Web indexing, Google Earth, Google Finance, Google Reader, YouTube and Gmail [12,13].

(3) BigTable uses Chubby (lock service for loosely-coupled distributed systems) to provide availability of the system using replicas that are served by tablet servers in Bigtable. As Bigtable relies on Chubby for a number of tasks, when Chubby fails, Bigtable fails, too [12] reports that the average percent of Chubby that caused the unavailability of Bigtable is 0.0047%. Also, the strong,

consistent policy of Bigtable which forces a log page to disk incurs a fair bit of overhead.

(4) Cassandra uses a mechanism that creates N replicas of the same object. Each key has a coordinator node that is the in-charge for replicating keys on N-1 nodes. Cassandra provides various options for data replication including:

- Rack-unaware.
- Rack-aware.
- Datacenter-aware.

If replication is rack-unaware, the coordinator simply chooses N-1 nodes from the ring. Otherwise, the system elects a leader amongst the nodes and every joining node will be told by the leader what ranges they have for the replicas.

(5) Hadoop DFS: The NameNode is in-charge of ensuring each block always has the intended number of replicas. Every time a report from a DataNode arrives, the NameNode will determine how many replicas each block has. If a block is under-replicated, it will get inserted in the replication priority queue. An over-replicated block will make the NameNode to remove one replica. If a replica resides on one rack and other replica is scheduled to be created on the same rack, the system will find a new rack to create the replica.

(6) In Haystack, each photo is replicated throughout all physical volumes of a store which means the same photo will be available in as many servers as the store contains. However, the same picture may be in the Cache and can be retrieved quickly.

4.2. Soft state

In soft state database provides a relaxed view of data in terms of consistency. Information on soft state will expire if it is not refreshed. The value stored in soft state may not be up-to-date, but handy for approximations. Soft state data are in changing state over time without user intervention and/or input due to eventual consistency.

The soft state layer is responsible for [14].

- Client interface.
- Data partitioning.
- Caching.
- Concurrency control.
- High level processing.

Replicating soft state provides applications with two critical capabilities:

- Rapid failover to other instances during crashes.
- Fine-grained load-balancing across instances to prevent overload [15].

Soft state is useful for efficiency because of the eventual consistency model such as short-lived user sessions, stored aggregates and transformations on large datasets and general purpose write-through caches for files and database records.

Whilst soft state is lost or made unavailable due to service instance crashes and overloads, reconstructing it through user interaction or third-tier re-access can be expensive in terms of time and resources [16]. User can afford database to be consistent over time by synchronizing information between different database nodes. They can Cache data (soft state) and use it later to increase the database response time. They may be having a number of database nodes with distributed data to be highly available (partition tolerance).

Many options exist for adding high availability of programs that manipulate soft state and these can be broadly classified into three categories [17]:

- Clustered application servers.
- Messaging toolkit.
- Collocated in-memory databases.

Nodes in the soft state layer are organized in a logical ring overlay as nodes of the fully decentralized type. For example, a user request can be transparently redirected during a crash or overload to a different service instance that has up-to-date session context, without requiring log-in again. Loss of connectivity to nodes prompts other nodes to take over. BASE databases can recover quickly and consistently since they usually reside on standard commodity hardware, making them affordable for most businesses.

Services using DDS (Data Distribution Service) may keep soft state, but they rely on the hash table to manage all persistent states. DDS library contains only soft state, including metadata about the cluster's current configuration and partitioning of data in the distributed hash tables across the BRICKS (Building Resources for Integrated Cultural Knowledge Services, an open-source software framework for the management of distributed digital assets).

4.3. Eventually consistent

As data is replicated when new storage nodes are added, it is eventually copied to all applicable nodes. But there is no requirement for all nodes to have identical copies of any given data all the time. BASE ensures at least 80% consistency at any given instant over the flow of the operations.

(1) Dynamo operates with high consistency at the cost of weaker availability. It is designed to be "Eventually Consistent" where consistency is maintained through "Object Versioning". As the system aims to provide an "always writable" state, the conflicts are resolved in the read operation using the versioning and application-specific resolution protocols. When dealing with the uncertainty of the correctness of an answer, the data are made unavailable until it is absolutely certain that it is correct. Dynamo points to self-repair mechanisms in the key-value store to implement eventual consistency in practice [18].

(2) GFS: GFS also has an eventual consistency model as Dynamo. It applies the operations log to recover the file and uses chunk version numbers to detect any stale replicas. Chunk server uses checksum to detect any additional data corruption due to server failure. In case of uncertainty, the data are made unavailable. Any successful operation of **atomic_record_append** guarantees consistency.

(3) Bigtable has its own features to support consistency. Each read or write is serializable, making it easy to maintain consistency in case of concurrent updates to the same row. Each row is copy-on-write to maintain row-level consistency.

Different cells in a table can contain multiple versions of the same data, indexed by timestamps. SSTables contain relatively old updates, while a buffer called memtable contains the recent updates. Updates are recorded in a commit log to recover the memtable. There is no synchronization of accesses required when reading from SSTables because SSTables are immutable.

(4) Cassandra uses a weak consistency model to maintain different replicas of an object. When a write request is received in a node, the system routes the requests to the replicas and waits for a quorum of replicas to acknowledge the completion of the **write**. Read requests are based on the consistency guarantees required by the client. The system can either route the request to the closest replica or send the request to all replicas and wait until a quorum of replicas respond.

Cassandra with TTL set up to make it completely soft state, but this is an unusual mode of usage i.e. essentially a transient cache. Soft state might also apply to the chat protocol within Cassandra. New nodes can determine the state of the cluster from the chat messages it receives and this cluster state must be constantly refreshed to detect unresponsive nodes.

(5) In HDFS (Hadoop DFS), applications create a new file by writing the data to it. Once the file is closed, the bytes cannot

be modified or removed except the new data can be added to the file by reopening the file again. HDFS implements a single-writer, multiple-reader model. Every time a node opens a file, it is granted a lease for the file, no other client can write to that file. A hit to the NameNode permits the lease to be extended. Once the lease expires and the file is closed, the changes are available to the readers.

In Hadoop DFS Queues are allocated a fraction of the capacity of the grid in the sense that a certain capacity of resources will be at their disposal. All applications submitted in the queue will have access to the capacity allocated to the queue. Administrators can configure soft limits and optional hard limits on the capacity allocated to each queue.

(6) Haystack: Even when a needle (a photo) in a Haystack is stored in all physical volumes of a logical volume, all updates go through the same logical volume and are applied to all the replicas of a photo. A store machine receives requests to create, modify and delete a photo. These operations are handled by the same store machine.

5. Discussion

This section provides a view on what type of NoSQL is suitable for a particular scenario.

(1) Web integration: The attractive features of a database system for present scenario are ease of integration with web and mobile applications. APIs are to be developed in line with the database system, so that it is easy to create web pages or mobile applications that interact with it. CouchDB is a popular database for integrating the web and mobile applications [19].

(2) Usage of SSD: [11] using a solid-state disk (SSD) for logging as an SSD can provide durable writes with very low latencies. Hadoop uses SSD for the response time critical data of relatively small size [20].

(3) Single master: Google embraced the concept of a single master in their distributed system as a valuable tool to simplify the design and make it more straightforward. Currently GFS uses one master per cell [21], with multiple cells per data center. Even though Google has worked around the issue this case underscores the importance of scalability and the limitation of assigning different levels of responsibility in any distributed system that is expected to scale. The biggest weakness in this system that, the Master can be the single point of failure. However, this can be overcome by replicating the master state. The "Shadow Masters" can provide read-only access in the absence of primary master.

(4) Overcompensated systems: CAP theorem explores trade-offs between Consistency, Availability, and Partition tolerance. CAP theorem only discusses about the limitations in the face of certain types of failures and does not constrain any system capabilities during normal operation. CAP allows the system to guarantee all of the ACID alongside high availability when there are no partitions [22].

(5) Consistency versus Latency: Studies indicate that latency is a critical factor in online interactions. Increase as small as 100 ms can dramatically reduce the probability that customers will continue to interact or return [23] to a web portal. In case of e-Commerce application this is directly related to the profits. Latency can be achieved without compensating the CAP theorem and hence it is expected that database management systems are increasingly going to pay more attention to decreasing the latency.

As a matter of fact, latency tends to come at the cost of consistency. Considering several replicas in the system, a client could immediately read the **first replica** which it can access without checking consistency. The second option is that the client could opt

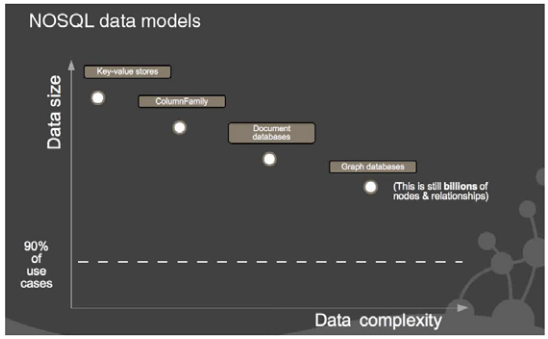

Fig. 1. NoSQL databases data size and data complexity analysis graph.

for the "Quorum reads", which requires the system to check for inconsistency across multiple replicas before reading. Surely the **second option** causes some amount of latency. Difference in latency between these two options can be a factor of **four** or **more** [22].

(7) Scalability and Complexity Analysis.

Fig. 1 shows the NoSQL Scalability Complexity relationship.

As shown in the figure Key-value NoSQL are suitable for data intensive applications, while the Graph databases (Non Aggregate) NoSQL are best fit for applications dealing with complex data. It is also evident from the figure that, the Document (Aggregate oriented) NoSQL is the most suitable NoSQL databases having a good balance of "Data Complexity" and "Data Size". Since 90% of use cases are much below the "Data Size" supported by the entire NoSQL database, still most of the application is perfectly running with conventional RDBMS.

(8) NRW Analysis.

NRW (Node, Read, Write) analysis is used to analyze the characteristics of distributed database how they will trade off Consistency, Read and Write performance [24].

Here,

N is the number of Nodes keeping copies of distributed record;

W, number of nodes that must successfully acknowledge for a Write to be successfully committed.

R, the number of nodes that must send back the data to be accepted as read by the system.

The majority of NoSQL databases uses $\mathbf{N} > \mathbf{W} > \mathbf{1}$ i.e. more than one write must complete, but not all nodes need to be updated immediately.

Tables 4 and 5 categorize the NoSQL according to NRW [25,24].

The "Read Repair" algorithm is often implemented to improve consistency when $R = 1$.

The following are some more muscle with various Consistency levels [24].

6. Benchmarking of NoSQL

The Yahoo! Cloud Serving Benchmark (YCSB) platform was developed to evaluate two types of benchmarking tiers:

- Performance and
- Scalability.

YCSB is a popular extensible framework designed to compare different Table stores under identical synthetic workloads. YCSB can be used for measuring the basic performance of [26]:

1. HBase.
2. Voldemort.
3. Cassandra and
4. MongoDB.

Table 4
Node, read, write is of NoSQL.

Parameter	NoSQL performance
$W < N$	High write availability
$R < N$	High read availability
$W = N, R = 1$	Strong consistency, *Read optimized*
$W = 1, R = N$	Strong consistency, *Write optimized*
$W + R > N$	Strong consistency. A read will see at least one copy of the most recent update. Write are fully overlapped
$W + R \leq N$	Eventual (Weak) Consistency

Table 5
NoSQL consistency analysis.

NoSQL	Consistency level
BigTable, used by Google App Engine, and HBase, which runs over Hadoop	Strongly *consistent* within a data center. Eventual consistency between data centers. Updates are propagated to all replicas *asynchronously*.
Google Spanner, a Globally-distributed and synchronously-replicated database	Updates are propagated to all replicas synchronously. Supports strong consistency even in the presence of wide-area replication
Amazon's Dynamo, Cassandra and Riak	Sacrifice consistency in favor of availability and partition tolerance. Offers eventual consistency
Oracle NoSQL Apache Cassandra	Consistency policy based on performance requirements. Tunable consistency

Table 6
Yahoo! cloud serving benchmarking.

Sl No	YCSB	Parameters for Benchmark
1	YCSB Workload **A**: Update-Heavy: Session Store	**50**% **read** and **50**% **update**. The standard size of the dataset is 200,000 Key–Value pairs.
2	YCSB Workload **B**: Read-Mostly: Photo Tags	Read-heavy benchmark consists of **95**% **read** and **5**% **update** operations. Standard size of set is 200,000 Key–Value pairs.
3	YCSB Workload **C**: Read-Only: User Profile Cache	100% **read** operations.
4	YCSB Workload **D**: Read-Latest: User Status Updates	**95**% **read** (recently Inserted record) operations and **5**% **insert** operations. Read load is shewed towards the end of the range.
5	YCSB Workload **E**: Short Ranges: Threaded Conversations	95% **Scan** and 5% **Insert**. Patterned after an online forum with threaded conversations, workload is based around **Scan operation** that retrieves posts in a given thread, clustered by the **thread ID**. **95**% of operations in this workload are **scan** operations that perform Search/Queries on ranges of records
6	Workload **F**	50% **Read** and 50% **Read–Modify–Write**.
7	**YCSB**++	Evaluate non-transactional access to distributed key-value stores. Adds functionality to enable bulk loading of data into HBase and Accumulo in the form of an extended API. Also allows operations like B-Tree splits to be performed on the database indexes to simulate application usage [30].
8	**YCSB** + **T**	**Wrap**s multiple database operations within **Transaction**s.

YCSB has been used in more than 70 published benchmark comparisons and represents a standardized way to compare the performance between systems [27]. However, YCSB was not designed to run stress tests (high availability tests) on the systems.

Table 6 shows the 6 major workload types in YCSB [28,29].

Web-scale distributed NoSQL systems like Spanner and Percolator offer transaction features to cater to new web-scale applications, leading to development of YCSB + T. **YCSB + T** is completely backward compatible with YCSB enabling existing benchmark code to run without any modification.

SandStorm is an enterprise performance testing tool for web, mobile and Bigdata applications. It provides a framework for benchmarking NoSQL, Hadoop, and Message queue systems. The current SandStorm release provides support for the following NoSQL technologies [31]:

- MongoDB.
- Cassandra.
- HBase.
- Oracle NoSQL.

Recently, Google has launched **PerfKit**, an open-source cloud-benchmarking tool. The PerfKit tools currently support Google's own Compute Engine, Amazon's AWS and Microsoft's Azure clouds [32].

The benchmarking test conducted by Engineers at the University of Toronto in 2012 found that, **Apache Cassandra** is the winner throughout their experiments. **End Point Corporation**, a database and open source consulting company, benchmarked the top 3 NoSQL databases as follows, all using a variety workloads on Amazon Web Services(AWS) [33]:

1. Apache Cassandra.
2. Apache HBase.
3. MongoDB.

A study titled "NoSQL Benchmark Study Release" by **Elizabeth Gallagher** sponsored by **Couchbase** claims that, Couchbase is the clear leader as far as complete throughput, when 90% of data is **In-memory**, followed by MongoDB and Cassandra [34]. All databases served over 60,000 Transactions Per Second (TSP) with **Couchbase** far exceeding that with over 900,000 TPS. "For balanced workloads (50% read, 50% write), **MongoDB** showed the best scaling behavior.

Couchbase and **Cassandra** both show scaling at about half of linear capacity throughout. **Cassandra** actually scales slightly better for the balanced workload''. According to this study ''for heavy-read workloads (95% read, 5% write), both **Couchbase** and **MongoDB** scale close to linearly'' [34].

Column Family databases **Cassandra** and HBase show excellent writing abilities, but its reading performance is poor, since these two products were optimized for writing resulting to lots of concurrent I/O when reading. Since Cassandra uses huge amounts of memory, it has to perform lots of disk I/O in **read** heavy workloads, leading to a highly decreased performance. Both Cassandra and HBase have a better performance during execution of **Update**s.

MongoDB has **similar** structures to RDBMS and shows great flexibility in data modeling, especially for medium and small-sized businesses [35]. MongoDB is the best fit for read intensive applications.

7. Observation

The downsides of NoSQL

- There is no universal query language like SQL.
- Each NoSQL product does things differently.
- SQL is very powerful and expressive.
- Relational databases are very mature, 40+ years (1970) while NoSQL are 6+ years old.
- Relational databases are part of a vast ''Ecosystem'' with lots of applications and Tool availability.

NoSQL communities have picked up the trend to abandon relational properties in favor of high-scalability by only supporting Key-Value type accesses in their data stores. However, abandoning SQL and its feature has nothing to do with Scalability. Another significant development in advanced database technology is ''Polyglot Persistence'', i.e. using multiple data storage technologies based upon the way data are being used by individual applications. The simple logic behind ''Polyglot Persistence'' is *why store binary images in a relational database, when there are better storage systems for the same?*

The following are some of the issues for selection of a particular NoSQL database:

- Fit workload requirements to the best suited cloud database system considering the read-optimized against write-optimized substitution.
- Latency versus Durability is another important axis. If developers know that they can lose a small fraction of writers such as web poll votes, etc. they can acknowledge success, writes without waiting for them to be synced to disk. An application requiring large number of small writes may use ''Redis''.
- Auto-completion, Caching may use Redis, Memcached.
- Data mining, Trending-MongoDB, Hadoop and BigTable.
- Content based web portals-MongoDB, Cassandra and Sharded ACID databases.
- Financial Portals-ACID database.

8. Conclusion

In this paper an analytical study of BASE properties of NoSQL database with a focus on large-scale NoSQL such as Dynamo, Google File System (GFS), Bigtable and Hadoop are done. Different techniques that are used to achieve Consistency and Availability were analyzed. Some recommendations based on observations are made for selection of a NoSQL for particular purposes.

References

[1] Ganesh Chandra Deka, A survey of cloud database systems, IT Prof. 16 (2) (2014) 50–57. http://dx.doi.org/10.1109/MITP.2013.1.

[2] Survey of Large-Scale Database Systems by So Youn Lee, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

[3] Grolinger, et al., Data management in cloud environments: NoSQL and NewSQL data stores, J. Cloud Comput. Adv. Syst. Appl. 2 (2013) 22.

[4] R. Stoffers, Acid vs. base [Web log message], 2010 July 2. Retrieved from http://soa-ind.blogspot.in/2010/07/acid-vs-base.html.

[5] G Inc. How entities and indexes are stored-google app engine, 2012, April. [Online]. Available: https://developers.google.com/appengine/articles/storagebreakdown?hl=pl#anc-background.

[6] MongoDB Forbes Case Study. http://www.10gen.com/customers/forbes.

[7] Chad DeLoatch, Scott Blindt, NoSQL databases: Scalable cloud and enterprise solutions, University of Illinois at Urbana Champaign Thursday, August 2, 2012.

[8] SAP SE. Soft state support for OData services, 2015, February 8. Retrieved from http://help.sap.com/saphelp_gateway20sp09/helpdata/en/f6/5f8e5318e83d27e10000000a44538d/content.htm.

[9] Kenneth P. Birman, Daniel A. Freedman, Qi Huang, Patrick Dowell, Overcoming CAP with consistent soft-state replication, Computer 45 (2) (2012) 50–58. http://dx.doi.org/10.1109/MC.2011.387.

[10] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. (TOCS) 26 (2) (2008) 4.

[11] Dropbox. Dropbox-where does Dropbox store everyone's data? 2012, April [Online]. Available: https://www.dropbox.com/help/7.

[12] R. Rees, NoSQL, No problem: An intro to NoSQL Databases, 2010, September 27. Retrieved from http://www.thoughtworks.com/insights/blog/nosql-no-problem-intro-nosql-databases.

[13] Amazon. Amazon dynamodb, 2012, April [Online]. Available: http://aws.amazon.com/dynamodb/.

[14] BEA Systems Inc. Clustering the BEA WebLogic Application Server, 2003. http://e-docs.bea.com/wls/docs81/cluster/overview.html.

[15] Tudor Marian, et al. Tempest: Soft state replication in the service tier*, Department of Computer Science, Cornell University, Ithaca, NY 14853.

[16] W. Vogels, Eventually consistent, ACM Queue (2008) 14–19.

[17] X. Zhang, et al. Customizable service state durability for service oriented architectures, in: Sixth European Dependable Computing Conference, 2006, pp. 119–128.

[18] D. Pritchett, BASE: An acid alternative, ACM Queue (2008) 48–55.

[19] LinkedIn. Project Voldemort, 2012, April [Online]. Available: http://project-voldemort.com.

[20] Apache. Apache couchdb, 2012, April [Online]. Available: http://couchdb.apache.org/.

[21] B. Li, E. Mazur, Y. Diao, A. McGregor, P. Shenoy, A platform for scalable one-pass analytics using mapreduce, in: SIGMOD Conference, 2011, pp. 985–996.

[22] K. McKusick, S. Quinlan, GFS: evolution on fast-forward, Commun. ACM 53 (3) (2010) 42–49.

[23] D. Abadi, Consistency tradeoffs in modern distributed database system design: Cap is only part of the story, Computer 45 (2) (2012) 37–42.

[24] vibneiro. Ivan voroshilin's blog:algorithmic contests, distributed systems and software architecture [Web log message], 2012 December 13. Retrieved from http://ivoroshilin.com/2012/12/13/brewers-cap-theorem-explained-base-versus-acid/.

[25] G. Harrison, Consistency models in non relational databases [Web log message], 2010 June 13. Retrieved from http://guyharrison.squarespace.com/blog/2010/6/13/consistency-models-in-non-relational-databases.html.

[26] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: Proc. of the 1st ACM Symposium on Cloud Computing, SOCC'2010, Indianapolis, IN, June 2010.

[27] ThotSpots (n.d.). Mongodb performance characteristics on joyent and Amazon EC2. Retrieved from https://www.joyent.com/content/02-public-cloud/02-benchmarks/02-mongodb/mongodb-benchmark.pdf.

[28] HyperDex (n.d.). Performance benchmarks. Retrieved from http://hyperdex.org/performance/.

[29] Tarantool. YCSB—benchmark results, 2015. Retrieved from http://tarantool.org/benchmark.html.

[30] S. Patil, M. Polte, et al., YCSB++: benchmarking and performance debugging advanced features in scalable table stores, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, Ser. SOCC'11, ACM, New York, NY, USA, 2011, pp. 9:1–9:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038925.

[31] SandStorm (n.d.). No SQL benchmark nosql databases for performance and scale. Retrieved from http://sandstormsolution.com/nosql.html.

[32] GitHub Inc. Googlecloudplatform/perfkitbenchmarker, 2015. Retrieved from https://github.com/GoogleCloudPlatform/PerfKitBenchmarker.

[33] DataStax. Benchmarking top nosql databases, 2015. Retrieved from http://www.datastax.com/resources/whitepapers/benchmarking-top-nosql-databases.

[34] E. Gallagher, NoSQL benchmark study release [Web log message], 2014 September 11. Retrieved from http://blog.thumbtack.net/nosql-benchmark-study-release/.

[35] H.J. Lee, Nosql benchmarking, 2012. Retrieved from http://www.cubrid.org/blog/dev-platform/nosql-benchmarking/.

Deka Ganesh Chandra is the Principal, Regional Vocational Training Institute for Women, Tura under the Directorate General of Training, Ministry of Skill Development and Entrepreneurship, New Delhi, India. His previous assignment includes Assistant Director of Training in DGE&T, Ministry of Labor & Employment, Government of India, New Delhi (2006–2014), Consultant (Computer Science), National Institute of Rural Development, North Eastern Regional Center, Guwahati, Assam, an autonomous body under the Ministry of Rural Development, Govt of India [2003–2006] and Programmer, Nowgong Polytechnic, Nagaon, Assam, under the Directorate of Technical Education, Assam [1995–2003]. His research interests include ICT in rural development, e-Governance, Cloud Computing, Data Mining, NoSQL Databases and Vocational Education and Training. He is the editor-in-chief of the International Journal of Computing, Communications, and Networking, and is the Co-editor with Pethuru Raj of the **Handbook of Research on Cloud Infrastructures for Big Data Analytics** (IGI Global, 2014), Chief Editor of the book **Handbook of Research on Securing Cloud-Based Databases with Biometric Applications** (IGI Global 2015). He is the co-author of 3 text books in Science (B.Ed., Gauhati University, Classes 11 and 12). He has published around 40 papers in various IEEE International Conferences and Journals of repute (2 IEEE Journals). Deka has a Ph.D. in computer science from Ballsbridge University, Roseau Dominica. He is a member of IEEE; the Institution of Engineers, India; the Institution of Electronics and Telecommunication Engineers and he is an Associate Member of the Institution of Engineers and the Institution of Electronics and Telecommunication Engineers. Contact him at ganeshdeka2000@gmail.com.