Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Growth of New Databases & Analysis of NOSQL Datastores

**Abhinay B. Angadi**
*M.Tech(CSE),BVBCET,Hubli*
*Hubli, India*

**Akshata B. Angadi**
*Lecturer in CSE, K.L.E.I.T,Hubli.*
*Hubli, India*

**Karuna C. Gull**
*Asst. Professor in CSE,K.L.E.I.T,Hubli*
*Hubli, India*

***Abstract— Relational database management systems (RDMBSs) today are the predominant technology for storing. In the past few years, the "one size fits all"-thinking concerning datastores has been questioned by both, science and web affine companies, which has lead to the emergence of a great variety of alternative databases. There has been an enormous growth in the distributed databases area in the last few years, especially with the NOSQL movement. Keeping this as a motivation, this paper aims at giving a systematic overview of DBMS, discusses about the change from traditional file processing to RDS & ends with NOSQL. Also we have focused on the projects dealt by the NOSQL models with their description & we have said about when it is best suitable. Lastly we have listed the compared features of NoSQL & SQL. Further our paper will help researchers to develop new projects by overcoming the drawbacks of existing or work on the existing one & add up features.***

***Keywords— ACID, BASE, CAP theorem, Datastores, RDS (Relational Databases).***

## I. INTRODUCTION

Data/Information can be stored using 2 ways: Files (1960) & Databases (1970). Traditional File processing were very helpful in placing the information but problems like redundancy (number of files grows with applications, and data is duplicated), Inconsistency (data is updated in one application's files, but not in another's), Difficulty in combining data (business needs may mean users want data from different applications) made users to opt databases for storing the data. That gave a rise to DBMS concept. DBMS is a collection of data (database) and programs to access that data. The goal of DBMS is to store, retrieve, and display information. Key characteristics of DBMS are performance, store large volume of database, share data (access), provide security (authorization), remove redundancy (normalization) and provide concurrent access (different users at the same time). After early traditional file processing, HDS (hierarchical database system) was used to store & manipulate the data. Storage was in the form of parent-child relationship.

Disadvantage of HDS:

- Entire structure has to be changed if we need to add a new level.
- It doesn't support many-to-many relationships. If you require such relationship then there will be a problem of redundancy.

Then there was a rise to Network database system (NDS) whose purpose was to overcome problem of HDS. Successfully it worked but the drawback was the complexity of structure because of the use of pointer concepts. Next, the relational database system (RDS) came into light to overcome the limits of NDS. It took a dominant place in database systems for traditional database applications. Now it has spread over all the places, from small pc's to large servers. RDS approach introduced the High-Level Query Languages. Writing queries takes less time then writing a code for an application. Object oriented programming led to the development of Object oriented database. This will store data along with this it also stores the objects which consist of data & methods that operates on data. But, it couldn't replace the RDS as Object oriented database technology is built on Relational database technology.

On one hand there is the NoSQL approach, which offers higher scalability, meaning that it can run faster and support bigger loads. On the other hand, a Relational Database Management System (RDBMS) offers more consistency as well as much more powerful query capabilities and a lot of knowledge and expertise gained over the years [1].

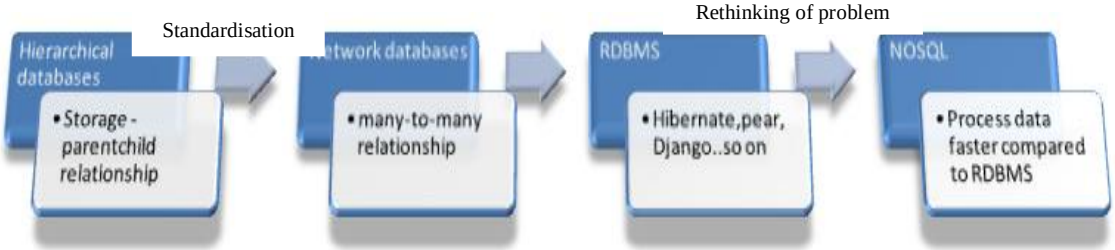


Fig 1. Rise of new databases

### A. NOSQL

It is basically a large serialized object store, retrieve objects by defined ID. In general, doesn't support complicated queries & doesn't have a structured schema. It recommends denormalization & is designed to be distributed (cloud-scale)

out of the box. Because of this, it drops the ACID requirements. Any database can answer any query, Any write query can operate against any database and will "eventually" propagate to other distributed servers. Because of the distributed model, any server can answer any query. Servers communicate amongst themselves at their own pace (behind the scenes).The server that answers your query might not have the latest data.

*B. Data model*

A data model is used in database design process; which provides a way to conceptually organize multiple data files in a database. In brief we can define data model as a collection of conceptual tools for describing data, data relationship, data semantics, and consistency constraints.Fig.2. shows the data models.

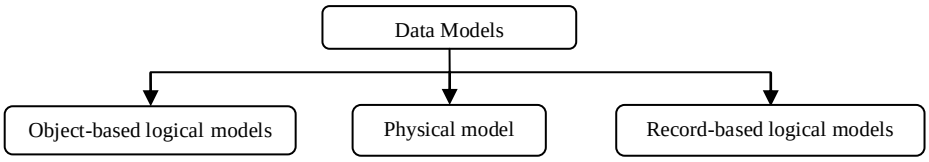


Fig 2. Shows Data Models categorized based on type of concepts used to describe Database Structure.

- Conceptual/Object-based/High Level data models: Are used to describe data at the conceptual and view level. Example of these the Entity-Relationship model and object-oriented model.
- Record-based/Implementation/Representational data models: Are used to describe data at the conceptual and view level. Examples of these are: Network model, Hierarchical model, and relational model.
- Physical data models: Are used to describe data at the physical level (bytes and words). It describes how data is stored as files in computer by representing information such as record formats, record orderings & access paths.

Even though we don't go for traditional file processing, ultimately the data in databases are stored in files itself but the structure of it is hidden. DBMS provides mainly 2 languages to alter & add the database:

- DDL (Data Definition Language): store files that contain data about data (metadata). For example storage of structure in data dictionary.-
- DML (Data Manipulation Language): enables users to access or manipulate data (retrieval, insertion, deletion). The part of DML that involves information retrieval is called a query language (QL).

*C. Query Language (QL)*

QL is the language in which a user requests information from the database. The most common query languages are Structured Query Language (SQL), Query By Example (QBE), and Quel. SQL has gain wide acceptance in commercial products.[2]

*D. SQL*

SQL database is a relational database. SQL (Structured Query Language) is a programming language that is used to manage data in relational database's. Microsoft SQL server is a best example. Microsoft SQL server is a relational database that is used to store and retrieve data by applications either on the same computers or over the network.

SQL Database Examples:

- Commercial: IBM DB2, Oracle RDMS, Microsoft SQL Server, Sybase SQL Anywhere
- Open Source (with commercial options): MySQL, Ingres

**Basic features of SQL server:**

- A relational database is a set of tables containing data fitted into predefined categories.
- Each table contains one or more data categories in columns.
- Each row contains a unique instance of data for the categories defined by the columns.
- User can access data from the database without knowing the structure of the database table.

**Limitations for SQL database:**

- Scalability: Users have to scale relational database on powerful servers that are expensive and difficult to handle. To scale relational database it has to be distributed on to multiple servers. Handling tables across different servers is a chaos.
- Complexity: In SQL server's data has to fit into tables anyhow. If your data doesn't fit into tables, then you need to design your database structure that will be complex and again difficult to handle.

In the past few years, the"one size fits all"-thinking concerning data stores has been questioned by both, Science and web companies, which has lead to the emergence of a great variety of alternative databases. The movement as well as the new datastores is commonly subsumed under the term NoSQL. It was first used in 1998 by Carlo Strozzi to name his relational database that did not expose the standard SQL interface. The term was picked up again in 2009 when a Last.fm develper, Johan Oskarsson, wanted to organize an event to discuss opensource distributed databases.

 

Strozzi was the first to coin the term NOSQL (1998) to distinguish his solution from other RDMBS solutions which utilize SQL (Strozzi's NoSQL still adheres to the relational model). He used the term NoSQL just for the reason that his database did not expose a SQL interface. Recently, the term NoSQL (meaning 'not only SQL') has come to describe a large class of databases which do not have properties of traditional relational databases and which are generally not queried with SQL (structured query language). The term revived in the recent times with big companies like Google/Amazon using their own data stores to store and process huge amounts of data as they appear in their applications and inspiring other vendors as well on these terms.[3]

The basic quality of NoSQL is that, it may not require fixed table schemas, usually avoid join operations, and typically scale horizontally. Academic researchers typically refer to these databases as structured storage, a term that includes classic relational databases as a subset. NoSQL database also trades off "ACID" (atomicity-, consistency, isolation and durability). NoSQL databases, to varying degrees, even allow for the schema of data to differ from record to record. If there doesn't exist schema or a table in NoSQL, then how do you visualize the database structure? Answer is:

*E. NoSQL Distinguishing Characteristics*

| | |
|---|---|
| 1. Large data volumes | 2. Asynchronous Inserts & Updates |
| 3. Google's "big data" | 4. Schema-less |
| 5. Scalable replication and distribution | 6. CAP Theorem |
| 7. Potentially thousands of machines and Potentially distributed around the world | 8. ACID transaction properties are not needed – BASE |
| 9. Queries need to return answers quickly | 10. Open source development |

- BASE (Basically, Available, Soft state, Eventually Consistent) Transactions: Acronym contrived to be the opposite of ACID. Characteristics:-Weak consistency, Availability first, Best effort, Approximate answers, Aggressive (optimistic).
- No schema required: Data can be inserted in a NoSQL database without first defining a rigid database schema. As a corollary, the format of the data being inserted can be changed at any time, without application disruption. This provides immense application flexibility, which ultimately delivers substantial business flexibility.
- Auto elasticity: NoSQL automatically spreads your data onto multiple servers without requiring application assistance. Servers can be added or removed from the data layer without application downtime.
- Integrated caching: In order to increase data through and increase the performance advance NoSQL techniques cache data in system memory. This is in contrast to SQL database where this has to be done using separate infrastructure.

**Advantages of NoSQL database:**

- NoSQL databases generally process data faster than relational databases.
- NoSQL databases are also often faster because their data models are simpler.
- Major NoSQL systems are flexible enough to better enable developers to use the applications in ways that meet their needs.

**CAP Theorem:**

In a keynote titled "Towards Robust Distributed Systems" at ACM's PODC1 symposium in 2000 Eric Brewer [4] came up with the so called CAP-theorem. Proponents o-f NoSQL often cite Eric Brewer's CAP theorem, which states that a system can have only two out of three of the following properties: consistency, availability, and partition-tolerance. The NoSQL systems generally give up consistency. The CAP theorem can be summarized as follows:

- Consistency: how a system is in a consistent state after the execution of an operation. A Distributed system is typically considered to be consistent if after an update operation of some writer; all readers see his updates in some shared data source.
- Availability and especially high availability meaning that a system is designed and implemented in a way that allows it to continue operation (i. e. allowing read and write operations) if nodes in a cluster crash or some hardware or software parts are down due to upgrades.

Partition Tolerance is understood as the ability of the system to continue operation in the presence of network partitions. These occur if two or more "islands" of network nodes arise which (temporarily or permanently) cannot connect to each other. Some people also understand partition tolerance as the ability of a system to cope with the dynamic addition and removal of nodes (e. g. for maintenance purposes; removed and again added nodes are considered an own network partition in this notion) [5]. So, NoSQL has emerged as a solution for today's data store needs and has been a topic of discussion and research in the recent times. There has been an enormous growth in the distributed databases area in the last few years, especially with the NOSQL movement. NoSQL databases are an efficient and powerful tool for storing and manipulating vast quantities of data. Most NoSQL databases scale well as data grows. This paper presents fundamental

 

concepts for getting you ready to know about types of NoSQL datastores with their features. Paper begins with a helpful introduction on the subject of DBMS, SQL & NoSQL, explains its characteristics and typical uses. Section 2, presents need of NOSQL. In Section 3, we have listed the NOSQL data models which will help you to choose which NoSQL data store is best for solving your specific needs. We have given a worthy comparison between the SQL & NOSQL. Lastly we concluded by a statement that "Both technologies NOSQL & SQL are best in what they do. It is up to a developer to make a better use of them depending on the situations and needs".

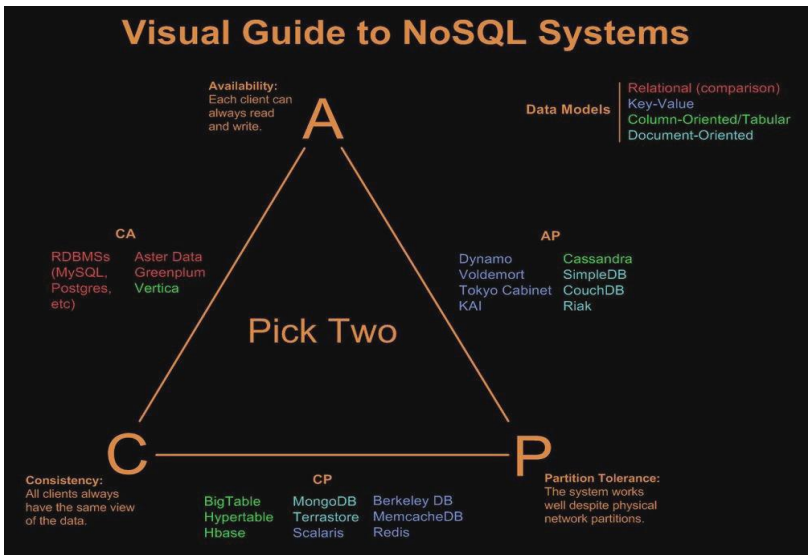


**Figure.3**. CAP theorem concept [5]

## II. WHY YOU NEED NOSQL

- The first reason to use NoSQL is because you have big data projects to tackle. A big data project is normally typified by:

a) **Data velocity** – lots of data coming in very quickly, possibly from different locations
b) **Data variety** – storage of data that is structured, semi-structured, and unstructured
c) **Data volume** – data that involves many terabytes or petabytes in size
d) **Data complexity** – data that is stored and managed in different locales, data centers, or cloud geo-zones.

- A second reason to consider a NoSQL solution is that your applications need to be continuously available.
- A third motivation to use NoSQL is that you need true location independence with a database. The term "location independence" practically means the ability to read and write to a database regardless of where that I/O operation physically occurs, and to have any write functionality propagated out from that location, so that it's available to users and machines at other sites.
- A fourth reason to use NoSQL databases is that you have applications that need modern transactional capabilities.
- One of the major reasons IT professionals move to a NoSQL database from a legacy RDBMS is the more flexible data model that's found in most NoSQL offerings. This is a fifth reason why you might use a NoSQL datastore: While the relational model works well for a number of use cases, a NoSQL data model can support many of those use cases and others that don't fit well into an RDBMS. Moreover, a NoSQL datastore is able to accept all types of data – structured, semi-structured, and unstructured – much more easily than a relational database. For applications that have a mixture of datatypes, a NoSQL database is a good option.
- A sixth reason why you would use a NoSQL database is because you need a more suitable architecture for a particular application. Some, but not all, NoSQL solutions provide modern architectures that can tackle the type of applications that require high degrees of scale, data distribution, and availability [6].

## III. NOSQL DATA MODELS

First, we should note that SQL and relational model in general were designed long time ago to interact with the end user. This user-oriented nature had vast implications:

- The end user is often interested in aggregated reporting information, not in separate data items, and SQL pays a lot of attention to this aspect.
- No one can expect human users to explicitly control concurrency, integrity, consistency, or data type validity. That's why SQL pays a lot of attention to transactional guaranties, schemas, and referential integrity.

On the other hand, it turned out that software applications are not so often interested in in-database aggregation and able to control, at least in many cases, integrity and validity themselves. Besides this, elimination of these features had an extremely important influence on the performance and scalability of the stores. And this was where a new evolution of data models/stores began.

 

Although data modeling techniques are basically implementation agnostic, this is a list of the particular systems that we have come across:

1. Key-Value Stores: Tokyo Cabinet/Tyrant, Redis, Voldemort, Amazon Dynamo
2. Document Databases: MongoDB, CouchDB
3. Graph databases: Neo4J, InfoGrid, Infinite Graph
4. Object Stores: db4o, GemStone
5. XML databases: Exist, MarkLogic
6. Distributed Peer Stores: Cassandra, HBase, Riak, Hypertable, Accumulo

Let's take a brief look on these data models by explaining when to use these techniques & for which kind of project, particular Models are useful. While SQL databases are insanely useful tools, their monopoly in the last decades is coming to an end. But, the differences between NoSQL databases are much bigger than ever was between one SQL database and another. This means that it is a bigger responsibility on software architects to choose the appropriate one for a project right at the beginning.

1. **Key-Value Stores**: The main idea here is using a hash table where there is a unique key and a pointer to a particular item of data. -The Key/value model is the simplest and easiest to implement. But it is inefficient when you are only interested in querying or updating part of a value. *Typical applications:* Content caching *Strengths:* Fast lookups *Weaknesses:* Stored data has no schema. *Example application:* You are writing forum software where you have a home profile page that gives the user's statistics (messages posted, etc) and the last ten messages by them. The page reads from a key that is based on the user's id and retrieves a string of JSON that represents all the relevant information. A background process recalculates the information every 15 minutes and writes to the store independently.

**a. Redis (V2.4)**

| | |
|---|---|
| **Written in:** C/C++ | Simple values or hash tables by keys, |
| **Main point:** Blazing fast | but complex operations like ZREVRANGEBYSCORE. |
| **Protocol:** Telnet-like | INCR & co (good for rate limiting or statistics) |
| Currently without disk-swap (VM and Diskstore were abandoned) | Has sets (also union/diff/inter), lists (also a queue; blocking pop) , hashes (objects of multiple fields) , transactions |
| Disk-backed in-memory database | Simple values or hash tables by keys, |
| Master-slave replication | Values can be set to expire (as in a cache) |
| Sorted sets (high score table, good for range queries) | Pub/Sub lets one implement messaging. |

Few good features of Redis are: It is Simple (less than 1min compile time), Persistence, Built-in replication, Good language support, Client based sharding, Async persistence, Can be fully durable.

Few drawbacks are: It is Single-threaded (but no lock contention), Data must fit in memory.

**Best used:** For rapidly changing data with a foreseeable database size (should fit mostly in memory).

**For example:** Stock prices. Analytics. Real-time data collection. Real-time communication. And wherever you used memcached before.

**b. Tokyo Cabinet/Tyrant**

Tokyo products are open source – released under LGPL & are powerful, portable, practical– written in the standard C, optimized to POSIX. [7] Examples of products are (Fig.4.):

- Tokyo Cabinet – database library
- Tokyo Tyrant– database server
- Tokyo Dystopia– full-text search engine

Tokyo Cabinet is a database management library that allows for data storage in multiple formats. Formats include hash, table, fixed length rows.

Tokyo Tyrant is the network Daemon that sits on top of a Tokyo Cabinet database.

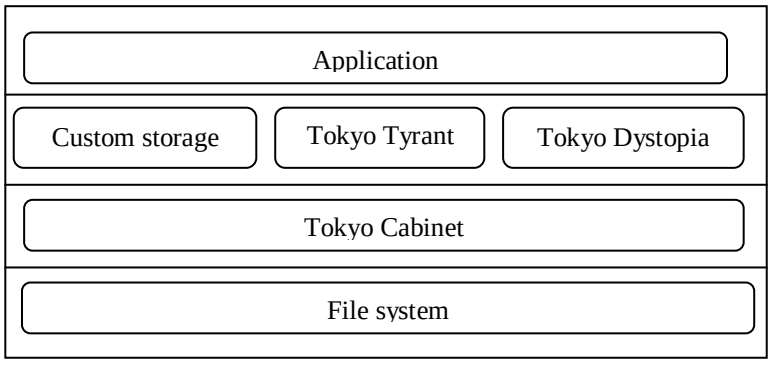


Fig.4. Tokyo Products

Tokyo Tyrant is a network server of Tokyo Cabinet- client/server model,multi applications can access one database, effective binary protocol.

Features of Tokyo cabinet:

- High performance: insert: 0.4 sec/1M records (2,500,000 qps), search: 0.33 sec/1M records 3,000,000 qps
- High concurrency : multi-thread safe, read/write locking by records
- High scalability: hash and B+tree structure = O(1) and O(log N), no actual limit size of a database file (to 8 exabytes)
- transaction: write ahead logging and shadow paging, ACID properties

 

- Various APIs: on-memory list/hash/tree, file hash/B+tree/array/table
- Script language bindings: Perl, Ruby, Java, Lua, Python, PHP, Haskell, Erlang, etc

Features: Easy to implement, Hash table behaves like memcached, only persistant, Very Fast, Lots of flexibility, Ability to use memcached , API allows for you to use with exsiting code, Replication, Embeddable.
Drawbacks: Not Durable, Single write thread, Minimal documentation, Small Community , Scalability Uncertainty.

**c. Voldemort**

Project Voldemort is an open source implementation of the basic parts of Dynamo's distributed key-value storage system. LinkedIn is using it in their production environment for "certain high-scalability storage problems where simple functional partitioning is not sufficient." Both, keys and values can be complex, compound objects as well consisting of lists and maps. When compared to relational databases—the simple data structure and API of a key-value store does not provide complex querying capabilities: joins have to be implemented in client applications while constraints on foreign-keys are impossible; besides, no triggers and views may be set up. However, they look at the following advantages of a simple key-value store:

- Only efficient queries are allowed.
- The performance of queries can be predicted quite well.
- Data can be easily distributed to a cluster or a collection of nodes.
- In service oriented architectures it is not uncommon to have no foreign key constraints and to do join in the application code as data is retrieved and stored in more than one service or data source.
- Gaining performance in a relational database often leads to de-normalized data structures or storing more complex objects as BLOBs or XML-documents.
- Application logic and storage can be separated nicely (in contrast to relational databases where application developers might get encouraged to mix business logic with storage operation or to implement business logic in the database as stored procedures to optimize performance).
- There is no such impedance mismatch between the object-oriented paradigm in applications and paradigm of the data store as it is present with relational databases.

**d. Amazon's Dynamo**

Amazon runs a world-wide e-commerce platform that serves tens of millions customers at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. Following are the key factors that Amazon has cited in their paper for the motivation behind Dynamo:

- Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. These services require that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.
- Due to the use of commodity hardware, software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.
- Amazon's platform has a very diverse set of applications with different storage requirements.
- Most of the services only store and retrieve data by primary key and do not require the complex querying and management functionality offered by an RDBMS.
- The available replication technologies are limited and typically choose consistency over availability. Although many advances have been made in the recent years, it is still not easy to scale-out databases or use smart partitioning schemes for load balancing.

Dynamo, a highly available key-value store addresses the above requirements to provide an "always-on" experience.

**2. Document Databases**: Document stores can be considered to be next step to the key-value stores because they store more complex data than the key-value stores. They store "documents" which allow values to be nested documents or lists as well as scalar values, and the attribute names are dynamically defined for each document at runtime. These were inspired by Lotus Notes. The model is basically versioned documents that are collections of other key-value collections. The semi-structured documents are stored in formats like JSON. Document databases support querying more efficiently. *Typical applications:* Web applications *Strengths:* Tolerant of incomplete data *Weaknesses:* Query performance, no standard query syntax. *Example application:* You are creating software that creates profiles of refugee children with the aim of reuniting them with their families. The details you need to record for each child vary tremendously with circumstances of the event and they are built up piecemeal, for example a young child may know their first name and you can take a picture of them but they may not know their parent's first names. Later a local may claim to recognize the child and provide you with additional information that you definitely want to record but until you can verify the information you have to treat it sceptically.

**a. SimpleDB**

SimpleDB is a multi-user transactional database server written in Java, which interacts with Java client programs via JDBC. Amazon SimpleDB is a highly available and flexible non-relational data store that offloads the work of database administration. Developers simply store and query data items via web services requests and Amazon SimpleDB does the rest. Features:

- Low touch: Amazon SimpleDB automatically manages infrastructure provisioning, hardware and software maintenance, replication and indexing of data items, and performance tuning.

 

- High Available: Amazon SimpleDB automatically creates multiple geographically distributed copies of each data item you store. This provides high availability and durability.
- Flexible: Amazon SimpleDB automatically creates multiple geographically distributed copies of each data item you store. This provides high availability and durability.
- Easy to use: allows you to quickly add data and easily retrieve or edit that data through a simple set of API calls.
- Designed for use with other Amazon Web Services – Amazon SimpleDB is designed to integrate easily with other AWS services such as Amazon S3 and EC2, providing the infrastructure for creating web-scale applications.
- Secure – Amazon SimpleDB provides an https end point to ensure secure, encrypted communication between your application or client and your domain[8].

**b. CouchDB (V1.2)**

| | |
|---|---|
| **Written in:** Erlang | Previous versions of documents are available |
| **Main point:** DB consistency, ease of use | Crash-only (reliable) design |
| **License:** Apache | Needs compacting from time to time |
| **Protocol:** HTTP/REST | Views: embedded map/reduce |
| Bi-directional replication, continuous or ad-hoc, with conflict detection, thus, master-master replication. | Attachment handling |
| MVCC - write operations do not block reads | Server-side document validation possible |
| Real-time updates via '_changes' | Authentication possible |

CouchDB, from Apache, is a "collection" of documents (similar to SimpleDB) whose data model is richer than SimpleDB. The main abstraction and data structure in CouchDB is a document. Documents consist of named fields that have a key/name and a value.

**Best used:** For accumulating, occasionally changing data, on which pre-defined queries are to be run. Places where versioning is important.

**For example:** CRM, CMS systems. Master-master replication is an especially interesting feature, allowing easy multi-site deployments.

**c. MongoDB (2.2)**

| | |
|---|---|
| **Written in:** C++ | Performance over features |
| **Main point:** Retains some friendly properties of SQL. | Journaling (with --journal) is best turned on |
| **Protocol:** Custom, binary (BSON) | On 32bit systems, limited to ~2.5Gb |
| Master/slave replication (auto failover with replica sets) | An empty database takes up 192Mb |
| Sharding built-in | GridFS to store big data + metadata (not actually an FS) |
| Queries are javascript expressions | Has geospatial indexing |
| Run arbitrary javascript functions server-side | Data center aware |
| Better update-in-place than CouchDB | Uses memory mapped files for data storage |

MongoDB is also a document store that has many similarities to CouchDB. It is a schema-free document store that contains one or more collections consisting of documents. Like CouchDB, MongoDB also provided indexes on collections and supports map-reduce for complex aggregations across documents. But it differs from CouchDB in a number of ways. MongoDB supports dynamic queries with automatic use of indices, like RDBMSs. MongoDB allows specifying indexes on document fields of a collection.

**Best used:** If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks.

**For example:** For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.

**3. Graph Databases**: Instead of tables of rows and columns and the rigid structure of SQL, a flexible graph model is used which, again, can scale across multiple machines. NoSQL databases do not provide a high-level declarative query language like SQL to avoid overtime in processing. Rather, querying these databases is data-model specific. Many of the NoSQL platforms allow for RESTful interfaces to the data, while other offers query APIs. *Typical applications:* Social networking, Recommendations *Strengths:* Graph algorithms e.g. shortest path, connectedness, n degree relationships, etc. *Weaknesses:* Has to traverse the entire graph to achieve a definitive answer. Not easy to cluster. *Example application:* Any application that requires social networking is best suited to a graph database. These same principles can be extended to any application where you need to understand what people are doing, buying or enjoying so that you can recommend further things for them to do, buy or like. Any time you need to answer the question along the lines of "What restaurants

 

do the sisters of people who are over-40, enjoy skiing and have visited Kenya dislike?" a graph database will usually help.

**a. Neo4j (V1.5M02)**

| | |
|---|---|
| **Written in:** Java | Indexing of nodes and relationships |
| **Main point:** Graph database - connected data | Nice self-contained web admin |
| **Protocol:** HTTP/REST (or embedding in Java) | Advanced path-finding with multiple algorithms |
| Standalone, or embeddable into Java applications | Indexing of keys and relationships |
| Full ACID conformity (including durable data) | Optimized for reads |
| Both nodes and relationships can have metadata | Has transactions (in the Java API) |
| Integrated pattern-matching-based query language ("Cypher") | Online backup, advanced monitoring and High Availability is AGPL/commercial licensed |
| Also the "Gremlin" graph traversal language can be used | Scriptable in Groovy |

Where big data management is concerned, Cassandra has a number of advantages over relational database management systems (RDBMSs) including:

- Superior write performance for data velocity
- Strong data type support for data variety
- Linear s-calability with horizontal scale-out for data volume
- Very fast response times for both reads and writes

**Best used:** For graph-style, rich or complex, interconnected data. Neo4j is quite different from the others in this sense.
**For example:** For searching routes in social relations, public transport links, road maps, or network topologies.

**b. The InfoGrid Software Stack**

In the left column, the traditional, relational database-centric software stack for web applications is shown: the application is built on an JEE Application Server, which utilizes a relational database (RDBMS) for persistence.
On the right hand side, the InfoGrid software stack is shown for web applications. InfoGrid sits on top of the JEE Application Server and provides additional services to the web application. One could consider InfoGrid simply a library of JAR files that are added to a regular JEE project, which it is. (This makes it easy to fit InfoGrid-based applications into the regular Java development and deployment cycle.) However, the scope of InfoGrid goes far beyond the scope of typical Java libraries.
By using InfoGrid:

- the size of the web application itself can shrink, as InfoGrid provides valuable higher-level services than JEE Application Servers do themselves
- applications have a choice in the persistence layer: they can write to relational databases, just like in a RDBMS-centric architecture, but can also write to grid technology, or other NoSQL Stores. InfoGrid provides a common abstraction layer.
- XPRISO and the InfoGrid Graph Database (Grid) Project enable applications to peer with other InfoGrid ap-plications or other applications supporting XPRISO.

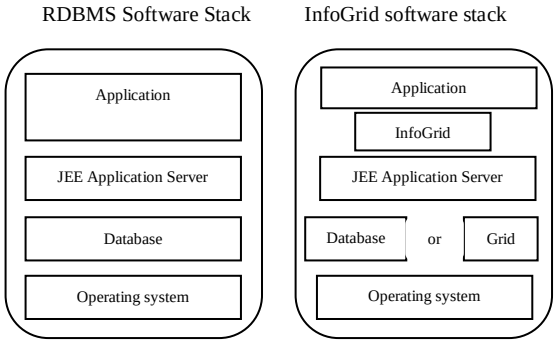


Fig.5. Software stack of RDBMS & InfoGrid

**4. XML Database**: *Typical applications:* Publishing *Strengths:* Mature search technologies, Schema validation *Weaknesses:* No real binary solution, easier to re-write documents than update them. Example application: A publishing company that uses bespoke XML formats to produce web, print and eBook versions of their articles. Editors need to quickly search either text or semantic sections of the markup (e.g. articles whose summary contains diabetes, where the author's institution is Liverpool University and Stephen was a revising editor at some point in the document history). They store the XML of finished articles in the XML database and wrap it in a readable-URL web service for the document production systems. Workflow metadata (which stage a manuscript is in) is held in a separate RDBMS. When system-wide changes are required, XQuery updates bulk update all the documents to match the new format.

**a. eXist:**

 

**eXist** is an native XML database, retrieval engine and portalware.
Features: XQuery, XUpdate, XML:DB API, xml-rpc support, REST support, WebDav, PHP API (needs dowloading from CVS), good Cocoon Integration, etc. eXist-db can be also defined as a XML database featuring efficient, index-based XQuery processing, extensions for keyword search, XUpdate support, XSLT support, XForms support, REST and tight integration with existing XML development tools.

**b. Marklogic**

For more than a decade, MarkLogic has delivered a powerful and trusted enterprise-grade NoSQL (Not Only SQL) database that enables organizations to turn all data into valuable and actionable information. Key features include ACID transactions, horizontal scaling, real-time indexing, high availability, disaster recovery, government-grade security. MarkLogic Server is an Enterprise NoSQL database.
It is a document-centric, transactional, search-centric, structure-aware, schema-agnostic, XQuery- and XSLT-driven, high performance, clustered, database server.
Even though MarkLogic is **not** a relational database, the core architectural features of- a MarkLogic-based application will seem familiar to anyone with an RDBMS background.

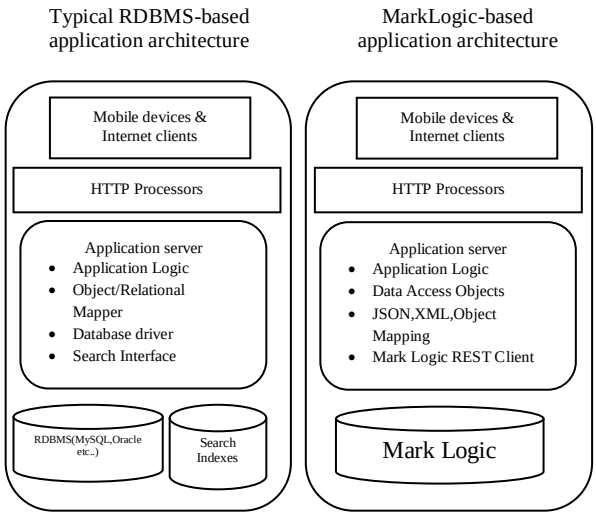


Fig.6. Architecture of RDBMS vs MarkLogic

**5.Distributed Peer stores**: Bigtable, as described by Google, is "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers". Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. *Typical applications:* Distributed file systems *Strengths:* Fast lookups, good distributed storage of data. *Weaknesses: Very low-level API. Example application:* You have a news site where any piece of content: articles, comments, author profiles, can be voted on and an optional comment supplied on the vote. You create one store per user and one store per piece of content, using a UUID as the key (generating one for each piece of content and user). The user's store holds every vote they have ever made while the content "bucket" contains a copy of every vote that has been made on the piece of content. Overnight you run a batch job to identify content that users have voted on, you generate a list of content for each user that has high votes but which they have not voted on. You then push this list of recommended articles into the user's "bucket".

**a. Riak (V1.2)**

| | |
|---|---|
| **Written in:** Erlang & C, some JavaScript | Secondary indices: but only one at once |
| **Main point:** Fault tolerance | Large object support (Luwak) |
| **Protocol:** HTTP/REST or custom binary | Comes in "open source" and "enterprise" editions |
| Stores blobs | Full-text search, indexing, querying with Riak Search |
| Tunable trade-offs for distribution and replication | In the process of migrating the storing backend from "Bitcask" to Google's "LevelDB" |
| Pre- and post-commit hooks in JavaScript or Erlang, for validation and security. | Masterless multi-site replication replication and SNMP monitoring are commercially licensed |
| Map/reduce in JavaScript or Erlang | Links & link walking: use it as a graph database |

**Best used:** If you want something Dynamo-like data storage, but no way you're gonna deal with the bloat and complexity. If you need very good single-site scalability, availability and fault-tolerance, but you're ready to pay for multi-site replication.
**For example:** Point-of-sales data collection. Factory control systems. Places where even seconds of downtime hurt. Could be used as a well-update-able web server.

**b. HBase (V0.92.0)**

HBase, from Apache, is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable. It is used in many data-driven websites and, in 2010, was elected for implementing Facebook's Messaging Platform.

 

- Written in: Java
- Main point: Billions of rows X millions of columns
- Protocol: HTTP/REST (also Thrift)

HBase features of note are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

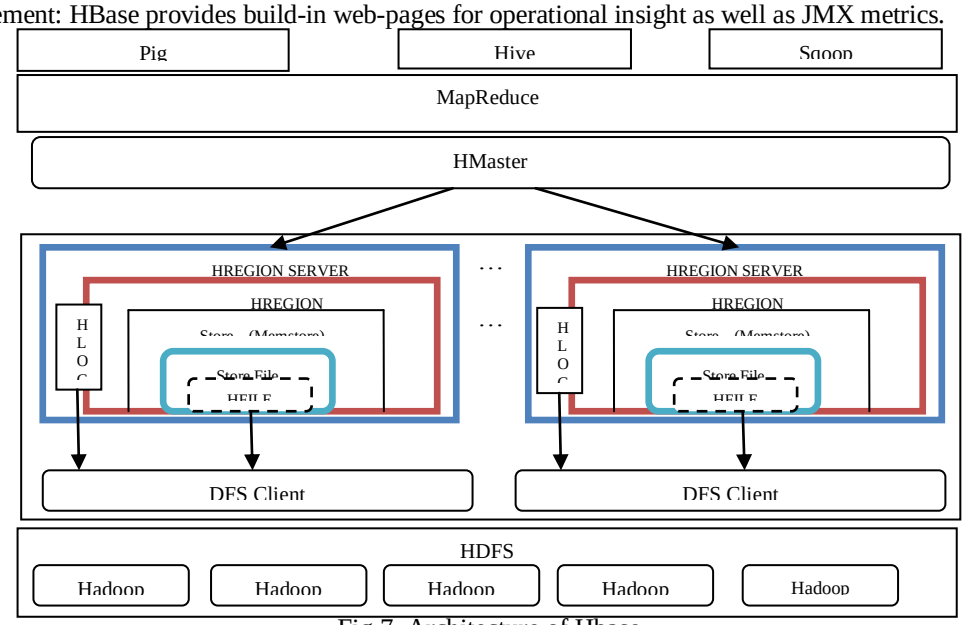


Fig 7. Architecture of Hbase

**When Hbase is used:**

- First, make sure you have enough data. If you have hundreds of millions or billions of rows, then HBase is a good candidate. If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle.
- Second, make sure you can live without all the extra features that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.) An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver, for example. Consider moving from an RDBMS to HBase as a complete redesign as opposed to a port.
- Third, make sure you have enough hardware. Even HDFS doesn't do well with anything less than 5 DataNodes (due to things such as HDFS block replication which has a default of 3), plus a NameNode.
- HBase can run quite well stand-alone on a laptop - but this should be considered a development configuration only.

In fig.7, Region: A subset of table's rows,
RegionServer(slave) : Serves data for reads and writes
Master: -Responsible for coordinating the slaves, Assigns regions, detects failures of Region Servers, Control some admin function.

**For example:** Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.

**Used by** : Adobe, Facebook, Twitter, Yahoo, Gmail, Google maps etc.

**c. Cassandra (1.2)**

| | |
|---|---|
| **Written in:** Java | **Protocol:** Thrift & custom binary CQL3 |
| **Main point:** Best of BigTable and Dynamo | Data can have expiration (set on INSERT) |
| Can be used as a distributed hash-table, with an "SQL-like" language, CQL (but no JOIN!) | Writes can be much faster than reads (when reads are disk-bound) |
| Tunable trade-offs for distribution and replication | Map/reduce possible with Apache Hadoop |
| Querying by column, range of keys (Requires indices on anything that you want to search on) | All nodes are similar, as opposed to Hadoop/HBase |

 

| | |
|---|---|
| BigTable-like features: columns, column families | Cross-datacenter replication |

Cassandra, originally developed by Facebook, adopts ideas and concepts of both, Amazon's Dynamo as well as Google's Bigtable. It has been adopted by other companies like Twitter, Dig and Rackspace. It can be described as a distributed storage system for managing structured data that is designed to scale to a very large size.

The 'Inbox Search' problem led to the initial design and development of Cassandra at Facebook. The users can exchange personal messages with their contacts which appear in the inbox of a recipient and the problem was to find an efficient way of st-oring, indexing and searching these messages.

**Best used:** When you write more than you read (logging).

**For example:** Banking, financial industry (though not necessarily for financial transactions, but these industries are much bigger than that.) Writes are faster than reads, so one natural niche is data analysis.

Features are: Larger Community then most other NoSQL Solutions, Very easy to add nodes, scale as needed, Configurable write settings, Replication/HA, Changing schema can be easy.

Drawbacks are: Inconsistent Experience (Thrift/Api related), Eventually Consistent, My Tests show mixed performance results, Benefit from lots of nodes, Not as polished as other

Solutions, Durability concerns, Limited config, Higher ramp up time then other solutions.

**d. Hypertable (0.9.6.5)**

| | |
|---|---|
| **Written in:** C++ | Can search by key, by cell, or for values in column families. |
| **Main point:** A faster, smaller HBase | Search can be limited to key/column ranges. |
| **License:** GPL 2.0 | Sponsored by Baidu |
| **Protocol:** Thrift, C++ library, or HQL shell | Retains the last N historical values |
| Implements Google's BigTable design | Tables are in namespaces |
| Run on Hadoop's HDFS | Map/reduce with Hadoop |

**Best used:** If you need a better HBase.

**For example:** Same as HBase, since it's basically a replacement: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.

**6. Object stores**: *Typical applications:* Finance systems *Strengths:* Matches OO development paradigm, low-latency ACID, mature technology *Weaknesses:* Limited querying or batch-update options. *Example application:* A global trading company has a monoculture of development and wants to have trades done on desks in Japan and New York pass through a risk checking process in London. An object representing the trade is pushed into the object store and the risk checker is listening to for appearance or modification of trade objects. When the object is replicated into the local European space the risk checker reads the Trade and assesses the risk. It then rewrites the object to indicate that the trade is approved and generates an actual trade fulfilment request. The trader's client is listening for changes to objects that contain the trader's id and updates the local detail of the trade in the client indicating to the trader that the trader has been approved. The trading system will consume the trade fulfilment and when the trade elapses or is fulfilled feeds back the information to the risk assessor.

**a. db4o:**

db4o (database for objects) is the open source object database, native to Java and .NET.It smoothly integrates into an OO system. It cuts down on development time by skipping the costly object-relational mapping .It stays highly reliable and performant.

Concepts of db4o include: Object Container, Querying, Transaction, Object Identity ,Indexing, Inheritance. Free software, available under the GPL and under a commercial license. Ideal for embedded use, e.g., in software running on mobile or medical devices, in packaged software & for real-time systems.

Key Features:

- The One-Line-of-Code-Database (One line of code stores any object; Class model = object schema; Smooth production process).
- Embeddable (Zero administration; Automatic schema versioning; 400 KB footprint)
- Multiple platform support (Native to Java and .NET; Mobile, PCs and servers; Runs cross-platform).
- Brings more OO to the database(Object-oriented replication; Native Queries; Object Manager browser)

Key Benefits:

| | |
|---|---|
| Slashes 90% of cost to develop persistence. | Build lean & truly object oriented software. |
| 10% faster to market with your application | Build distributed, fully synchronized data architectures. |
| Runs up to 44x faster than conventional systems. | Fewer errors, better refractorability & software longevity. |
| Deployable in large volumes without local administration. | |

**b. GemStone**

The GemStone data management system, developed at Servio Logic, was one of the first and simplest commercial OODBMS products. It is based on Smalltalk, with very few extensions. GemStone merges object-oriented language concepts with those of database systems. And provides an object-oriented database language called OPAL which is used

 

for data definition, data manipulation and general computation. GemStone provides a wide range of services to help you build objects-based information
systems.

GemStone

- is a multi-user object server also a programmable server object system
- manages a large-scale repository of objects
- supports partitioning of applications between client and server, queries and indexes for large-scale object processing, transactions and concurrency control in the object repository, connections to outside data sources
- provides object security and account management, provides services to manage the object repository

GemStone provides services that can:

- support flexible backup and restore procedures,
- recover from hardware and network failures; perform object recovery when needed,
- tune the object server to provide high transaction rates by using shared memory and asynchronous I/O processes,
- accommodate the addition of new machines and processors without recoding the system,
- make controlled changes to the definition of the business and application objects in the system
-

## IV. COMPARISON BETWEEN SQL DATABASE & NOSQL DATABASE[11]

| | SQL Databases | NoSQL Databases |
|---|---|---|
| Types | One type (SQL database) with minor variations | Many different types including key-value stores, document databases, wide-column stores, and graph databases |
| Development History | Developed in 1970s to deal with first wave of data storage applications | Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale, replication and unstructured data storage |
| Examples | MySQL, Postgres, Oracle Database | MongoDB, Cassandra, HBase, Neo4j |
| Data Storage Model | Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Separate data types are stored in separate tables, and then joined together when more complex queries are executed. For example, "offices" might be stored in one table, and "employees" in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables together to get all the information necessary. | Varies based on NoSQL database type. For example, key-value stores function similarly to SQL databases, but have only two columns ("key" and "value"), with more complex information sometimes stored within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically. |
| Schemas | Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline. | Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically. |
| Scaling | Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required. | Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The NoSQL database automatically spreads data across servers as necessary |
| Development Model | Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database) | Open-source |
| Supports Transactions | Yes, updates can be configured to complete entirely or not at all | In certain circumstances and at certain levels (e.g., document level vs. database level) |
| Data Manipulation | Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE… | Through object-oriented APIs |
| Consistency | Can be configured for strong consistency | Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra) |

 

## V. CONCLUSION

RDBMS is a great tool for solving ACID problems: a) When data validity is super important and b) When you need to support dynamic queries. NoSQL is a great tool for solving data availability problems: a) When it's more important to have fast data than right data and b) When you need to scale based on changing requirements. Pick the right tool for the job. In this paper we have given a brief introduction about NoSQL at the start & later we have presented the NoSQL models with their description & tried to answer the questions of user's, "When XYZ data model is best suited & what are the features of XYZ?"For DB beginners, our paper will give the basic knowledge of DBMS, DB models & little bit history of it. As per researchers point of view, we hope our paper will surely help them to develop new projects by overcoming the drawbacks of existing or work on the existing one & add up features if possible.

Lastly it's important to know that both SQL and NoSQL has been great inventions over time in order to keep data storage and retrieval optimized and smooth. Criticizing any one of them will not help the cause. If there is a buzz of NoSQL these days, it doesn't mean it is a silver bullet to all your needs. Both technologies are best in what they do. It is up to a developer to make a better use of them depending on the situations and needs. There are many more projects under these NOSQL datastores but we have dealt only few but the good ones. In our further paper we will work on other projects & come up with new stuff.

## REFERENCES


[1] Michael Stonebraker: SQL databases v. NoSQL databases. Commun. ACM 53(4): 10-11 (2010)

[2] Dr. Mohamed Yagoub Mohamed- " Database Management System (DBMS)" .

[3] Strozzi, Carlo - "NoSQL – A relational database management system" , 2007–2010. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[4] Brewer, Eric A.: Towards Robust Distributed Systems. Portland, Oregon, July 2000.---CAP theorem

[5] Amir H. Payberah, " NoSQL Databases" April 26, 2012 .

[6] WHITE PAPER on "Why NoSQL? " By DataStax Corporation September 2012.

[7] Mikio Hirabayashi "Introduction to Tokyo Products"; http://tokyocabinet.sourceforge.net/.

[8] http://aws.amazon.com/simpledb/

[9] Rick Cattell " Scalable SQL and NoSQL Data Stores" Originally published in 2010, last revised December 2011.—Scalaris-membase

[10] Lakshman, Avinash: Cassandra - A structured storage system on a P2P Network. August 2008. – Blog post of 2008-08-25 - http://www.facebook.com/note.php?note_id=24413138919.

[11] Keith W. Hare "A Comparison of SQL and NoSQL Databases" JCC Consulting, Inc. Convenor, ISO/IEC JTC1 SC32 WG3.

[12] Kristof Kovacs has given features of the projects/datastores in detail in website http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis . Features written in tables are extracted from tables.

[13] http://maxivak.com/newsql-is-coming/ NewSQL is coming : Max Ivak Personal Site "Differences between "NoSQL" databases: Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase…"

[14] Santhosh Kumar Gajendran "A Survey on NoSQL Databases".

[15] Luís Ferreira, Universidade do Minho "Bridging the gap between SQL and NoSQL" Universidade do Minho - Master Course on Informatics - State of the Art Reports 2011, MI-STAR 2011, Pages 187–197.

[16] http://www.angelfire.com/mo/yagoub

[17] BeJUG – "NoSQL with HBase and Hadoop" on 17/6/2010 www.outerthought.com.

[18] Christof Strauch, Prof. Walter Kriha "NoSQL Databases"

[19] White paper on "Why NoSql" CouchBase.

[20] Anuj Shetye , Vinay Boddula ppt "Distributed RDF data store on HBase".

[21] Rick Cattell "Scalable SQL and NoSQL Data Stores" Originally published in 2010, last revised December 2011.


 