Database Technology (Nosql)

Rupesh suresh vedante*
Mumbai university

Ashwini Sampat Salunkhe
Mumbai university

Abstract— ***NoSQL databases have gained popularity in the recent years and have been successful in many production systems Motivated by requirements of Web 2.0 applications, a plethora of non-relational databases raised in recent years. Since it is very difficult to choose a suitable database for a specific use case, this paper evaluates the underlying techniques of NoSQL databases considering their applicability for certain requirements. These systems are compared by their data models, query possibilities, concurrency controls, partitioning and replication opportunities.***

Keywords— Include at least 5 keywords or phrases

I. *I* INTRODUCTION

The current NoSQL trend is intended by applications stemming mostly of the Web 2.0 domain. Number of these applications has storage necessities that exceed capacities and prospects of relational databases.

In the past SQL databases were used for nearly each storage problem, even if a data model did not match the relational model well. The object-relational impediment mismatch is one example, the transformation of graphs into tables another one for using a data model during wrong approach. These advantages to an increasing complexity by using expensive mapping frameworks and sophisticated algorithms. Even if a data model can simply be coated by the relational one, the large feature set offered by SQL databases is an unneeded overhead for straightforward tasks like logging. The strict relative schema is often a burden for internet applications like blogs that include many alternative varieties of attributes. Text, comments, pictures, videos, source code and other information have to be stored within multiple tables. Since such internet applications are unit terribly agile, underlying databases got to be versatile still so as to support straightforward schema analysis. Adding or removing a feature to a blog is not possible without system inaccessibility if an on-line database is getting used.

The increasing quantity of information in the web is a problem which has to be considered by successful web pages like the ones of Facebook, Amazon and Google. Besides coping with tera- and peta bytes of information, large browse and write requests have to be compelled to be responded with none noticeable latency. So as to contend with these needs, these corporations maintain clusters with thousands of trade goods hardware machines. Due to their normalized information model and their full ACID support, relational databases are not appropriate in this domain, as a result of joins and locks influence performance in distributed systems negatively. Additionally to high performance, high handiness could be an elementary demand of the many corporations. Amazon guarantees for its services possibility of a minimum of 99.9% throughout a year [1]. Therefore, databases should be simply replicable and have to provide an integrated failover mechanism to share with node or datacentre failures. They also must be able to balance read requests on multiple slaves to contend with access peaks which can exceed the capacity of a one server. Since replication techniques offered by relational databases are restricted and these databases are generally supported consistency rather than possibility, these needs will only be achieved with extra effort and high experience [1]. Due to these needs, several corporation and organizations developed own storage systems, that are currently classified as NoSQL databases. Since each store is specialized on the precise wants of their principals, there is no cure on the market covering all of the higher than mentioned needs. Therefore, it is very terribly to pick out one database out of the embarrassment of systems that is the most fitted for an exact use case.

Even if NoSQL databases have already been introduced and compared within the past [2] [3] [4] [5], no use case oriented survey is available. Since single options of bound databases are changing on a weekly basis, an evaluation of the various option of certain stores is superannuated at the instant it is revealed. Therefore, it's necessary to contemplate the impact of the underlying techniques on specific use cases so as to produce a sturdy summary. This paper highlights the foremost vital criteria for a database selection, introduces the underlying techniques and compares a wider vary of open source NoSQL databases as well as graph databases, too.

In order to guage the underlying techniques of those systems, the foremost vital options for resolution the higher than mentioned needs have to be compelled to be classified. Since the relational data model is taken into account as not appropriate for certain use cases, chapter two can examine structure and suppleness of various data models offered by NoSQL systems. Afterwards, query

ISSN: 2278 – 1323

prospects of those stores and their impact on system complexity will be analysed in chapter three. In order to contend with several parallel browse and write requests, some stores loose concurrency restrictions. Completely different method of these systems for handling concurrent requests is inspected in chapter four. Since immense amounts of information and high performance serving exceed the capacities of single machines, partitioning strategies of NoSQL databases are analysed in chapter five. Replication techniques and their effects on handiness and consistency are examined in chapter six.

II. DATA MODEL

Mostly NoSQL databases dissent from relational databases in their data model. These systems are classified in this during this analysis into 4 groups.

A. *Key Value Stores*

Key value stores are similar to maps or dictionaries wherever information is addressed by a novel key. Since values are uninterpreted byte arrays, which are utterly opaque to the system, keys are the only way to retrieve stored data. Values are isolated and freelance from one another wherefore relationships should be handled in application logic. Due to this terribly straightforward simple data structure, key value stores are utterly schema free. New values of any kind can be added at runtime without conflicting any other keep information and without influencing system handiness. The grouping of key value pairs into assortment is the only offered possibility to add some kind of structure to the data model. Key value stores are useful for simple operations, which are based on key attributes only. In order to speed up a user specific rendered webpage, components of this page can be calculated before and served quickly and easily out of the store by user IDs when required. Since most key value stores hold their dataset in memory, they are oftentimes used for caching of longer intensive SQL queries.

B. *Document Stores*

Document Stores databases are those NoSQL databases which use records as documents. This type of database store unstructured (text) or semi-structured (XML) documents which are usually hierarchal in nature. Here each document consists of a set of keys and values which are almost same as there in the Key Value databases. Each database residing in the document stores points to its fields using pointers as it uses the technique of hashing. Document Stores Databases are schema free and are not fixed in nature. Databases point to its value using some unique key residing in its database. This consists of an array of databases .Document stores offer multi attribute lookups on records which may have complete different kinds of key value pairs. Therefore, these systems are very convenient in data integration and schema migration tasks. Most popular use cases are real time analytics, logging and the storage layer of small and flexible websites like blogs. The most prominent document stores are CouchDB [9], MongoDB [10] and Riak [11]. Riak offers links, which can be used to model relationships between documents.

C. *Graph Databases*

A graph database, also called a graph-oriented database, is a type of NoSQL database that uses graph theory to store, map and query relationships.

Graph theory is the study of points and lines. In particular, it involves the ways in which sets of points, called vertices, can be connected by lines or arcs, called edges. Graphs in this context differ from the more familiar coordinate plots that portray mathematical relations and functions. Graphs are classified according to their complexity, the number of edges allowed between any two vertices, and whether or not directions (for example, up or down) are assigned to edges. Various sets of rules result in specific properties that can be stated as theorems. Graph theory has proven useful in the design of integrated circuits (IC s) for computers and other electronic devices. These components, more often called chip s, contain complex, layered microcircuits that can be represented as sets of points interconnected by lines or arcs. Using graph theory, engineers develop chips with maximum component density and minimum total interconnecting conductor length. This is important for optimizing processing speed and electrical efficiency. Twitter stores many relationships between people in order to provide their tweet following service. These one-way relationships are handled within their own graph database Flock DB [20] which is optimized for very large adjacency lists, fast reads and writes.

Use cases for graph databases are location based services, knowledge representation and path finding problems raised in navigation systems, recommendation systems and all other use cases which involve complex relationships. Property graph databases are more suitable for large relationships over many nodes, whereas RDF is used for certain details in a graph. Flock DB is suitable for handling simple I-hop-neighbor relationships with huge scaling requirements.

III. QUERY POSSIBILITIES

The main idea behind basic NoSQL databases is to focus solely on the storage of arbitrary values indexed by keys and let the application worry about the business logic and entity relationships. This allows NoSQL databases to be significantly less complex and more flexible than traditional RDS databases at the expense of moving complexity, such as

ISSN: 2278 – 1323

transaction systems and schema management, into the application itself. While this might seem like a step back from RDS systems where this functionality is offered by default, the big alure for NoSQL systems is the performance and scalability benefits that such simplification entails, especially for those operations. Due to their simple data model, APIs of key value stores provide key based put, get and delete operations only. Any query language would be an unnecessary overhead for these stores. If additional query functionalities are required, they have to be implemented on the application layer which can quickly lead to much more system complexity and performance penalties. Therefore, key value stores should not be used, if more complex queries or queries on values are required. Very useful in the domain of web applications are REST interfaces. Heterogeneous clients can directly interact with the store in a uniform way, while requests can be load balanced and results can be cached by proxies. Membase is the only key value store, which offers a REST API natively.

NoSQL solutions should provide a rich set of possibilities for locking granularity. Locking should be possible but access to snapshots while data is being written, providing consistent views at all times regardless of concurrent write activity, should be allowed. Locking at all levels, such as document, should be supported and applied according to context. Column family stores only provide range queries and some operations like "in" , "and/or" and regular expression, if they are applied on row keys or indexed values. Even if every column family store offers a SQL like query language in order to provide a more convenient user interaction, only row keys and indexed values can be considered in where-clauses as well. Since these languages are specialized on the specific features of their stores, there is no common query language for column family stores available yet.

As clusters of document and column family stores are able to store huge amounts of structured data, queries can get very inefficient if a single machine has to process the required data.

Therefore, all document and column family stores provide graph databases can be queried in two different ways. When relationships are the important aspect to the data, graph databases shine. The data does not have to be “big” for the graph database to provide significant performance benefits over other database technologies. The data itself can be homogenous, such as all people and their relationships as in a “social graph” or heterogeneous. Fast checks of how many degrees nodes are from each other or list pulls of all those a certain number of degrees apart are workloads that would be slow, nested table self joins in relational databases and probably worse in the other NoSQL data models. Graph Databases yield very consistent execution times that are not dependent on the number of nodes in the graph. It might be tempting to select from these solutions without giving much “enterprise” thought to the matter. However, an organization that sees the value in a NoSQL database for one application could soon need or use several NoSQL implementations. You could adopt multiple NoSQL databases to satisfy different requirements (e.g. a Document Database and a Key-Value Store). Or you can use a NoSQL database that functions as both a competent Key-Value Store and a competent Document Store. It may also be advantageous to have skills around a single multi-purpose NoSQL database.

In this paper, we aim to search for the answer of the question how to process web data quickly. Thus, we propose a method to exploit a NoSQL database, specifically MongoDB, to store and query RDF. MongoDB is chosen because it is one of widely used NoSQL databases. The system first invokes NoSQL API to retrieve MongoDB data in JSON format. Then, the JSON parser module converts JSON data to RDF data. We evaluate our design and implementation by using the Berlin SPARQL Benchmark, which is one of the most widely accepted benchmarks for comparing the performance of three RDF storage systems which include Apache Jena TDB (native RDF store), MySQL (relational database), And MongoDB (NoSQL database).

IV. CONCURRENCY CONTROL

Several users have access to an information supply in parallel; ways for avoiding inconsistency supported conflicting operations square measure necessary. Ancient databases use hopeless consistency ways with exclusive access on a dataset. These ways square measure appropriate, if prices for protection square measure low and datasets aren't blocked for a protracted time. Since locks square measure terribly expensive in information clusters that square measure distributed over giant distances and plenty of internet applications got to support terribly high browse request rates, hopeless consistency ways will cause large performance loss.

Multiversion concurrency management (MVCC) relaxes strict consistency in favour of performance. Simultaneous access isn't managed with locks however by organization of the many unmodifiable written account ordered versions.

Since datasets aren't reserved for exclusive access, browse requests will be handled by providing the most recent version of a worth, whereas a simultaneous method accomplishes write operations on identical dataset in parallel. So as to deal with 2 or a lot of conflicting write operations, each method stores, further to the new worth, a link to the version the method browse before.

Therefore, algorithms on information or shopper facet have the chance to resolve conflicting values by completely different ways. HBase, Hyptertable, Bigdata and GraphDB use the storage of various versions not just for conflict breakdown however additionally for providing versioning.

ISSN: 2278 – 1323

Besides consistency cut backs MVCC additionally causes higher space for storing needs, as a result of multiple versions of 1 worth square measure hold on in parallel. Further to an employee, that deletes not used versions, conflict breakdown algorithms square measure required to agitate inconsistencies. Therefore, MVCC causes higher system quality.

In order to support transactions while not reserving multiple datasets for exclusive access, optimistic protection is provided by several stores. Before modified knowledge is committed, every dealing checks, whether or not another transactions created any conflicting modifications to identical datasets. Just in case of conflicts, the dealing is rolled back. This idea functions well, once updates square measure dead seldom and possibilities for conflicting transactions square measure low. During this case, checking and rolling back is cheaper than protection datasets for exclusive access.

V. PARTITIONING

At the time huge amounts of data and very high read and write request rates exceed the capacity of one server, databases have to be partitioned across database clusters. Due to their normalized data model and their ACID guarantees, relational databases do not scale horizontally. Doubling the amount of relational database server does not double the performance of the cluster. Due to that lack, big web 2.0 companies like Google, Facebook and Amazon developed their own so called web-scale databases which are designed to scale horizontally and therefore satisfy the very high requirements on performance and capacity of these companies.

NoSQL databases differ in their way they distribute data on multiple machines. Since data models of key value stores, document stores and column family stores are key oriented, the two common partition strategies are based on keys, too. The first strategy distributes datasets by the range of their keys. A routing server splits the whole keyset into blocks and allocates these blocks to different nodes. Afterwards, one node is responsible for storage and request handling of his specific key ranges. In order to find a certain key, clients have to contact the routing server for getting the partition table.

This strategy has its advantages in handling range queries very efficiently, because neighbour keys are stored with high percentile on the same server. Since the routing server is responsible for load balancing, key range allocation and partition block advices, the availability of the whole cluster depends on the failure proneness of that single server. Therefore, this server is oftentimes replicated to multiple machines. Higher availability and much simpler cluster architecture can be achieved with the second distribution strategy called consistent hashing [27].

In this shared nothing architecture, there exists no single point of failure. In contrast to range based partitioning, keys are distributed by using hash functions. Since every server is responsible for a certain hash region, addresses of certain keys within the cluster can be calculated very fast.

Good hash functions distribute keys intuitively even wherefore an additional load balancer is not required. Consistent hashing also scores by dynamic cluster resizing. In contrast to other approaches, addition or removal of nodes only affects a small subset of all machines in the cluster. This simple architecture leads to performance penalties on range queries caused by high network load since neighboured keys are distributed randomly across the cluster.

All aforementioned key value stores and the document stores Riak and CouchDB are based on consistent hashing, whereas MongoDB documents are partitioned by the range of their ID. In contrast to key value and document stores, column family stores can be partitioned vertically, too. Columns of the same column family are stored on the same server in order to increase attribute range query performance. Cassandra datasets are partitioned horizontally by consistent hashing, whereas the BigTable clones HBase and Hypertable use range based partitioning. Since column family data models can be partitioned more efficiently, these databases are more suitable for huge datasets than document stores.

In contrast to key value based NoSQL stores, where datasets can easily be partitioned, splitting a graph is not straightforward at all. Graph information is not gained by simple key lookups but by analysing relationships between entities. On the one hand, nodes should be distributed on many servers evenly, on the other hand, heavily linked nodes should not be distributed over large distances, since traversals would cause huge performance penalty due to heavy network load. Therefore, one has to trade between these two limitations. Graph algorithms can help identifying hotspots of strongly connected nodes in the graph schema. These hotspots can be stored on one machine afterwards. Since graphs can rapidly mutate, graph partitioning is not possible without domain specific knowledge and many complex algorithms. Due to these problems, Sesame, Ne04j and Graph DB do not offer any partitioning opportunities. In contrast, FlockDB is designed for horizontal scalability. Since FlockDB does not support multi-hop graph traversal, a higher network load is no problem.

Since distributed systems increase system complexity massively, partitioning should be avoided if it is not absolutely necessary. Systems which do mostly struggle with high-read-request-rates can scale this workload more easily through replication.

ISSN: 2278 – 1323

VI. REPLICATION AND CONSISTENCY

Data replication is that the idea of getting knowledge, among a system, be geo-distributed, ideally through a non-interactive, reliable method. In ancient RDBMS databases, implementing any style of replication may be a struggle as a result of these systems weren't developed with horizontal scaling in mind. Instead, these systems are protected via a semi-manual method wherever live recovery wouldn't be a lot of a difficulty. Even with live recovery not being a lot of a difficulty, it downplays the complexness of this setup. Once managing today's globally distributed knowledge, the previous collocated replication ideas won't serve once enforced at geographic scale.

Today's infrastructure needs systems that natively support active and period replication, achieved through clear and easy configurations. The power to dictate wherever and the way your knowledge is replicated via simply tunable settings, besides providing users with simply understood ideas is what modern-day NoSQL databases attempt to supply.

To achieve high availability and durability, Dynamo replicates its data on multiple hosts. Each data item is replicated at N hosts, where N is a parameter configured "per-instance". Each key, k, is assigned to a coordinator node which is in charge of the replication of the data items that fall within its range.

Dynamo is designed to be an eventually consistent system. This means that update operations return before all replica nodes have received and applied the update. Subsequent read operations therefore may return different versions from different replica nodes. The update propagation time between replicas is limited in Amazon's platform if no errors are present; under certain failure scenarios however "updates may not arrive at all replicas for an extend period of time".

Such inconsistencies need to be taken into consideration by applications.

Systems that area unit eventually consistent so as to extend accessibility area unit Redis, CouchDB and Ne04j. Considering that CouchDB and Ne04j conjointly supply master-master replication, these systems area unit appropriate for offline support required e.g. in mobile applications. Voldemort, Riak, MongoDB, prophetess and FlockDB supply optimistic replication, wherefore they will be utilized in any context. Since Membase, HBase, Hypertable, herb and GraphDB don't use replication for load leveling, these stores supply full consistency. BigData is that the solely store, that supports full consistency and replication natively.

VII. CONCLUSION

"Use the proper tool for the job" is that the propagated ideology of the NoSQL community, as a result of each NoSQL information is specialised on sure use cases. Since there's no analysis obtainable that answers the question "which tool is that the right tool for the job?" blessings and downsides of those stores were compared during this paper. prons and corns of these stores were compared in this paper.

First of all, developers ought to assess their information so as to spot an acceptable information model to avoid gratuitous complexness because of transformation or mapping tasks. Queries that ought to be supported by the information ought to be thought of at an equivalent time, as a result of these necessities massively influence the planning of the information model. Since no common source language is out there, each store differs in its supported question feature set. Afterwards, developers ought to trade between high performance through partitioning and cargo balanced duplicate servers, high handiness supported by asynchronous replication and strict consistency. If partitioning is needed, the choice of various partition methods depends on the supported queries and cluster complexness. Beside these completely different necessities, conjointly sturdiness mechanism, community support and helpful options like versioning influence the information choice. In general, key price stores ought to be used for in no time and straightforward operations, document stores provide a versatile information model with nice question potentialities, column family stores area unit appropriate for terribly massive datasets that ought to be scaled at massive size, and graph databases ought to be employed in domains, wherever entities area unit as necessary because the relationships between them.

REFERENCES

[1] G. DeCandia, *et al.,* "Dynamo: amazon's highly available key-value store," in *SOSP '07 Proceedings of twenty-first ACM SIGOPS,* New York, USA, 2007, pp. 205-220.
[2] K. Orend, "Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer," Master Thesis, Technical University of Munich, Munich, 2010.
[3] R. Cattell, "Scalable SQL and NoSQL Data Stores," *ACM SIGMOD Record,* vol. 39, December 2010.
[4] C. Strauch, "NoSQL Databases" unpublished.
[5] M. Adam, "The NoSQL Ecosystem," in *The Architecture of Open Source Applications,* A. Brown and G. Wilson, Eds., lulu.com, 2011, pp. 185-204.
[6] "Project Voldemort." Internet: http://project-voldemort.com, [30.09.2011]
[7] "Redis." Internet: http://redis.io, [30.09.2011]
[8] "Membase." Internet: http://couchbase.orgjmembase, [30.09.2011]
[9] "CouchDB." Internet: http://couchdb.apache.org, [30.09.2011]
[10] "MongoDB." Internet: http://mongodb.org, [30.09.2011]
[11] "Riak." Internet: http://basho.com/Riak.html, [30.09.2011]

ISSN: 2278 – 1323