Partitioning Schemes used for Context Storage using Non Relational Databases. A Review

Kamaljeet kaur
Assistant Prof(CSE deptt)
Sri Guru Granth Sahib World University
Fatehgarh Sahib, Punjab(India)

Jaspreet kaur
Student of M.Tech(CSE)
Sri Guru Granth Sahib World University
Fatehgarh Sahib, Punjab(India)

Abstract:— NoSQL (Not only SQL) is a database used to store large amounts of data. NoSQL databases are distributed, non-relational, open source and are horizontally scalable (in linear way). NoSQL does not follow property of ACID as follow in SQL. Improving the performance of a database system is one of the key research issues now a day. As publications of context is rising day by day. So to handle increasing dataload, we use partitioning schemes. In this paper we discussed about partitioning schemes, existing work which comprises XCOA-XMPP context architecture which explains how the context is stored using non relational databases as we employs vertical partitioning scheme which gives better results as compared to horizontal partitioning to handle better dataload and to improve performance in all tradeoff using non relational databases as an approach. So we proposed an algorithm which comprises vertical partitioning scheme using couchdb as a non relational database which gives better results as compared to existing work in context management platform.

Keywords:-Non Relational Databases, Horizontal Partitioning, Vertical Partitioning, Partitioning Schemes.

I INTRODUCTION

A partition is a division of a logical database or its constituting elements into distinct independent parts. Database partitioning is normally done for manageability, performance or availability reasons. A popular and favourable application of partitioning is in a distributed database management system.[2] Each partition may be spread over multiple nodes, and users at the node can perform local transactions on the partition. This increases performance for sites that have regular transactions involving certain views of data, whilst maintaining availability and security. In distributed systems literature, partition tolerance refers to a system's overall ability to live up to its specification in the presence of network partitions. A bank could possibly do to facilitate a transfer if my record is on this side of the chasm and yours is on the other. In such cases, it is almost always better to build services that degrade gracefully under partitions. In this case, we would want failure detection and carry out those transfers where the accounts are both on the same side of the partition, while deferring transfers that cross the chasm. If we do perform this failure detection and defer operations that cannot be satisfied because data is not available, we lose some business, but at least we have a mostly working bank.

Horizontal partitioning[2] is a partitioning method that partitioning the table into no.of smaller tables on the basis of rows. It involves putting different rows into different tables. Perhaps customers with ZIP codes less than
50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest,[3] while a view with a union might be created over both of them to provide a complete view of all customers.

In traditional database management systems introduced the concept of row oriented database. Row oriented database is the database which stores data in rows. It has high performance for the OLTP i.e. online transaction processing. But there are many problems with row oriented database like data compression, difficult to handle complex read queries, creates very large I/O burdens and perform low operations, increase the no of actual disk reads to satisfy a query, storage of multiple indexes.

**Vertical partitioning**[3] involves creating tables with fewer columns and using additional tables to store the remaining columns. Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized. Different physical storage might be used to realize vertical partitioning. Non relational databases introduced the concept of column oriented databases. Column oriented database is the database which stores data in column. It overcomes most of the problems which are used in horizontal partitioning.

- High performance on aggregation queries (like COUNT, SUM, AVG, MIN, MAX).
- Highly efficient data compression and/or partitioning.
- True scalability and fast data loading for Big Data.
- Provides hard disk access and reduce disk space.
- Improved Bandwidth Utilization.

We can create subpartitions—that is, partitions of partitions and then subpartitions to combine all types of partitions: range partitions, list partitions, and hash partitions. For example, use hash partitions in combination with range partitions, creating hash partitions of the range partitions. For very large tables, this composite partitioning may be an effective way of separating the data into manageable and tunable divisions.[3]

### 1.1 FEATURES OF NOSQL

**(i) Dynamic schemas**

Relational databases require that schemas be defined before you can add data. For example, you might want to store data about your customers such as phone numbers, first and last name, address, city and state – a SQL database needs to know this in advance. This fits poorly with agile development approaches, because each time you complete new features, the schema of your database often needs to change. A few iterations into development to store customers favorite items in addition to their addresses and phone numbers, then add that column to the database, and then migrate the entire database to the new schema.[4]

**(ii) Auto-sharding, replication and integrated caching**

As relational databases are structured, they usually scale vertically. A single server has to host the entire database to ensure reliability and continuous availability of data. This gets expensive quickly, places limits on scale, and creates a relatively small number of failure points for database infrastructure. The solution is to scale horizontally, by adding servers instead of concentrating more capacity in a single server. Cloud computing makes this significantly easier, with providers such as Amazon Web Services providing virtually unlimited capacity on demand, and taking care of all the necessary database administration tasks. Developers no longer need to construct complex, expensive platforms to support their applications, and can concentrate on writing application code. In addition, a group of commodity servers can provide the same processing and storage capabilities as a single
high-end server. Sharding" a database across many server instances can be achieved with SQL database and on other complex arrangements for making hardware act as a single server. NoSQL databases, on the other hand, usually support auto-sharding, meaning that they natively and automatically spread data across an arbitrary number of servers, without requiring the application to even be aware of the composition of the server pool. Data and query load are automatically balanced across servers, and when a server goes down, it can be quickly and transparently replaced with no application disruption.[4]

Generally, NoSQL databases include the following designs:[1]

**Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

**Graph stores** are used to store information about networks, such as social connections. Graph stores include Neo4J and HyperGraphDB.

**Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name, or key together with its value. Examples of key-value stores are Riak and Voldemort. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.

**Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

### 1.2 COUCHDB

CouchDB is a “NoSQL” database, categorized in document stores. While this term is a rather generic characterization of a database, or data store, it does clearly define a break from traditional SQL-based databases. A CouchDB database lacks a schema, or rigid pre-defined data structures such as tables. Data stored in CouchDB is a JSON document. The structure of the data, or document, can change dynamically to accommodate evolving needs. It is written in Erlang.[6]

#### FEATURES OF COUCHDB

The main features of couchdb are given below:

- CouchDB stores data as "documents", as one or more field/value pairs expressed as JSON. [6]
- CouchDB provides ACID semantics. It does this by implementing a form of Multi-Version Concurrency Control.[7]
- The stored data is structured using views. CouchDB can index views and keep those indexes updated as documents are added, removed, or updated.[6]
- Couchdb also provides Map/Reduce functionality .[6]
- Couchdb has distributed architecture with replication .[7]
- CouchDB guarantees eventual consistency to be able to provide both availability and partition tolerance.
- CouchDB can replicate to devices (like smart phones) that can go offline and handle data sync for you when the device is back online. CouchDB also offers a built-in administration interface accessible via web called Futon .[7]

### II LITERATURE REVIEW

[1] Vatika Sharma et al, describes the comparison b/w sql and nosql databases, overview of NoSQL databases, with its background and characteristics. It also describes features of nosql. [2][3] describes the types of partitioning and their advantages.[5][6] D. W. Cornell et al, describes the vertical partitioning algorithms to handle
This paper proposed a vertical partitioning algorithm for improving the performance of database systems at design cycle. The algorithm uses the number of occurrences of an attribute in a set of queries rather than the FOQ accessing these attributes. [10] Dr. A. S. M. et al, research works proved in [9] describes the fragmentation solution based on empirical data about the type and frequency of the queries submitted to a centralized system. These solutions are not suitable at the initial stage of a database design for a distributed system. In this paper, fragmentation technique has been applied at the initial stage as well as in later stages of a distributed database system for partitioning the relations. Allocation of fragments is done simultaneously in our algorithm. Result shows that proposed technique can solve initial fragmentation problem of relational databases for distributed systems properly. [11] Nitin R. Chopde et al, this paper various techniques of load balancing algorithm are discussed, analyzed and compared. Also new load balancing algorithm which has new capabilities to provide optimum load balancing in system is discussed. [12] In this paper they proposed a new Context Management Architecture to solve the problems of context information in web based technologies. [13] Nuno Santos et al, proposes the usage of a NoSQL storage system for the purpose of context information storage and retrieval in an XMPP broker-based context platform such as XCoA, together with a full-text searching engine. The paper clearly demonstrates the advantages of NoSQL storage systems applied to the area of Context Management. The proposed algorithm comprises vertical partitioning scheme using couchdb as a non relational database which gives better results approximately in all tradeoffs as compared to existing work in context management platform.

III Existing Work And Their Solutions

XCOA- XMPP CONTEXT ARCHITECTURE

Fig-1 Xcoa- Xmpp Context Architecture[12]

[11] In this architecture, context information is first collected by Context Agents (CA) such as mobile terminals, social networks, or wireless sensor networks, and aggregated by CPs dealing with specific context scopes such as Location, or Social Preferences. Each CP is responsible to publish the aggregated context information on a CB Publish/Subscribe service which stores this information (History), and sends it to CCs as requested using the PubSub model. All the context information between the CPs, the CB and the CCs is exchanged in XMPP Publish-Subscribe messages. CCs can subscribe to specific Context data, such as Location, and receive notifications when the Providers publish information in broker.

IV. CONTEXT STORAGE USING NOSQL

Although in a Publish-Subscribe system it is not necessary to store every published item, in a Context Management Platform it becomes advantageous to do so. Storing every piece of context information provides a comprehensive history of a user’s context data, which allows for very powerful features such as context-aware advertising, actuation and environment adaptation according to the user’s preferences. With the need to store every user’s full context history comes the problem of handling very large quantities of data.[12] This is a very active
research problem, with several proposed solutions such as the so-called NoSQL storage systems. The information to be published, as previously above is in XML format. This information can have any structure, and the Platform should be able to efficiently handle context information with any XML structure.

In order to study the feasibility of storing Context Information in a PubSub XMPP Component, a prototype was implemented. It was decided to use an external XMPP PubSub component that integrated with an existing XMPP server and offered XMPP PubSub capabilities. For the selection of a NoSQL solution, both performance advantages, scalability, reliability and indexing / searching capabilities were kept in mind. CouchDB is a document-oriented database with many important features such as replication capabilities, distributability, allows for horizontal scalability, as well as integration with external document indexing / searching. It is open-source, widely deployed. Other NoSQL solutions such as document-oriented MongoDB were also kept in mind. MongoDB, however, does not offer data durability guarantees in single-node deployments. Wide-column store Cassandra and HBase offer data durability, but their complex data model was deemed incompatible with the XML nature of context information, as it would fit better in a document-oriented database. The structure of the documents stored in CouchDB, representing Items, is as follows:

```
{
  "doc_type": "item",
  "item_id": <item_id>,
  "node": <identifier of the node>,
  "publisher": <JabberID of the publishing entity>,
  "date": <Date and time of item publishing>,
  "data": <Item XML String>
}
```

Adjacency List is simpler, where each node has a connection to its parent node, queries use recursion, which can adjust impossible in certain SQL solutions. However, as the database used was PostgreSQL 8.4, which already supports recursive queries, the Adjacency List model was chosen. It should be noted that CouchDB does not have native support for full-text search queries, that’s why Apache Lucene is used in integration with CouchDB. There is currently no supported integration between Lucene and PostgreSQL, so the comparison was made between CouchDB with Lucene and PostgreSQL, Openfire and thus XCoA platform’s default database. Results for Item retrieval and Item search are shown in Graph 1 and 2.

Graph 1- PostgreSQL vs CouchDB: Item retrieval

Graph 1 shows the performance degradation of Item retrieval when the database increases in the number of Items. Retrieving an Item in PostgreSQL on a 1,000,000-Item database takes on average over 1 second, while in CouchDB takes less than 80 ms. PostgreSQL shows a 98% performance degradation between datasets of 10K and 1M Items, against the 24% degradation observed in CouchDB. [12]
Graph 2- PostgreSQL vs CouchDB / Lucene: Item searching

Graph 2 shows the performance degradation of Item searching. With the increase of database Items, the performance advantages of the CouchDB / Lucene system become apparent. PostgreSQL shows 98% performance degradation between 10K (about 2ms) and 1M Items (about 181ms), against CouchDB / Lucene’s 66% (4ms for 10K, 12ms for 1M Items).[12]

V. OUR PROPOSED ALGORITHM

In our proposal we are concentrating upon the partitioning schemes as we are employing the vertical partitioning which gives us the better results as compared to horizontal partitioning approximately in all tradeoffs to handle better load and to get better performance as compared to previous work in context management platform using couchdb as a nosql database. Our proposed algorithm for the nosql database partitioning scheme is as follow:

\[
\begin{align*}
\text{I} & \quad \text{is counter} \\
\text{T} & \quad \text{is time} \\
\text{T}_i & \quad \text{is time at } i\text{th second} \\
\text{W} & \quad \text{is weight of database} \\
\text{P} & \quad \text{is partitioned data} \\
\end{align*}
\]

R is path function i.e. path through which the data is communicating
D is subset of query
\(\text{C}_{\text{comm}}\) is cost of communication
\(\text{C}_{\text{route}}\) is cost of path finding and attaining
\(i = 0;\)
\(t_i = 0;\)
Total time taken for m/c we’ll define a cost function
Call Greedy Algorithm
\[
\text{T}_i = \text{cost}(w,|D|)
\]
For all\(\text{cost}(r(I,w)) = c_{\text{route}}\)
Total communication cost for databases
\[
\text{C}_{\text{com}} = \text{C}_{\text{request}} + \text{C}_{\text{response}}
\]
Communication Time \(\text{T}_{\text{comm}}(w,i,N) = \{^0\text{if } r(N,w)=i \text{\ it } \text{C}_{\text{comm}} \text{ \ if } r(N,w)\neq i)\}
\]
For N servers
\[
\text{T}_{\text{comm}} = \text{C}_{\text{comm}} - \text{C}_{\text{comm}}/N
\]
Total time for N servers
\[
\text{T}_N = \text{C}_{\text{route}} + \text{C}_{\text{comm}} - \text{C}_{\text{comm}}/N+\text{cost}(w,|p|)
\]
Speed up is
\[
\text{S}(N) = \text{T}_1/\text{T}_N
\]

VI CONCLUSION AND FUTURE WORK

In this paper, it has been concluded that horizontal partitioning increases data load in context management platform as no of context publications rising day by day which reduces the performance in distributed databases. So we proposed an algorithm using vertical partitioning scheme using non relational databases as an approach which gives better results as compared to horizontal partitioning. Future work, we can implement distributed database on all nosql databases.

VII REFERENCES


