Cassandra as a Big data Modeling Methodology for Distributed Database System

[1]Dr. Kalpesh U. Gundigara ,[2] Ms. Vibha H. Mehta
[1]Academic Head, [2]Assistant .Professor..
Shree Swaminarayan College of Computer Science,Bhavnagar,India

***Abstract*— Databases are defined as collections of related data. Databases are important components of Information Systems and are concurrent to the existence of computer technology. Selection of Appropriate Database is necessary for storing and retrieving database in meaning manner.**

In Recent Days There is requirement for higher volume of data with high operation rates ,good development and big database.Because Of Higher volume Data many Companies have adopted different types of non relational database like Mongodb, Cassendra, Hbase/Hadoop,CouchDB etc.These Database is also known as NoSQL database. This paper attempts to use NoSQL database to replace the relational database. This Paper will focus on Cassandra database.It mainly focuses on one of the new technology of NoSQL database i.e. Cassandra , and makes a comparison study with SQL.

This paper presents an architectural overview of Cassandra and discusses how its design is founded in fundamental principles of distributed systems. Merits of this design and practical use are given throughout this discussion. Additionally, a project is supplied demonstrating how Cassandra can be leveraged to store and query high-volume, consumer-oriented airline flight data.

***Index Terms*— Apache Cassandra, data modeling, CQL,Keyspace,Supercolumn,cluster.**

I. INTRODUCTION OF NO SQL DATABASE

A database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases is known as NoSQL Database(Sometimes called as Not Only SQL). These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.
The primary objective of a NoSQL database is to have:

- simplicity of design,
- horizontal scaling, and
- finer control over availability.

NoSql databases use different data structures compared to relational databases. It makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it must solve.

NOSQL systems are mostly non-relational database systems that are distributed and are understood as Not Only SQL. NOSQL databases are known to provide easier scalability, storage flexibility, and greater data manipulation and performance improvement. There are various types of NOSQL database systems among which Key-value stores; Wide-column stores,Graph databases and Document stores are identified most commonly . MongoDB, Cassandra,DynamoDB and CouchDB, Neo4j, Riak are the more popular NOSQL databases used commonly in today's environment.

II. NOSQL DATABASE TYPES

- Document Oriented Databases

Document oriented databases treat a document as a whole and avoid splitting a document in its constituent name/value pairs. At a collection level, this allows for putting together a diverse set of documents into a single collection. Document databases allow indexing of documents on the basis of not only its primary identifier but also its properties. Document orientated databases work in a similar way to column-orientated database but differ in providing deeper nesting and complex structures that is document within document and so on . Some of the popular document orientated databases are MongoDB, CouchDB and Couchbase.

- Graph Based Databases

Graph databases are totally different from the three previous NOSQL database types. Graph orientated databases are databases that rely on explicit graph structure where nodes and edges connect to each other through relations in a tree like structure. Each node knows it adjacent nodes and there is an index for searches. Nodes store data about each entity in the database, relationships describe relationship between the nodes and property is just the opposite node of the relationship . Some of the popular graph orientated databases are OrientDB, MarkLogic and Neo4j.

- Column Based Databases

In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows.

A record could be in one column or multiple columns and also columns can be nested inside other columns known as super columns. Columns can be grouped together in one column family or multiple column families. Data retrieval happens by using primary key per column family.

Contrary to relational databases where a particular data is stored in rows of a table, column orientated databases store data in column(s). On the surface, column orientated databases might appear to look like a relational database, however they are different as these don't have any prestructured table to work with the data.
This property of column orientated database makes retrieving of large amounts of a particular attribute faster. Column orientated databases are in essence a two-dimensional array where each key, a record/row has one or more key/value pairs attached to it . This allows for the management for large unstructured data.
Cassandra, Google's Big Table and HBase are some of the more popular wide column database systems

- Key Value Databases

The key of a key/value pair is a unique value in the set and can be easily looked up to access the data. Key/value pairs are of varied types: some keep the data in memory and some provide the capability to persist the data to disk. A simple, yet powerful, key/value store is Oracle's Berkeley DB. Some of the popular Key-value databases are DynamoDB, Riak and Redis.

III. POPULAR NOSQL DATABASE

Key Value	Redis	Riak
Graph Database	Neo4j-the graph database	Hyper GraphDB
Document Oriented	Mongo DB	CouchDB
Column Family	Cassandra	HBASE

IV. HISTORY OF CASSANDRA DATABASE

Facebook is the largest social networking platform that serves hundreds of millions users at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Facebook's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable.

In Facebook , Inbox Search is a feature that enables users to search through their Facebook Inbox. At Facebook this meant the system was required to handle a very high write throughput, billions of writes per day, and also scale with the number of users. Since users are served from data centers that are geographically distributed, being able to replicate data across data centers was key to keep search latencies down. Inbox Search was launched in June of 2008 for around 100 million users and today they are at over 250 million users.

There are always a small but signicant number of server and network components that are failing at any given time. As such, the software systems need to be constructed in a manner that treats failures as the norm rather than the exception.

Cassandra originated at Facebook in 2007 to solve that company's inbox search problem, in which they had to deal with large volumes of data in a way that was difficult to scale with traditional methods. Specifically, the team had requirements to handle huge volumes of data in the form of message copies, reverse indices of messages, and many random reads and many simultaneous random writes.To meet the reliability and scalability needs described above Facebook has developed Cassandra Database.Cassandra uses a synthesis of well known techniques to achieve scalability and availability. Cassandra was designed to fufill the storage needs of the Inbox Search problem. and Cassandra has kept up the promise so far. Cassandra is now deployed as the backend storage system for multiple services within Facebook.

V. INTRODUCTION OF CASSANDRA DATABASE

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure.

Cassandra database use cases include product catalogs and playlists, sensor data and Internet of Things, messaging and social networking, recommendation, personalization,fraud detection, and numerous other applications that deal with time series data.
Cassandra Database has been adopted in big data applications because of its scalable and fault-tolerant peer-to-peer architecture , versatile and flexible data model that evolved from the BigTable data model , declarative and user-friendly Cassandra Query Language (CQL), and very efficient write and read access paths that enable critical big data applications to stay always on, scale to millions of transactions per second, and handle node and even entire data center failures with ease.

VI. FEATURES OF CASSANDRA DATABASE

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- **Elastic scalability**: Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- **Always on architecture**: Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- **Fast linear-scale performance**: Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- **Flexible data storage**: Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- **Easy data distribution**: Cassandra provides the flexibility to distribute data where you need by replicating data across multiple datacenters.
- **Transaction support**: Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- **Fast writes**: Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency

VII. DATA MODEL OF CASSANDRA DATABASE

Traditional data modeling methodology, which is used in relational databases, defines well-established steps shaped by decades of database research. A database designer typically follows the database schema design work- flow depicted in Figure to define a conceptual data model, map it to a relational data model, normalize relations, and apply various optimizations to produce an efficient database schema with tables and indexes. In this process, the primary focus is placed on understanding and organizing data into relations, minimizing data redundancy and avoiding data duplication.
Queries play a secondary role in schema design. Query analysis is frequently omitted at the early design stage because of the expressivity of the Structured Query Language (SQL) that readily supports relational joins, nested queries, data aggregation, and numerous other features that help to retrieve a desired subset of stored data. As a result, traditional data modeling is a purely data-driven process, where data access patterns are only taken into account to create additional indexes and occasional materialized views to optimize the most frequently executed queries.

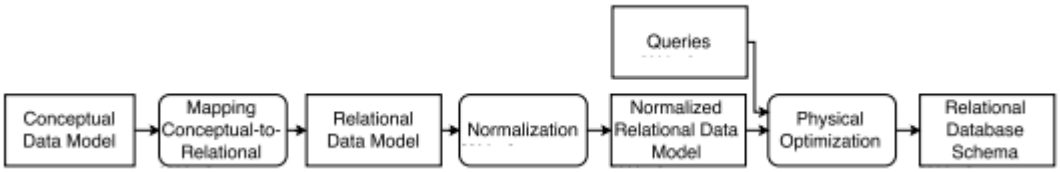

Fig.I Relational Data Model

Fig .II Cassandra Data Modeling

In contrast, known principles used in traditional database design cannot be directly applied to data modeling in Cassandra. Data model of Cassandra is significantly different from what we normally see in an RDBMS.

Cassandra is a distributed key-value store. Unlike SQL queries which allow the client to express arbitrarily complex constraints and joining criteria, Cassandra only allows data to be queried by its key. Additionally, indexes on non-key columns are not allowed and Cassandra does not include a join engine– the application must implement any necessary piecing together of related rows of data. For this reason the Cassandra data modeler must choose keys that can be derived or discovered easily and ensure maintenance of referential integrity. Cassandra has adopted abstractions that closely align with the design of Bigtable.
The primary units of information in Cassandra parlance are outlined as follows.

Cluster

Cluster Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are:

- Replication factor: It is the number of machines in the cluster that will receive copies of the same data.
- Replica placement strategy: It is nothing but the strategy to place replicas in the ring. We have strategies such as simple strategy (rackaware strategy), old network topology strategy (rack-aware strategy), and network topology strategy (datacenter-shared strategy).
- Column families: Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The following illustration shows a schematic view of a Keyspace.

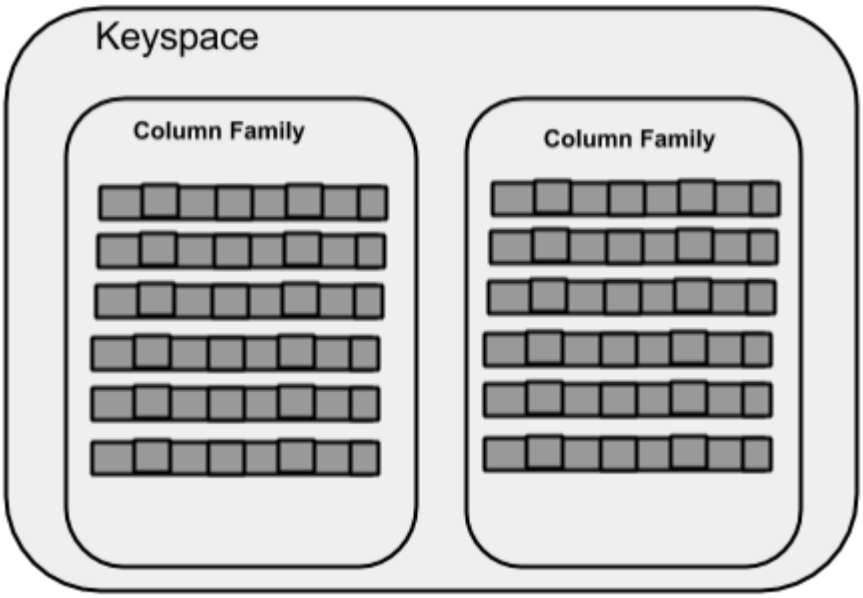

Fig.III Keyspace

Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

Relational Table Cassandra Column Family	Relational Table Cassandra Column Family
A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value	In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time.
Relational tables define only columns and the user fills in the table with values.	In Cassandra, a table contains columns, or can be defined as a super column family.

Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

Column		
name : byte[]	value : byte[]	clock : clock[]

Fig.IV Column

SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns. Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same column family, and a super column can be helpful here. Given below is the structure of a super column.

Super Column	
name : byte[]	cols : map<byte[], column>

Fig.V SuperColumn

VIII. SYSTEM ARCHITECTURE

The Cassandra architecture is very sophisticated and relies on the use of several different theoretical constructs. The design goal of Cassandra is to handle big data workloads across multiple nodes without any single point of failure. Cassandra has peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- All the nodes in a cluster play the same role. Each node is independent and at the same time interconnected to other nodes.
- Each node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- When a node goes down, read/write requests can be served from other nodes in the network.

○ Components of Cassandra

The key components of Cassandra are as follows:

- Node: It is the place where data is stored.
- Data center: It is a collection of related nodes.
- Cluster: A cluster is a component that contains one or more data centers.
- Commit log: The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- Mem-table: A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- SSTable: It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- Bloom filter: These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query

○ Data Replication in Cassandra

To achieve high scalability and durability of a Cassandra cluster, data gets replicated to a number of nodes which can be defined as a replication factor per Cassandra instance. Replication is managed by a coordinator node for the particular key being modified; the coordinator node for any key is the first node on the consistent hash ring that is visited when walking from the key's position on the ring in clockwise direction.

The following figure shows a schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure

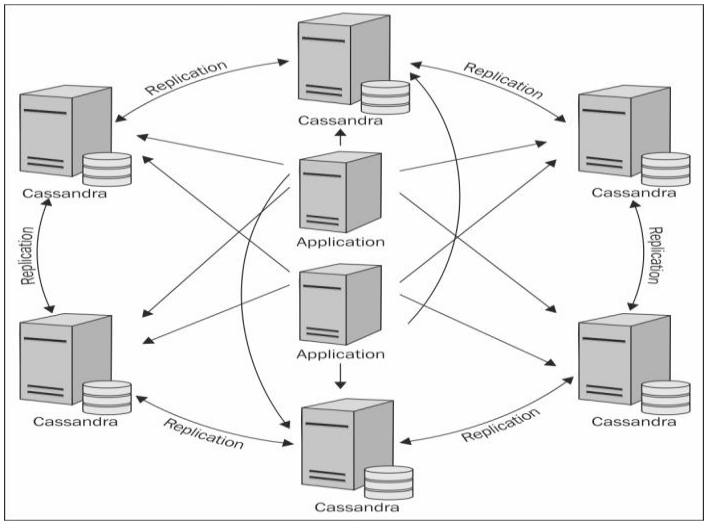

Fig.VI Data Replication in Cassandra Database

Cassandra uses the Gossip Protocol in the background to allow the nodes to communicate with each other and detect any faulty nodes in the cluster.

System Keyspace

Cassandra has an internal keyspace called system that it uses to store metadata about the cluster to aid in operations. In Microsoft SQL Server, two meta-databases are maintained: master and tempdb. The master is used to keep information about disk space, usage, system settings, and general server installation notes; the tempdb is used as a workspace to store intermediate results and perform general tasks. The Oracle database always has a tablespace called SYSTEM, used for similar purposes. The Cassandra system keyspace is used much like these. user cannot modify the system keyspace

The metadata includes:

• The node's token
• The cluster name
• Keyspace and schema definitions to support dynamic loading
• Migration data
• Whether or not the node is bootstrapped

Peer-to-Peer

Cassandra has a peer-to-peer distribution model, such that any given node is structurally identical to any other node—that is, there is no "master" node that acts differently than a "slave" node. The aim of Cassandra's design is overall system availability and ease of scaling. The peer-to-peer design can improve general database availability, because while taking any given Cassandra node offline may have a potential impact on overall throughput, it is a graceful degradation that does not interrupt service. Assuming that you are using a reasonable replication strategy, the data on a failed node will still be available for reads and writes.

This design also makes it easier to scale Cassandra by adding new nodes. Because the behavior of each node is identical, in order to add a new server, you simply need to add it to the cluster. The new node will not immediately accept requests so that it has time to learn the topology of the ring and accept data that it may also be responsible for. After it does this, it can join the ring as a full member and begin accepting requests.

Gossip and Failure Detection

To support decentralization and partition tolerance, Cassandra uses a gossip protocol for intra-ring communication so that each node can have state information about other nodes. The gossiper runs every second on a timer. Hinted handoff is triggered by gossip, when a node notices that a node it has hints for has just come back online. Anti-entropy, on the other hand, is a manual process; it is not triggered by gossip.

Gossip protocols (sometimes called "epidemic protocols") generally assume a faulty network, are commonly employed in very large, decentralized network systems, and are often used as an automatic mechanism for replication in distributed databases. They take their name from the concept of human gossip, a form of communication in which peers can choose with whom they want to exchange information.

Here is how the gossiper works:

1. Periodically (according to the settings in its TimerTask), the G=gossiper will choose a random node in the ring and initialize a gossip session with it. Each round of gossip requires three messages.
2. The gossip initiator sends its chosen friend a Gossip Digest SynMessage.
3. When the friend receives this message, it returns a GossipDigestAckMessage.
4. When the initiator receives the ack message from the friend, it sends the friend a GossipDigestAck2Message to complete the round of gossip.

IX. DATA MODELS OF CASSANDRA AND RDBMS

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. (ROW x COLUMN)	In Cassandra, a table is a list of "nested key-value pairs". (ROW x COLUMN key x COLUMN value)
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

X. CASSANDARA QUERY LANGUAGE

Keyspace: A keyspace is an object that holds the column families, user defined types. In Cassandra, Keyspace is similar to RDBMS Database. Keyspace holds column families, indexes, user defined types, data center awareness, strategy used in keyspace, replication factor, etc.

Command "**Create Keyspace**" is used to create keyspace in Cassandra.

Syntax

```
Create keyspace KeyspaceName with replicaton={'class':strategy name, 'replication_factor': No of replications on different nodes}
```

Example

```
Create keyspace University with replication = {'class':'simplestrategy', 'replication_factor' : 3};
```

Alter Keyspace

Command "Alter Keyspace" alters the replication factor, strategy name and durable writes properties in created keyspace in Cassandra.

Syntax

```
Alter keyspace “keyspace Name” with replication = {'class': ‘Strategy name’, 'replication_factor' : ‘No.Of replicas’};
```

Example

```
Alter keyspace University with replication = {'class':'NetworkTopologyStrategy', 'replication_factor' : 3};
```

Drop Keyspace

Command 'Drop Keyspace' drops keyspace including all the data, column families, user defined types and indexes from Cassandra. Before dropping the keyspace, Cassandra takes a snapshot of the keyspace. If keyspace does not exist in the Cassandra, Cassandra will return an error unless IF EXISTS is used.

Syntax

```
Drop keyspace KeyspaceName;
```

Example

```
DROP keyspace University;
```

Cassandra Create table

Column family in Cassandra is similar to RDBMS table. Column family is used to store data.Command 'Create Table' is used to create column family in Cassandra.

Syntax

```
Create table KeyspaceName.TableName
(
ColumnName DataType,
ColumnName DataType,
ColumnName DataType
.
.
.
Primary key(ColumnName)
) with PropertyName=PropertyValue;
```

Example

```
CREATE TABLE University.Student(
Rollno int,
Name text,
dept  text,
Primary Key(Rollno)
);
```

Cassandra Alter table

Command 'Alter Table' is used to drop column, add a new column, alter column name, alter column type and change the property of the table.

Syntax

```
Alter table KeyspaceName.TableName  + Alter ColumnName TYPE ColumnDataype | Add ColumnName ColumnDataType |Drop ColumnName |Rename ColumnName To NewColumnName | With propertyName=PropertyValue
```

Example

```
ALTER TABLE University.Student  ADD semester int;
```

Drop Table

Command 'Drop table' drops specified table including all the data from the keyspace. Before dropping the table, Cassandra takes a snapshot of the data not the schema as a backup.

Syntax

```
Drop Table KeyspaceName.TableName
```

Example

```
Drop Table University.Student;
```

Insert Data

Command 'Insert into' writes data in Cassandra columns in row form. It will store only those columns that are given by the user. You have to necessarily specify just the primary key column.
It will not take any space for not given values. No results are returned after insertion

Syntax

```
Insert into KeyspaceName.TableName(ColumnName1, ColumnName2, ColumnName3 . . . .)
values (Column1Value, Column2Value, Column3Value . . . .);
```

Example

```
Insert into University.Student (Rollno,Name,dept,semester) VALUES(2,'Michel','CS',2);
```

Here is the snapshot of the executed command 'Insert into' that will insert one record in Cassandra table 'Student'.

Cassandra Read Data

In Cassandra, data retrieval is a sensitive issue. The column is filtered in Cassandra by creating an index on non-primary key columns.

Syntax

```
Select from <Keyspace>.<tablename>;
```

Example

```
Select * from University.Student;
```

XI. USERS OF CASSANDRA DATABASE

List of companies using Cassandra is growing. These companies include:

- Twitter is using Cassandra for analytics. Twitter's primary Cassandra engineer, Ryan King, explained that Twitter had decided against using Cassandra as its primary store for tweets, as originally planned, but would instead use it in production for several different things: for real-time analytics, for geolocation and places of interest data, and for data mining over the entire user store.
- Mahalo uses it for its primary near-time data store.
- Facebook still uses it for inbox search, though they are using a proprietary fork.
- Digg uses it for its primary near-time data store.
- Rackspace uses it for its cloud service, monitoring, and logging.
- Reddit uses it as a persistent cache.
- Cloudkick uses it for monitoring statistics and analytics.
- Ooyala uses it to store and serve near real-time video analytics data.
- SimpleGeo uses it as the main data store for its real-time location infrastructure.
- Onespot uses it for a subset of its main data store

XII. CONCLUSION

Relational Database Management system won't go away for big databases application because have a drawback to our applications like time consuming and execution speed and scaling of queries, they are define compulsory. But the storage requirement for the new generation of applications are huge different from heritage applications. We can choose NoSQL(Cassandra Database) instead of SQL because of two factors, ease of use and timing performance.

So This paper has provided an introduction to Cassandra and fundamental of Cassandra Database systems principles on which it is based. We have Discussed Cassandra database system that provides scalable, high performance, and wide applicability. We have empirically demonstrated that Cassandra can support a very high update throughput while delivering low latency.

REFERENCES

- http://cassandra.apache.org
- http://www.tutorialspoint.com/cassandra/
- Dietrich Featherston, Cassandra: Principles and Application, 2010
- Lakshman, P. Malik, Cassandra - A Decentralized Structured Storage System, Cornell, 2009
- Companies that use Cassandra, http://planetcassandra.org/companies/.
- Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," Operating Sys. Review, vol. 44, no. 2, pp. 35–40, 2010