CALIFORNIA STATE UNIVERSITY SACRAMENTO

Spring 2020 CSC 244 Project Report

ON

<u>ClustrixDB</u>

Submitted By:

Anser Parvez Nadvi

Brunda Suresh

Department of Computer Science

California State University Sacramento

ABSTRACT

Relational Database has been the prime data management standard all around the world for several years. It has been the legacy vendor with mediocre performance in the new world. The need for faster, more scalable and flexible data management system raised in the recent years, that led to the development of NoSQL. However, NoSQL compromised on SQL and ACID to obtain Scalability. Hence a new Data Management System called NewSQL was brought into light which is ACID compliant, SQL based, Scalable, Distributed and highly available RDBMS system. They focus on making the Traditional SQL (RDBMS) perform better and scale better than NoSQL.

Few of the NewSQL databases are ClusterixDB, NuoDB and VoltDB. They are capable of automated recovery on failure and are suitable for Billions of transactions on Billions of rows of data. NuoDB is an elastically scalable, Java based, Multi-tenant, multilevel concurrency control database with Tiered Architecture. An in-memory, horizontally scalable NewSQL database called VoltDB avoids the overhead of traditional database by keeping all memory and hence logging and buffer management are not required. NewSQL databases can hence be declared as an Extension of SQL with internal architecture changes.

Clustrix is a scale-out database system which is self-managing with built-in instrumentation. It can be scaled by adding new nodes to clusters and is the first cluster-based database. They are particularly used in E-Commerce, Consumer Web and Advertising Analytics. The following report will hence describe in detail about the ClustrixDB and compare it with NuoDB and VoltDB.

TABLE OF CONTENTS

INTRODUCTION

Database can be described as a systematic collection of information or data that can be accessed, manipulated and analysed. It can be used in simplest of ways, for example, in making an appointment at a hospital or in asking a store employee to check for the availability of a particular item. The concept of database was first implemented in the 1960's by Charles Bachmen in the form of Integrated Data Store (IDS). The databases have evolved since with many different functionalities and usages. Most of the databases used since the invention of databases can be classified into three different classes:

1.1 Old SQL

The traditional databases such as Oracle, SyBase, IBM, MySQL, Microsoft SQL server and others which were invented during 1980's and 1990's are called Old SQL databases. They used Structured Query Language (SQL) to perform transactions and are compatible with several different tools. A typical Old SQL database could perform about 30 transactions per second and efforts were made to improve to 1000 transactions per second.

1.1.1 ACID Properties

All of the OldSQL databases have four important properties usually called as ACID property, which are Atomicity, Consistency, Isolation, and Durability.

1.1.2 Scaling

RDBMS initially start with a single relational database server and then it is scaled vertically by adding more memory and CPUs. Other method of scaling data in OldSQL is sharding which is the process of partitioning the data manually and spread it across multiple servers. However, filling a shard is disruptive and benefits of RDBMS are lost since performing transactions across multiple servers created due to sharding is difficult.

Overtime, a lot of data generation resulting in high volume and high data velocity were not handled well by the traditional databases. They were not able to scale well since they were disk-based systems.

1.2 NoSQL

Scaling of Relational databases results in inconsistency which led to compromise on consistency and ACID properties of relational databases and to prioritize scaling. This compromise resulted in the implementation of NoSQL by sharding in distributed ways. As SQL gets compiled internally in the DBMS, it is not the reason for performance or scalability issues. NoSQL did not require schema, because of which using SQL might be difficult. However it helps to get started with transactions, with improved performance and fault tolerance. Two languages, similar to SQL, CQL by Cassandra and UnQL by mongoDB are being used by NoSQL.

NoSQL are good for non-transactional systems such as logging, DNS queries, single record transactions and others. It is not suitable for OLTP processing such as gaming where about 100 million people do transactions. NoSQL has succeeded with providing good availability and scalability even for large datasets.

NoSQL also implement Sharding and replication of data at high rates. Data is replicated amongst the Master-Slave architecture where if the master fails the replicas are available with the slaves. This provides better fault tolerance during sharding. But the problems with sharding are:

- Accessing and processing data from two different servers can be difficult, so transactions do not work well. Since NoSQL works with sharding it works well with Non-transaction systems, and not OLTP.
- Dynamic Sharding is not possible since only a fixed number of shards can exist.

1.3 NewSQL

OldSQL had lower performance rates because they spent most of their time on unnecessary overhead due to which they had to get rid of all overhead resources and also needed better legacy codes. NewSQL, on the other hand, can be described as the advanced version of OldSQL with good performance, availability and horizontal scalability. NewSQL also preserves SQL and ACID properties, unlike NoSQL.

NewSQL is used in many scenarios which require high throughput and low latency. They are hybrids which can perform both transactional and analytical processing and provide in-depth analytics.

NewSQL supports the CAP theorem which states that at least 2 of the following 3 conditions should hold good:

- They have good consistency where all the clients will have the same view of the data
- They support partition tolerance where a particular amount of failure below a threshold can be accepted
- All data is available to the clients at all times for read and write within some maximum latency

1.3.1 <u>NewSQL Architecture Principles</u>

1. Minimize Locking

   This helps improve performance and overcome one of the OldSQL overheads by using timestamp during process transactions with no locking. This can be done by preventing row-level record-level locking by using other methods and implementing Multilevel or Optimistic Concurrency control
2. Reliance on memory

   The Architecture must be able to take advantage of the available memory and overcome the buffer pool overhead seen in OldSQL. An effort is made to keep entire database in memory and perform replication on them. Another option is where we use the memory moderately and use persistence for durability.
3. Use an architecture that avoids latching for shared data structures in a multithreading system. This can be done by using some innovative methods of B-tree or by using Single-Threading.
4. Implement automatic Built-in replication and failover.

1.3.2 <u>Scaling</u>

Vertical Scaling is a method of increasing the size of resources. For example, increasing the size of the disks or memory used for processing data within one single node/CPU. Vertical Scaling however has a limit since there exists a maximum possible size upto which a memory disk capacity can be scaled. Horizontal scaling is where instead of increasing the size of the resources within a node we just increase the number of nodes as and when needed. This is resilient since a single point failure does not result in an overall failure; however, Load balancing becomes important. It scales well as and when data increases.

NuoDB and VoltDB

2.1 NuoDB

NuoDB is an elastically scalable 100% SQL ACID compliant New SQL database. They are elastically scalable since they can scale even across data centers. They have heterogeneous database which is not only in memory but also on disk. It is available instantly, on demand and with little to no latency. It is architected by Jim Starkey. It is a java-based database that runs completely on JVM.

2.1.1 Architecture

NuoDB is a multi-tenant database with heavy use of memory. All the important data is kept in memory while the other data are placed in persistent storage. They perform partial on-demand replication across tiers or data centers. They have a tiered architecture with multi version concurrency control across the tiers. The three important tiers are:

- Transactional tier that manages the global transactions. It acts as a transaction engine that parses, compiles, optimizes, organizes and executes SQL queries. This tier has a caching logic built into it and also consists of a map of all the transaction tiers across the globe and helps find data from any of these tiers or engines irrespective of the global location of the machine. Higher the number of transaction tiers higher the throughput.
- Storage tier that helps manage the durability of data. Although data is stored on SAN across raids this storage manager which is heterogeneous can store data on HDFS, Amazon S3, Google compute engine and more. A single database can have multiple storage managers which makes the database more resistant to failure.
- Management tier that helps the client work on the transactional tier. It acts as Agents and Brokers that keep track of all the daemons that are running. They restart a daemon automatically if and when it goes down. They communicate with the storage and transaction tiers and keep them connected.

2.1.2 <u>Admin Benefits</u>

It is a new tier added to NuoDB. NuoDB can now run in containers such as Docker, OpenShift and others. It also helps attain better reliability, failsafe restart mechanism resulting in better resiliency. This resulted in the elimination of the snapshot feature of older NuoDB version since it has automatic start feature. Load balancing is made easy too.

2.1.3 <u>Recovery from Failure</u>

NuoDb is considered to have high availability and database resilience since they have the ability to remain running despite failure and outages while keeping the workloads running. If any of the transaction tiers fail in any of the machines NuoDB automatically connects to one of the other transaction tiers which might reduce the throughput until a replacement transaction tier is added to the machine. They also provide continuous availability since they move the data from data center to the cloud during datacentre outages which will not disrupt the application service.

- Scalability:

  A database needs a minimum of one transaction and one storage tier in a single machine. They can be split into two different machines with transactional tier in one and storage tier in another machine. They can further be scaled as follows:

  - Scaling transaction tier by adding more transactional tiers to the machine with transactional tiers.
  - Horizontal scaling by adding new machines with new transactional tier so if one machine crashes the other machine can carry on transactions.
  - Multiple storage managers in the storage manager machine to help maintain durability.
  - Scaling storage managers by adding new storage managers on new machines in the same database
- Multi tenancy: One machine with multiple schemas and databases within them.
- Multi Version concurrency control: Update and Delete queries creates new rows instead of replacing old ones

2.2 VoltDB

VoltDB is a database that was architected by Michael Stonebraker to be an in-memory fast RDBMS which is horizontally scalable and ACID compliant. It was built upon an open source database called Shore by modifying the code. It is available in community and commercial editions for use.

2.2.1 Architecture

VoltDB architecture avoids the traditional overhead by keeping all data in memory with partition across multiple machines using K-Safety for fault tolerance. They do not implement buffer management due to which the throughput can be maximum.

In VoltDB the data is partitioned and can operate autonomously because of which locking and latching are not required. This method also helps improve performance due to high throughput and achieve horizontal scaling. For example, a machine with 16 cores can have one core for OS and another for users to login with while the remaining 14 cores are daemons each of which are considered to be a partition. Each of the partitions operate individually in a single threaded method. Since each partition works on its own data, locking and latching are prevented.

Each of the VoltDB partitions have two parts which are the table data and the execution engine. The transactions which are queued in the execution engine are executed sequentially.

The ACID transactions in VoltDB can be done in two methods of partition:

a) Single-Partition (Local Transaction)

   Transactions are done only within single partition where the majority of the transactional workload is on that single partition. This is done in a single threaded way. For example, consider a machine with 4 partitions, where one partition does the OS related work and the other 3 partitions have two tables a and b spread across them. On performing a query, this method returns rows from only one of the three partitions even though the rows for which the query condition satisfies also exists in other partitions. Hence, if each client performs one transaction on each of the partitions, 3 clients can do 3 transactions at one point of time.

b) Multi-Partition (Global Transaction)

Transactions are done across multiple partitions. In the example mentioned above the query result returns the rows from all three partitions. One of the partition acts as global co-ordinator which co-ordinates data across different partitions put the query results from each partition together and returns the results of the query as one. Although the data is taken from all partitions each of them are still single threaded transaction.

2.2.2 Security

Each of the VoltDb users are given roles in the database and will have to validate themselves by a security process. Every time a user runs a stored procedure while using the database the validation credentials are rechecked and the permissions given to the role of the user is validated. If the User's role has access to that stored procedure then it is executed.

2.2.3 Recovery

K-safety is an important feature of VoltDB that enables recovery from failure. When one of a partition fails, VoltDB is restarted by giving the address of one of the remaining partitions as the host. A new partition is added in place of the failed partition. This new partition will receive the database schema, the stored procedure and its share of data until which time it will remain inactive. Until this process is completed the cluster is not K-safety enabled since one of the K nodes is still inactive. Once this node is active k-safety is enabled.

2.2.4 Features and Benefits

- The programming model consists of Schema file and all the work is done in terms of stored procedures since they do both processing and optimization on the server side. The schema and stored procedures go through project.xml files that generates .jar files in VoltDb compiler that gets distributed across all partitions.
- Scalability: RAM is increased across servers or add more machines to add capacity and increase performance.
- Availability and redundancy is achieved by K-safety. If the K-safety value, for example, is set to 3 then 3 copies of the data is stored across 3 different machines.
- Durability is handled with the help of command logging or snapshots where a snapshot of the database is taken at regular intervals so the data can be recovered even after crashing.

ClustrixDB

ClusterixDB is considered to be first ever SQL database that uses Scale-Out approach engineered for the cloud. It provides a lot of flexibility, resilience, availability and is sensitive to the performance characteristics like throughput and latency. This database is prominent among E-commerce website for the same reason.

3.1 Architecture

The architecture aims at providing massive transaction volume along with real time analytics In order to accommodate large-scale database systems ClusterixDB uses Shared nothing architecture that distributes data across several nodes and scales linearly. This completely eliminates the need for sharding and improves performance by reducing single failure bottleneck and parallelizing the processes.

In Shared nothing architecture each node has its own set of Query Compiler, Data Map, database Engine and Data(Table slices). The Query compiler distributes the fragments of query that it receives to the appropriate node using the data map that consists of details of all nodes and data that they contain. Each of the nodes have the ability to perform all Query executions with the help of database engine that perform all database operations. Hence each database handles its own reads and writes and the application always sees a single instance of database. On querying, the data is to be fetched from a node that has the required data in its local memory hence good networking is important between the nodes.

3.2 Data Distribution

ClusterixDB ensures that every data that is inserted into the database has a minimum of two replicas. A replica of the node can be present in each node if required. The data is distributed across the various nodes basically with the help of rebalancer. The distribution of data is said to be symmetric in ClustrixDB since all the nodes have same level of importance and have any special nodes.

Data distribution is possible because of the Independent index distribution method since it can continuously adapt to the increasing workload needs. In index distribution, each slice of data is allotted a particular range of 64-bit number space. The distribution key of each tuple in the slice is hashed into this 64-bit number space.

3.2.1 Rebalancer

Rebalancer ensures that the data is consistent and also distributed well in the cluster. On inserting data into a table the rebalancer first divides the inserted data into equal slices. The number of slices is usually equal to the number of nodes in the cluster. Hence as and when nodes are added or removed from the cluster the size of the slices differ. If any of the nodes in the cluster fails the rebalance ensures that the data is protected and that the replicas of the data exists. If any of the data slices are getting many hits then that slice is divided into equal halves and one of the halves is moved to a different node to reduce the workload on one single node.

3.3 Scaling

The elastic model of ClusterixDB makes it very easy to add or remove resources. ClustrixDB aims at enabling scaling of reads and writes without sharding. Although data is distributed there is no sharding since there horizontal partitioning across the different slices of data. This is done by index of multiple keys or by split table across multiple nodes.

As we know , ClusterixDB uses shared nothing architecture with each node having its own portion of data because of which the data is already scaled. Both Scale-out and Scale-in can be done by adding or removing nodes as per need and the data is automatically redistributed by the rebalancer.

3.3.1 High Availability

Each cluster is multinode in ClusterixDB and it provides write and read linear scale-out at each node which automatically makes it highly available. Automatic high availability is also possible because each of the data has two replicas across all nodes. During a planned outage of any node the data is replicated into a new node and during unplanned outage the replica of the data is used to copy the data contents to the new node. All the data replicas are consistent with respect to one another. Due to rebalancing data is always available for reads and writes and MVCC is used to perform reads without a need for locking during writing.

3.4 **Storage Enhancements**

Storage now has guardrails and warnings that prevent the database from being full with the help alert features to inform the user of the situation. They also have the ability to pre-allocate storage size during cluster creation which allows the user the complete control of the size which is configurable using the ‘alter’ Command.

3.5 **Security**

a. UDP connections are used to access administrative tool, process manager and internode communication along with TCP which is also used for Administration and upgrade, communications within database, Interface for Clustrix GUI. These are the security provided for internal communications amongst nodes

b. TCP connection protection is also provided to external access such as remote management of cluster access, Database access, Heartbeat monitor for cluster which checks if the node is still working.

c. The multiport cluster in Clusterix enables communication between the cores of various clusters.

d. Certain Users of Clusterix have a special privilege which allows them to access the database with OSAuth which authenticates a user with their OS instead of a password.

e. Accesses provided to the user are similar to MySQL where in few users have database level accesses while others have table level access.

f. ClustrixDB passwords and connections are also encrypted by SSL and SHA256 password security is provided to user Accounts.

3.6 Replication and Recovery

ClustrixDB provides a SQL compatible Asynchronous multi-point Replication which enables data access from different parts of the world. They also provide very fast parallel backup and load capabilities through parallel replication. Each node acts as both master as well as slave during replication.

Recovery of data is made easy with the help of data replicas. When one of the nodes fail and become unavailable, that node is removed from the cluster. The rebalancer will create additional copies of the data from the failed node in the background. The failed node is replaced with the new node containing replica of data or the data replica is just rebalanced among the remaining nodes of the cluster.

3.6.1 Fault Tolerance

ClusterixDB architectures ensure that the system does not go down. They do not have Single Point failures due to multiple nodes. If any of the nodes go down the data and the workload are automatically rebalanced. They have multiple copies of the data that is inserted because of which failure of nodes do not result in data loss. They have a value called MAX_FAILURES which holds the value of maximum possible failures in a cluster. As and when the value of MAX_FAILURES increases the latency also increases since the replica maintenance results in performance overhead.

3.7 Other Features

- ClusterixDB has a CLX tool that allows the interaction with few nodes simultaneously. It also gives status of clusters, obtain logs or files, executing linux commands and others.
- Better Guardrails and setup process during installation
- Faster Backup or restore which can exclude backup of particular tables using database.table Exclude command.
- Consistency and concurrency control are designed to provide non-locking parallelism during reads and writes

Implementation of ClustrixDB

4.1 Installation

The newest version of ClusterixDB available is Version9. However, the database is not available for general public implementation. Installation of ClustrixDB can be done via OS instructions as follows:

An Installer and a license key is to be obtained by contacting Clustrix sales representatives. The machines used as servers should be running on CentOS or RHEL 7.4 OS or above with root privileges. A minimum of 3 servers are to be present in each cluster of ClustrixDB with each server having about 8 to 32 cores. Database storage is either an SSD with RAID-0 or SATA HDD. They need a RAM of 64GB or higher along with front and backend networks. Configuration can be done for users to run and manage the database process.

In order to begin installation, login to each of the sessions on command prompt serving as a node as the root user with sudo permissions. Check if the required packages are present using the following commands:

```
sudo yum install bzip2 xz wget screen ntp ntpdate vim htop mdadm [9]

sudo yum-config-manager --enable rhui-REGION-rhel-server-optional [9]
```

All the firewall services are then disabled so that the nodes can communicate with one another. This is done by enabling NTP and disabling Firewalld as follows:

```
sudo systemctl start ntpd [9]

sudo systemctl disable firewalld [9]
```

Disk based logging is enabled to store the logs on the disk instead if the RAM . Hence on each node we perform cat Storage=persistent /etc/systemd/journald.conf [9]

The clusterix Installer will automatically create 3 users: root, clustrixDB Daemon(clxd), clustrixDB management(clxm).Now, we need to install the ClustrixDB installation file and untar it before running the installer script as follows:

```
tar xvfj current_version.el7.tar.bz2[9]

sudo ./clxnode_install.py[9]
```

Before the cluster is formed place all the nodes in a secure environment allowing access between all nodes. Set the cluster license and add nodes to the clusters using the commands:

```
Set GLOBAL license = license_key_value ; [9]

ALTER CLUSTER ADD ' node2_ip_address', 'node3_ip_address'; [9]
```

4.2 GUI

The clustrixDB GUI is automatically installed during the ClustrixDB installation and can be run through any web browser to connect to one of the ClustrixDB nodes by specifying port 8080. Initial login to the GUI is done using the temporary credentials after which we need to create a user using ‘Manage Users’ option in the GUI. The GUI consists of 5 important sections:

i. Health Dashboard: It displays the information such as number of transactions per second, CPU utilization, Memory Utilization such as in-memory and buffer management, Storage consumption across active nodes, storage by node, read and write latencies to mention a few.

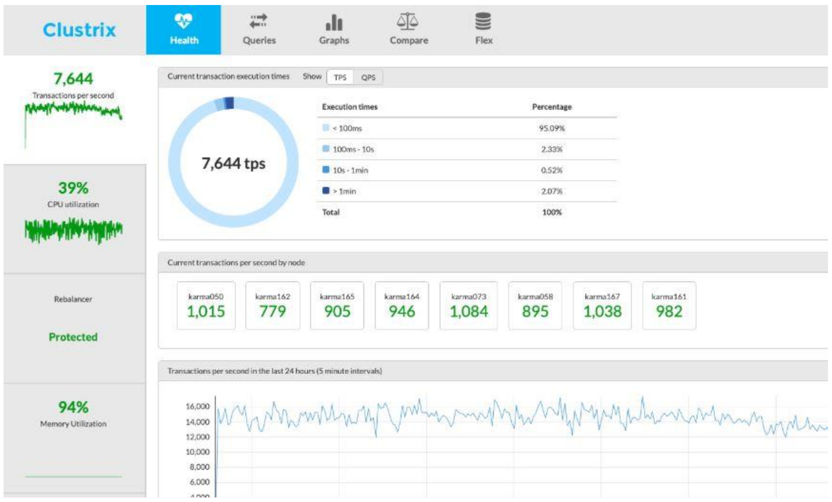
ii. Queries: They have two views that show the current and recent queries. The current query view is refreshed every 5 seconds and displays the current query being executed, its status, execution time, database on which query is run several IDs, Username and IP address of the machine.

    The recent queries view display all the queries executed recently along with the number of times it was executed, average execution time, Database on which querying is done, Average output Rows and other data.

iii. Graphs: It displays graphs for various attributes such as health, storage, Sessions, Queries, Transactions and performance.

iv. Compare: It displays the various comparisons of data attributes at different points in time. Minimum, Maximum, Average and Query latency of the CPU are the default attributes compared.

v. Flex: It displays the list of all nodes in the cluster and their statuses. Nodes can be easily added and removed here. On dropping nodes here the node undergoes Softfail.



Execution times	Percentage
< 100ms	95.09%
100ms - 10s	2.33%
10s - 1min	0.52%
> 1min	2.07%
Total	100%

Current transactions per second by node

karma050	karma162	karma165	karma164	karma073	karma058	karma167	karma161
1,015	779	905	946	1,084	895	1,038	982

4.3 <u>Querying</u>

ClusterixDB has multiple individual nodes in each cluster and the cluster grows as and when the nodes are added also resulting in the database growth. They also have a cluster database engine written from scratch called Sierra. Sierra breaks down queries into individual query fragments which are run in parallel across the nodes with the help of load balancer. Each of these fragments go through stages of execution. This helps attain better workload distribution and increase parallelism. The Query execution phases are carried out by sessions that parse the SQL, generate an execution plan, compile the query into binary fragments, find the required data's node, send the query fragment to that node to complete the transaction.

Few of the Queries of ClustrixDB are:

a. A create table query can be as simple as:

CREATE TABLE `Foo`(`pk` INT(11) NOT NULL auto_increment, `a` INT(11),`b` INT(11),`c` INT(11),PRIMARY KEY (`pk`) ,KEY `idx_ab` (`a`, `b`)) auto_increment=50002; [8]

b. A select Query:

SELECT Bar.a,Sum(Bar.b) FROM   Foo, Bar WHERE Foo.pk = Bar.pk GROUP BY Bar.a; [8]

c. Most of the SQL syntaxes of DML are supported by ClusterixDB such as:

Straight_Join, Distinct, [Left|Right|Outer] Join, Union, Limit, Asc, Order By, Having, For Update, Group By, Truncate, Load data Infile, Count(), Sum(), Group_concat(), Auto_increment.

Few unsupported DML:

Rollup, Except, Lock Tables, Not, Intersect, Collate, Cube, Ignore, All of Some

d. DDL statements supported by ClusterixDB:

ALTER CLUSTER ADD|DROP|SET MAX_FAILURES| RESIZE DEVICESDistribute statement is used to decide which of the columns or rows of the tables are to be distributed Several variations to the slices of the table can be made using the slices syntax.The number of replicas that are to be created of a data are decided using the replicas variable . If the value of REPLICAS is ALLNODES then the table is replicated on all the nodes.

Alter table…convert, Create sequence, Create view…with check option are few of the DDL statements that are not supported by ClustrixDB.

e. Administrative SQl statements supported by MySQL are:

Optimize Table, Analyze table, Repair table, Explain, Show binary logs, Show databases, Show character set, Show engines.Show slave hosts, Show master status.

Unsupported Administrative SQL statements:

Show errors, Show privileges, show user_statistics, show contributers, show columns with where clauses, flush, show authors

f. ClustrixDB supports MySQL syntax for Stored Procedures as well as triggers.

SUMMARY

ClustrixDB is a newSQL database that can scale data without any need to shard. Conventional RDBMS characteristics such as ACID compliance, real-time referential integrity, multi-table joins are maintained. They overcome many of the problems encountered during the use of traditional RDBMS and NoSQL databases. True data distribution with shared-nothing architecture is achieved. Scaling is achieved by simple methods of adding and removing resources and rebalancing data amongst the nodes. The data distribution remains transparent to the users as they access data using SQL interface from applications that do not require customization. Good fault tolerance is achieved by redundant copies of data present in various nodes. Automatic recovery of these nodes during unexpected failure helps prevent data loss. The database also provides various securities to the user accounts, communications within the database and interface communications along the GUI. They make processing of queries easier by dividing the queries into fragments and processing each of the query fragments in the nodes. ClusterixDB is, hence, a complete and fast database that can provide online schema change mechanism without any service interruptions.

REFERENCES

1. K. Kaur and M. Sachdeva, "Performance evaluation of NewSQL databases," 2017 International Conference on Inventive Systems and Control (ICISC), Coimbatore, 2017, pp. 1-5.
2. K. A. Doshi, T. Zhong, Z. Lu, X. Tang, T. Lou and G. Deng, "Blending SQL and NewSQL Approaches: Reference Architectures for Enterprise Big Data Challenges," 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Beijing, 2013, pp. 163-170.
3. Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, Pat Helland, The End of an Architectural Era(It's Time for a Complete Rewrite), VLDB '07, September 23-28, 2007, Vienna, Austria, Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.
4. Vadim Tkachenko, Rodrigo Gadea, Clustrix tpcc-mysql Benchmark Percona Inc.,
5. Michael Stonebraker and Rick Cattell, 10 Rules for Scalable Performance in 'SimpleOperation' Datastores, communications of the acm, vol. 54, no. 6, june 2011.
6. Andrew Pavlo, Matthew Aslett, What's Really New with NewSQL?, SIGMOD Record, Vol. 45, No. 2, June 2016
7. https://docs.voltdb.com/UsingVoltDB/IntroHowVoltDBWorks.php
8. http://docs.clustrix.com/display/CLXDOC/ClustrixDB+Documentation
9. https://docs.clustrix.com/display/CLXDOC/Prepare+the+Linux+Operating+System