A Multitenant Data Store Using a Column Based NoSQL Database

Aditi Sharma
Jaypee Institute of Information Technology, Noida, India
aditi.sharma@jiit.ac.in

Parmeet Kaur
Jaypee Institute of Information Technology, Noida, India
parmeet.kaur@jiit.ac.in

***Abstract*— As the world is frantically moving towards digitization, handling huge volumes of data is becoming complex and difficult to optimize. This is mainly because of increasing dimensions of data, its unstructuredness and increased need for storage space. To increase the efficiency of the cloud environment, many service providers have moved towards *multi-tenant architecture*. In a multi-tenant architecture, a number of individual applications termed as *tenants* work under a shared environment. In such a scenario, it is imperative that data of one tenant be isolated from that of other tenants. Further, it is also desirable to optimize the use of storage space. With these considerations, this paper proposes a multi-tenant database architecture for applications using a column-based NoSQL data store. The architecture has been implemented with the NoSQL database, Cassandra.**

Keywords- Multitenancy; distributed database; NoSQL; Cassandra, Materialized View

I. INTRODUCTION

With exponential increase in the quantum of data being generated, cost of managing the data centers has also increased tremendously. Considering this, many application providers have moved or have strategically been moving towards distributed environment. In the last few years the deployment of distributed computing and also the study of distributed systems have increased steadily. It is quite impossible to store, compute the humungous data on a single machine; thereby leading to the need for distributed and cloud based solutions for data storage.

Data being stored in distributed environment [1] or in a cloud is of huge volumes and requires large storage space to handle it. It has been observed that multiple users of cloud based applications, such as those using Software as a Service model; share some common data. Storage of the data for each application in the cloud leads to data that is usually redundant, due to multiple copies of the same datasets. For instance, consider a text file of 2MB that has been shared among 10 people over a network. Instead of storing this text file for all 10 users, a single instance of this file can be shared among all applications; thus, saving a lot of memory space which is one of the most critical resources in the cloud environment. This architecture in which multiple applications, i.e., tenants can share single data copy is termed as Multitenancy.

Multitenancy, in the context of data storage, is employed to reduce wastage of resources. Moreover, this leads to additional benefits such as lowering the requirements of maintenance and backups. However, it is also necessary that data of multiple tenants is isolated from each other. This isolation is required for security as well as protection from unwarranted modifications of data. Thus, a successful implementation of multitenant data store should ensure optimal use of storage space as well as isolation of data of tenants from each other. An additional requirement is the need for flexibility in data schemas. This is needed to allow a tenant to have some specific data attributes that may not be used by other tenants.

This paper proposes the implementation of a multi-tenant data store using a column based NoSQL database. NoSQL databases are increasingly being used by cloud based applications since they provide features of data distribution, flexibility of schema design and handling of unstructured nature of data [2,3]. One important family of NoSQL databases is column based databases which additionally are appropriate for handling sparse data. Cassandra [4] has been employed for implementation of the proposed multi-tenant datastore. However, the system can be implemented using any columnar database.

The rest of the paper is structured as follows: The related work is discussed in Section II. Design of the proposed datastore is presented in Section III. Results of implementation are described in Section IV. Finally, we conclude the article.

II. RELATED WORK

Multi-tenant database architecture has been discussed in literature in recent times. Most of the proposed systems are, however, implemented using relational databases. Authors of [5] have explained different data storage strategies in multi tenant architecture: (i) *Separate Application, Separate Database*: Maintenance cost of this model is quite high as compared to others as all the tenants function independently, are isolated from each other. (ii) *Shared Application, Separate Database*: Software is shared between all the tenants and it can be customized according to the individual tenant's requirement but their databases are different. (iii) *Shared Application, Shared Database:* Each tenant has different collection of tables i.e. schemas in the same database (iv)

978-1-7281-3591-5/19/$31.00 ©2019 IEEE

Shared Database, Same Schema: All tenants share same tables in the same database. Each of these offers a tradeoff between data isolation and flexibility.

Merits and demerits of a multi-tenant architecture have also been discussed in [5]. The authors emphasize that with the use of multi-tenant architecture hardware resources and software resources can be efficiently utilized, thus reducing the cost to a great extend. On the other side of the coin, a problem caused by a single tenant can affect the system as a whole because all tenants share same hardware.

The paper describes implementation of evidence based software engineering approach to conduct the research with Systematic Literature Review and Systematic Mapping Study. After performing systematic mapping study, it was found that Shared application-Shared database storage strategy is the most popular strategy to be used in the multi-tenant architecture. Number of factors contributed in this decision, foremost is the low cost involved in this storage strategy. Other factors are safety features and characteristics of tenants i.e. number, size of the database and the number of users per tenant.

This paper [6] addresses different challenges such as scalability, ease of development & maintenance, isolation etc. associated with multitenant architecture. Authors have listed challenges affecting Multi-tenant architecture, namely: configuration and extensibility. In the multi tenant environment, though all the tenants share resources, some components or features can be customized according to tenant specific needs. Thus, the system needs to be extremely configurable and extensible. Another problem that has been highlighted is of tenant isolation. Every tenant should feel that it is working in isolation. Configuration and extensions specific to a particular tenant should not directly affect other tenants. Implementation should support horizontal scalability, ease of development and maintenance. Development and maintenance of processes of a multi-tenant architecture should be less complex; otherwise it would lead to an increase in the application cost.

The work in [7] implements Domain Specific Modeling Language (DSML) to model a multi-tenant architecture. In this paper, to address multi–tenancy problems such as to generate or maintain cloud implementations Domain Specific Language (DSL) can be used. Different DSL are there, for instance different cloud services can be described by CloudDSL[8] using common cloud vocabulary, CloudML [8] handles provisioning, deployment and adaptations concerns.

Different domain classes and their relationships are also explained in this paper, namely; Database Model, Shard, Tenant Type, Table, Field, Enumeration and the tenant type can reference to tables whereas tables can reference other tables or enumerations.

To check the feasibility of DSML, authors have applied this to a case study- Education Centre which is a single tenant application, after re-engineering it is converted to a multi-tenant architecture. Authors have thoroughly explained all the steps involved in this evolution process along this proper evaluation of the proposed modeling language. This paper proposes a DSML to model multi tenant architecture and evaluates this modeling language using refracting methods and elastic database tools. In general, converting a single tenant application to a multi-tenant one is a costly and error prone process but after evaluating DSML over this case study, decrease in the number of errors was observed. Also, it was found that by using DSML time and effort can be reduced to a great extent.

The proposed work in this paper aims to use a new approach for implementing multi-tenancy with NoSQL databases.

III. PROPOSED SYSTEM:

Traditional relational databases are not well equipped to handle large volume of unstructured data. It is here that NoSQL databases are appropriate. The current work uses Apache Cassandra; a column based NoSQL database management system which is a distributed, scalable and highly available data store. It does not have a single point of failure. The design of a multitenant architecture using Cassandra has been illustrated in the Fig 1 and described next taking an event booking system as a SaaS application.

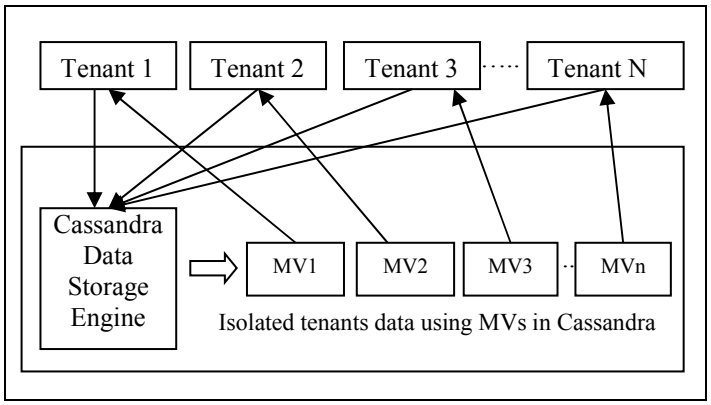


Fig 1: Architecture of proposed system

A. *Extracting data from the data source and design table schema:*

The proposed system follows a Shared Database, Same Schema Multitenancy model for an Event booking system. For this application there can be different tenants, namely Movie, Conference/Workshop, Adventure parks etc. The creation of the shared schema with all possible attributes (columns) using the query language of Cassandra, CQL is illustrated in Fig 2.

```
CREATE TABLE ticketbooking.events (
    event_code int ,
    actor  text,
    venue_code int,
    director_name text,
    duration int,
    Movie_genre  text,
    Event_type text,
    EVENT_title text,
    occu_perc decimal,
    ticket_price decimal,
    tickets_sold int,
    title_year int,
    total_sales decimal,
    location_venue text,
    event_organisors text,
    event_startingdate date,
    event_enddate date,
    email text,
    start_time text,
    end_time text,
    facebookpage text ,
    Primary Key(event_code,Event_type);
```

Fig 2: Events table

B. Import the extracted data into the Cassandra datastore:

The data for the multiple tenants is imported into the Cassandra database from the application.

C. Isolate tenants' data by creation of Materialized Views (MV):

Cassandra provides a feature to create materialized views. This feature is used by the proposed system to isolate a tenant's data from other tenants. Use of one MV for one tenant also allows for flexibility in schema being used by each tenant. Further, Cassandra is apt at handling sparseness in data. This is required as some columns values can be NULL for a particular tenant. For instance, for an event type "Conference", value of columns movie_genre, director_name, duration, actor will be NULL. Similarly, for an event type "Movie", value of columns facebookpage, email, event_organisors are NULL. Thus, each row can have multiple NULL values, to handle such sparse data we have used Cassandra's storage engine. Here, being a column family NoSQL data store, Materialized views can be created which only store the columns present in that row for each tenant. Each tenant's MV can be observed in Fig. 3.

Materialized View for Tenant MOVIE

```
create MATERIALIZED VIEW Movie
AS Select event_code, actor_1_name,actor_2_name
,actor_3_name,director_name,duration,movie_genre,event_titl
e, event_type, location_venue, occu_perc, start_time,
ticket_price, tickets_sold, title_year, total_sales, venue_code
from events
where event_code is not null and
 venue_code is not null and
 event_type is not null
 primary key(event_code,event_type,venue_code);
```

Materialized View for Tenant Conference/Workshop

```
create MATERIALIZED VIEW Conference
AS Select event_code, event_title, event_type,
location_venue, start_time, email, end_Time, event_enddate
,Event_Organisors, event_startingdate, event_title,
FacebookPage, venue_code
from events
 where event_code is not null and
venue_code is not null and
 event_type is not null
primary key((event_code,venue_code),event_type);
```

Materialized View for Tenant Music Festival

```
create MATERIALIZED VIEW MusicFestival
AS Select event_code, event_title, event_type,
location_venue, start_time, email, end_Time, event_enddate ,
Event_Organisors, event_startingdate, event_title,
FacebookPage, venue_code, ticket_price, tickets_sold,
occu_perc
from events
 where event_code is not null and
event_type is not null and
event_title is not null
primary key((event_code,event_type),event_title);
```

Fig 3: Materialized Views of some Tenants

IV. RESULTS

The Materialized views created [in Fig 6] are then used to process queries for different tenants in Cassandra data store engine, thus giving a sense of isolation to each tenant. Results have been tabulated in Fig 4.

Unlike MySQL, Cassandra does not explicitly displays time taken to execute a query, command TRACING ON/OFF is used to calculate the timestamp in Cassandra. Cassandra is good for large datasets, for small datasets Cassandra is much slower than MySQL [9].

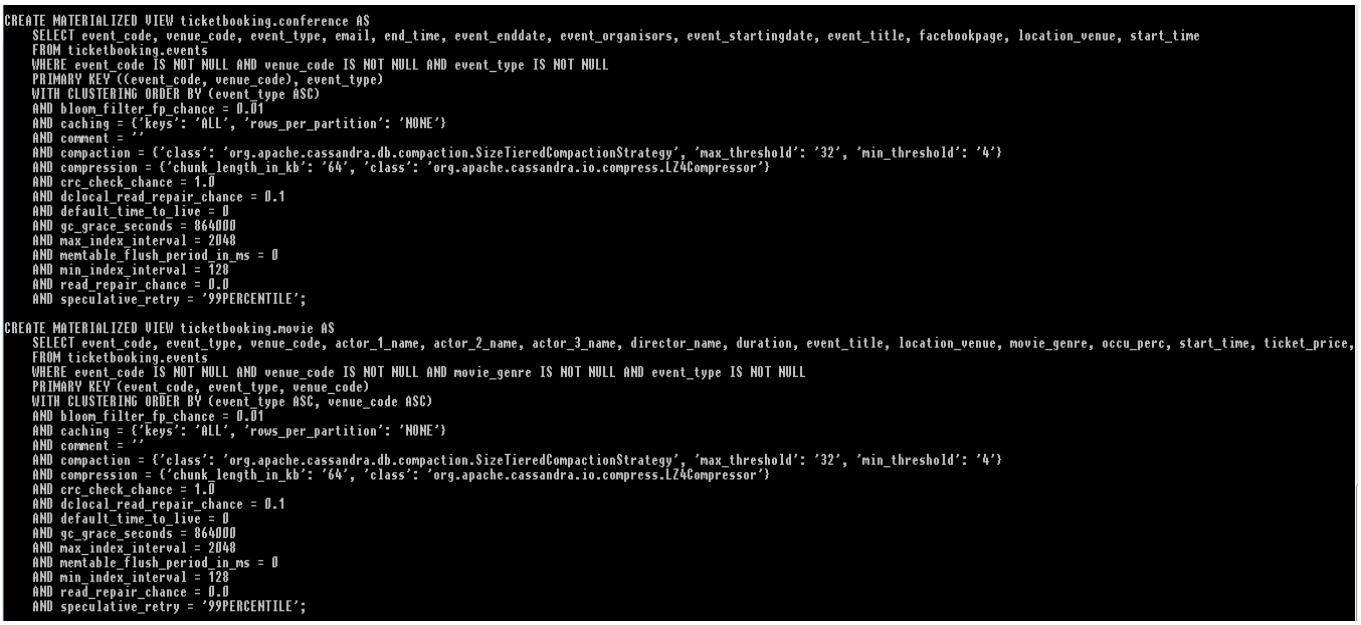
The results obtained show that multitenant architecture give desired results for large dataset in less time but it is not advisable to use Multitenant architecture for small dataset. Dataset used in the proposed system comprised of 360 rows, among these some were sparse i.e. maximum values of columns were null.

Table 1 : Sample Queries with processing time

S.No	QUERIES	PROCESSING TIME (miliseconds)
1.	Select * from movie ;	0.336
2.	Select event_title, director_name, location_venue from movie where event_type='Movie' allow filtering;	0.409
3.	select * from conference where event_type='Conference' allow filtering;	0.31
4.	select Sum(tickets_sold) from Movie where event_code =735 and event_title='The Departed';	0.26
5.	select Max(ticket_price) from MusicFestival where event_code =139 and event_type='Music';	1.10
6.	Select event_title, location_venue, start_time, event_startingdate from conference where venue_code in (1022,1061) and event_code in (731) ;	1.21
7.	select event_title, director_name,actor_1_name from Movie where movie_genre=' Crime|Drama|Thrillerr' and director_name=' Joss Whedon' Allow filtering;	0.98
8.	select count(*) from MusicFestival where event_code =735 and event_type='Music';	0.78

```
CREATE TABLE ticketbooking.events (
    event_code int,
    event_type text,
    actor_1_name text,
    actor_2_name text,
    actor_3_name text,
    director_name text,
    duration int,
    email text,
    end_time text,
    event_enddate date,
    event_organisors text,
    event_startingdate date,
    event_title text,
    facebookpage text,
    location_venue text,
    movie_genre text,
    occu_perc decimal,
    start_time text,
    ticket_price decimal,
    tickets_sold int,
    title_year int,
    total_sales decimal,
    venue_code int,
    PRIMARY KEY (event_code, event_type)
) WITH CLUSTERING ORDER BY (event_type ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Fig 4: Schema of Table Events



```
CREATE MATERIALIZED VIEW ticketbooking.conference AS
    SELECT event_code, venue_code, event_type, email, end_time, event_enddate, event_organisors, event_startingdate, event_title, facebookpage, location_venue, start_time
    FROM ticketbooking.events
    WHERE event_code IS NOT NULL AND venue_code IS NOT NULL AND event_type IS NOT NULL
    PRIMARY KEY ((event_code, venue_code), event_type)
    WITH CLUSTERING ORDER BY (event_type ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';

CREATE MATERIALIZED VIEW ticketbooking.movie AS
    SELECT event_code, event_type, venue_code, actor_1_name, actor_2_name, actor_3_name, director_name, duration, event_title, location_venue, movie_genre, occu_perc, start_time, ticket_price,
    FROM ticketbooking.events
    WHERE event_code IS NOT NULL AND venue_code IS NOT NULL AND movie_genre IS NOT NULL AND event_type IS NOT NULL
    PRIMARY KEY (event_code, event_type, venue_code)
    WITH CLUSTERING ORDER BY (event_type ASC, venue_code ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Fig 5: Materialized View for different Tenants

V. CONCLUSION

The paper has presented a NoSQL based architecture for a multi-tenant data store. A column based NoSQL family database, Cassandra has been used for` the implementation of the data store. The advantage of the proposed system lies in its ease of implementing data isolation among tenants while allowing flexibility of schema design or extension. Results of implementation show that the proposed Multitenant architecture provides desired data isolation to its tenant as well as performs well for large datasets but for small datasets, query processing time increases.

REFERENCES

[1] W. Zeng, Y. Zhao, K. Ou, and W. Song. “Research on cloud storage architecture and key technologies”. Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, 2009.

[2] J. Han, H. E, G. Le and J. Du. “Survey on NoSQL database”. 6th international conference on pervasive computing and applications IEEE , pp. 363-366, 2011.

[3] R. Burtica, E. M. Mocanu, M. I. Andreica and N. Tapus. ”Practical Application and Evaluation of NoSQL Databases in Cloud Computing”, in Proceedings of Systems Conference (SysCon), IEEE International., Vancouver, 2012, pp. 1-6.

[4] G. Wang and J. Tang. “The nosql principles and basic application of cassandra model”. International Conference on Computer Science and Service System, pp. 1332-1335. IEEE.2012.

[5] G. Karataş, F. Can, G. Doğan, C. Konca and A. Akbulut, "Multi-tenant architectures in the cloud: A systematic mapping study". International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, pp. 1-4,2017.

[6] A. Jumagaliyev, J. Whittle and Y. Elkhatib, "Using DSML for Handling Multi-tenant Evolution in Cloud Applications," IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, pp. 272-279, 2017.

[7] G. C. Silva, L. M. Rose, and R. Calinescu, “Cloud DSL: A language for supporting cloud portability by describing cloud entities,” in Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud, co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, CloudMDE@MoDELS, 2014, pp. 36–45.N. Ferry, G.

[8] Brataas, A. Rossini, F. Chauvel, and A. Solberg, “Towards Bridging the Gap Between Scalability and Elasticity,” in Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER)—Special Session on Multi-Clouds. SCITEPRESS, 2014, pp. 746–751.

[9] M. R. Murazza and A. Nurwidyantoro. “Cassandra and SQL database comparison for near real-time Twitter data warehouse.” 195-200. 10.1109/ISITIA.2016.7828657, 2016.