**International Journal of Electrical and Computer Engineering (IJECE)**
Vol. 14, No. 2, April 2024, pp. 1720~1729
ISSN: 2088-8708, DOI: 10.11591/ijece.v14i2.pp1720-1729

# A systematic review of in-memory database over multi-tenancy

**Arpita Shah, Nikita Bhatt**
Chandubhai S. Patel Institute of Technology, Faculty of Technology and Engineering, Charotar University of Science and Technology, Gujarat, India

**Article Info**

***Article history:***

Received Aug 12, 2023
Revised Oct 17, 2023
Accepted Oct 20, 2023

***Keywords:***

Cloud computing
Column-store
In-memory database
Multi-tenancy
Relational database query

**ABSTRACT**

The significant cost and time are essential to obtain a comprehensive response, the response time to a query across a peer-to-peer database is one of the most challenging issues. This is particularly exact when dealing with large-scale data processing, where the traditional approach of processing data on a single machine may not be sufficient. The need for a scalable, reliable, and secure data processing system is becoming increasingly important. Managing a single in-memory database instance for multiple tenants is often easier than managing separate databases for each tenant. The research work is focused on scalability with multi-tenancy and more efficiency with a faster querying performance using in-memory database approach. We compare the performance of a row-oriented approach and column-oriented approach on our benchmark human resources (HR) schema using Oracle TimesTen in-memory database. Also, we captured some of the key advantages on optimization dimension(s) are the traditional approach, late-materialization, compression and invisible join on column-store (c-store) and row-base. When compression and late materialization are enabled in a query set; it improves the overall performance of query sets. In particular, the paper aims to elucidate the motivations behind multi-tenant application requirements concerning the database engine and highlight major designs over in-memory database for the tenancy approach on cloud.

***Corresponding Author:***

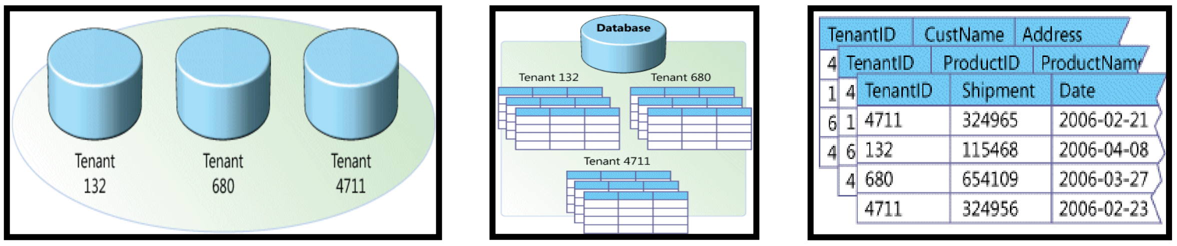
Arpita Shah
Chandubhai S. Patel Institute of Technology, Faculty of Technology and Engineering, Charotar University of Science and Technology
Changa, Gujarat 388421, India
Email: arpitashah.ce@charusat.ac.in

## 1. INTRODUCTION

Traditional enterprise applications are now running towards the host-based and on-demand model instead of on-premise deployment. On the other hand, service providers can automate tasks by consolidating tenants onto a single machine referred to as multi-tenant. On the database level, numbers of read and write queries are balanced to perform placement over various in-memory platforms to support row and column-oriented database(s) [1]. On other hands a dictionary-based compression, which support the architecture of multiple users. Also, multi-tenancy can be a core of cloud computing which could be solved with the same resources or other software applications. Multi-tenancy also added a component of security in which, to guarantee the isolation of data from multiple users. Magalhaes *et al.* [2] have analyzed log data distribution in a hosted multi-tenant application environment like Microsoft Azure. There is a challenge to consolidate data of multiple tenants over the same database management system (DBMS) to reduce the cost of operation for profit maximization. As a response to advancements in computer main memory technology and its affordability, the exploration of in-memory databases (IMDB) commenced early 80s [3]. The central focus of this research is to analyses on the capabilities of computer main memory by hosting the entire database, aiming to achieve rapid access and facilitate real-time analysis. This work explores the significance of integrity in the area of in-memory databases for tenant placement [4]; which can be used as public and private cloud services in the future. Within the database layer, multi-tenancy can be leveraged to enable multiple customers, or tenants, to utilize a single database. Multi-tenancy can materialize primarily at the database layer within the software as a service (SaaS) application [5]. This implies that customers leverage a singular shared application and database instance, thereby efficiently utilizing the same hardware resources. Simultaneously, the tenant remains completely isolated from one another.

Following major schemas [6]–[8] are highlighted multi-tenancies that are applicable for database sharing among tenants are concerned. Separate databases, application code and computing resources are distributed among all the tenants that are sharing a server. Figure 1 shows the multi-tenancy database architecture. In Figure 1(a) each tenant/user consists of its own set of data which is sensibly inaccessible from all other tenants' data. The DBMS ensures security measures that prevent any undefined or deliberate access by one tenant to the data of other tenant.

Shared database, separate schemas-an alternative depicted in Figure 1(b) involves consolidating multiple tenants within a single database. Each tenant holds a distinct collection of tables, organized into a schema exclusively for that tenant's needs. Shared database, shared schema-a third method shown in Figure 1(c) uses a shared database along with a set of tables to accommodate huge amounts of data from several tenants. A table aggregates records from multiple tenants and is stored in any order. The *TenantID* column is used to associate each record with the appropriate tenant. In multi-tenant design, there are many approaches to designing multi-tenancy database models. Table 1 shows the suitable pattern for designing a multi-tenancy database.



| TenantID | Shipment | Date |
|---|---|---|
| 4711 | 324965 | 2006-02-21 |
| 132 | 115468 | 2006-04-08 |
| 680 | 654109 | 2006-03-27 |
| 4711 | 324956 | 2006-02-23 |

(a) (b) (c)

Figure 1. Multi-tenancy database architecture (a) separate databases (with the tenant) (b) shared databases, separate schemas and (c) shared database, shared schema

Table 1. Pattern for designing multi-tenancy database

| Approach | Security pattern | Configurable pattern | Scalability pattern |
|---|---|---|---|
| Separate databases, separate schema | Protected database tables<br>Renter data encryption<br>Reliable database connections | Customized columns | Single tenant scale-out |
| Shared database, shared schema | Reliable database connections<br>Occupant view filter<br>Renter data encryption | Pre-assigned fields<br>Name-value pairs | Tenant-based parallel Partitioning |
| Shared database, isolated schemas | Reliable database connections<br>Protected database tables<br>Renter data encryption | Customized columns | Tenant-based parallel partitioning |

To development tenancy model is also to be analyzed. The following criteria are used to assess the general tenancy model [9]–[11]. In a single-tenant system, each customer (tenant) manages a dedicated database instance hosted on a dedicated physical server. In such a system, the service provider's maintenance expenses can be substantial, even when tenants do not use their systems continuously or at full capacity. Conversely, the multi-tenancy model allows multiple customers to share resources on a single machine. This approach, facilitated by an integrated administration framework, enhances system management efficiency and maximizes resource utilization. The technique was initially embraced on a significant scale by the SaaS provider Salesforce.com [12], [13]. Multi-tenancy can be accomplished through three distinct methods, each varying in granularity: shared machine, shared database instance, and shared table. The choice of approach depends on the nature of the extensive application [14], [15]. Table 2 depicts that in the context of an in-memory computing engine, the important parameters are relevant to the tenant type.

Table 2. Tenant types

| Type | Table Content | Meta Data |
|---|---|---|
| Tenant-Independent | Data residing within the tenant's system; Accessible for reading by other tenants | Stored within the tenant; Accessible for read by other tenants |
| Tenant-Dependent | Distinct copies of the table are present in individual tenants; tenant-specific data | Tenant-specific enhancements, such as additional columns, remain stored as confidential metadata; the definition of the table is centrally stored within the system tenant, accessible for reading by other tenants |
| Tenant-Private | Exclusive tenant data with restricted access from other tenants | Maintained as tenant-specific confidential metadata. |

Contribution of proposed study as, i) analyze different in-memory tendency for effective query execution, ii) proposed cloud integrated in-memory based model for column and row-oriented database, and iii) authors have simulated proposed integrated architecture for benchmark dataset. Rest of the paper as section 2 contain background study for different in memory database and also demonstrates comparative analysis with state-of-the-art (SOTA). Section 3 represent result analysis for various in-memory database schema.

## 2. LITERATURE SURVEY

The research presented in reference [16] offers an extensive survey on service-level agreement (SLA) based cloud studies, aiming to assess the current research areas with unresolved challenges. The primary focus of this study revolves around the resource provisioning stage within the SLA lifecycle. However, it also highlighted on its impact and implications on other phases of the lifecycle. Through this research, significant contributions are made towards the initial features of cloud SLAs and their autonomic management. These valuable insights serve as a strong motivation for future research endeavors and the development of industry-oriented solutions. In Pythia [17] classification of database tenants is done with a complex set of features and for establishing the configuration of tenant categories that may lead to performance violations. Many studies considered only a minimal number of SLA parameters. Resource allocation is mostly attempted using heuristics, policies, and optimization techniques. Liu [18] addresses placing a tenant aiming to optimize performance objectives while minimizing costs. They developed an algorithm associated with distributed replicated block device (DRBD), whereas we utilize a distinct methodology that is estimated using a multi-tenancy workload with in-memory database engine. Lang *et al.* [19] present a study that looks at the same time tenant placement concurrently with server setup. As a workload benchmark using transaction processing performance council (TPC-C), the outcomes are measured. Their research on finding arrangements that can handle the real-world inconstancy in the Microsoft generation traces. To enhance the performance of in-memory databases can be achieved by caching [20]. For faster access, caching involves the mechanism by which on-disk databases store frequently accessed records in memory. In any case, caching helps to speed up the recovery of data or to perform database read operations faster.

### 2.1. Data management strategies–in-memory database

Table 3 depicts a summarization of in-memory information management frameworks represented through their data models, supported workloads, index mechanisms, and strategies for controlling memory overflow. After conducting a comparison, our recommendation is to opt for Oracle's relational database system to effectively handle both online transaction processing (OLTP) and online analytical processing (OLAP) workloads [21]. Smart memory management is to be required for minimizing the number of servers are required to accommodate a fixed number of tenants. We have integrated Oracle's in-memory structured query language (SQL) into our work as a prerequisite for multi-tenancy [22], [23].

In a particular application, a crucial requirement for the database system is seamless resource-sharing transparency. The application needs to work on a tenant database, providing the illusion that each tenant has its dedicated database instance. For example, the application should have the capability to establish a database connection for a particular tenant. Subsequently, all queries made through this connection should be confined to the tenant's data. This approach eliminates the need for the application to incorporate "special logic" to handle queries for tenant-specific information. Ensuring transparency of resource sharing also requires isolating each tenant from the impact of other coexisting tenants. For instance, if one tenant experiences a crash, it should not affect or interrupt other tenants. Additionally, in scenarios where a tenant demands peak database performance, other tenants sharing the same resources must still fulfill their respective service level agreements (SLA) without compromise [24], [25].

Table 3. Comparison of in-memory databases

| | System | Data model | Workload | Indexes | Memory |
|---|---|---|---|---|---|
| Relational databases | H-store | relational | OLTP | hashing, B+ tree, Binary tree | anti-caching |
| | Hekaton | relational (row) | OLTP | latch-free, hashing | project Siberia |
| | SAP HANA (High-performance Analytic Appliance) | relational, graph, text | OLTP/ OLAP | Timeline index, B+ tree | Table/partition-level swapping |
| | Oracle | relational (row/column) | OLTP/ OLAP | B+ tree, Bitmap index | table partition, compression |
| NoSQL databases | MongoDB | document | object operation analytics | B-tree | N/A |
| | Radis | key-value | object operation | hashing | compression |
| | Cassandra | column-based | object operation analytics | CF-based index | N/A |
| | RAM Cloud | key-value | object operation | hashing | N/A |
| Graph Database | Bitsy | graph | OLTP | N/A | N/A |
| | Trinity | graph | graph operation | N/A | N/A |
| Big Data Analytics Systems | Talend | key-value | analytics | N/A | N/A |
| | Spark | Resilient distributed datasets (RDD) | analytics | N/A | block-level swapping |

## 3. METHOD

Proposed study used in-memory DBMS based architecture to improve query process with multi-tendency database. Authors have demonstrated some notational conventions, and that helps to understand a few basic principles and formulas. A Cartesian product or a cross product which takes combining two relationships, R1 and R2, results in a new relationship R1×R2. Each of these relationships has nR1 and nR2 attributes, with respective cardinalities of |R1| and |R2|. By utilizing the × operator, a collection of ordered pairs (r1, r2) is generated, where r1 is drawn from R1 and r2 from R2. This operation culminates in the creation of a novel relation, R3, characterized by nR3 attributes equivalent to the sum of nR1 and nR2 and |R3|=|R1|. |R2| tuples are returned. Projection works to filter or rearrange the attributes within its input relation. It is expressible as $\pi\ j1 \ldots\ jn(R)$, where $j1$ through jn form a sorted sequence that represents the attributes belonging to the relation $R$ which consists projected result. The selection of an attribute gives how the selection is projected on a specific column is:

$$selectivity\ yp = \frac{\sigma p(R)}{|R|} \tag{1}$$

In general, the *SELECT* command is commonly used. The "*SQL SELECT*" is defined as (2).

$$select\ \pi_{j1,\ldots,jn}(R)\ From\ R\ where\ \sigma_{a\theta b}(R) \tag{2}$$

From a data management standpoint, adopting a well-structured framework can significantly enhance efficiency. When dealing with a row layout, the central processing unit (CPU) fetches a portion of data from the specific tuple into the cache. This necessitates the retrieval of bytes from the main memory with consideration of the processing core's speed, as indicated by (3).

$$\text{Response time} = \frac{byes\ (to\ read\ from\ main-memory)}{speed\ (bytes\ /ms/core)}\ microseconds \tag{3}$$

In the case of column layout, the CPU will transfer certain attributes of the provided object into the cache. So, using (3) reading bytes from main-memory (attribute values) with consideration of the speed of processing core, using (3). Equation (4) and (5) will be used to calculate ate memory consumption of the index structure.

$$I_m = D_l\ \lceil \log 2(\ IP_l\ )\rceil + AV_l \lceil \log 2(\ AV_l)\rceil \text{ bits} \tag{4}$$

$$I_m = (Dl + AV_l)\ (\lceil log2(AV_l)\rceil) \text{ bits} \tag{5}$$

where $I_m$ index memory consumption, $D_l$ length of the dictionary, $IP_l$ length of the position of the index, $AV_l$ length of attribute vector. The number of entries to be retrieved from the index position (6) depends upon the specific column (with the distribution of values). For the read more frequently usage attribute, requisite to read additional entries, and for the less frequently usage attribute needs less reading. So, to read $AV_l \div D_l$ entries, with the width of $\lceil \log 2(AVl) \rceil$ bits.

$$Index - positions = \frac{AVl \cdot \lceil \log 2(AVl) \rceil}{Dl} \tag{6}$$

The database cost with tenancy model per selected time frame is:

$$Cost_{d,y} = base_{d,y} + \lambda\, a(w_d, y) \tag{7}$$

Consist of the base cost, $base_{d,Y}$:

$$base_{d,y} = \frac{1}{N_{d,y}} \sum_{ti=1}^{N_{d,y}} (CPU_{t,ti}) \tag{8}$$

where, $Cost_{d,y}$ cost of database $d$ during each training period $y$, $base_{d,y}$ base cost of database $d$,$a(w_{d,}\ y)$ finding additional cost, $N_{d,y}$ database $d$ exists in training period $y$ during a number of given slices.

## 4. RESULTS AND DISCUSSION

The query accessing approach is very important over the in-memory database. There are two storage approaches: row-oriented and column-oriented [26], [27]. From a database application perspective serving RDBMS with an in-memory approach over multi-tenant support scalability, simple deployment, and customer isolation. Table 4 shows a comparison of IMDB features of Oracle TimesTen, SAP high-performance analytic appliance (SAP Hana), and GridGain.

Table 4. IMDBs features comparison

| Feature | Oracle TimesTen | SAP HANA | GridGain |
|---|---|---|---|
| In-Memory Data Storage | Yes | Yes | Yes |
| Data Persistence | Optional | Yes | Yes |
| Key-Value Data Model | No | No | Yes |
| SQL Support | Yes | Yes | Yes |
| Atomicity, Consistency, Isolation, and Durability (ACID) Transactions | Yes | Yes | Yes |
| Horizontal Scaling | Yes | Yes | Yes |
| Secondary Indexing | Yes | Yes | Yes |
| Data Partitioning | Yes | Yes | Yes |
| Data Replication | Yes | Yes | Yes |
| In-Memory Caching | No | No | Yes |
| Work with relational database management system (RDBMS)s | Yes | No | No |
| Organize data with a column-based storage strategy | Yes | Yes | Yes |
| Organize data with a row-based storage strategy | Yes | Yes | No |

### 4.1. Row format v/s column format–experiment

In a column-oriented approach Table 5, the database maintains each attribute within its distinct column structure. Table 6 indicates a row-oriented format database from the sample data set shown in Table 7, every new record or transaction stored in the database is regarded as a new row in a table. A row format allows fast access to all of the columns in a row because all the data for a given row are kept together in memory and storage. Therefore, it is ideal for applications that require online transaction processing [28].

Table 5. Column-oriented data

| Column-based Tuples |
|---|
| 1, 2, 3; |
| Joe, Lara, Bill; |
| Finance, IT; |
| $500, $700, $500; |

Table 6. Row-oriented data

| Row-based Tuples |
|---|
| 1, Joe, Finance, $500; |
| 2, Lara, Finance, $700; |
| 3, Bill, IT, $500; |

Table 7. IMDB-sample data

| ID | Name | Dept | Salary |
|---|---|---|---|
| 1 | Joe | Finance | $500 |
| 2 | Lara | Finance | $700 |
| 3 | Bill | IT | $500 |

A column-oriented approach which is ideal for analytical and transaction processing. It facilitates summarized data retrieval when a query accesses a significant portion of the dataset but selects only a limited number of columns (*e.g.* $ID = 3$, $Dept =$ information technology (IT)) from the sample data set shown in Table 7. Typically, row-oriented database management systems often permit data definition operations, whereas in a column-oriented database, individual columns are stored separately from one another, each residing within its own distinct block. Using the human resources (HR) schema dataset [26] shown in Figure 2, we have tested the performance of row-oriented and column-oriented approaches.

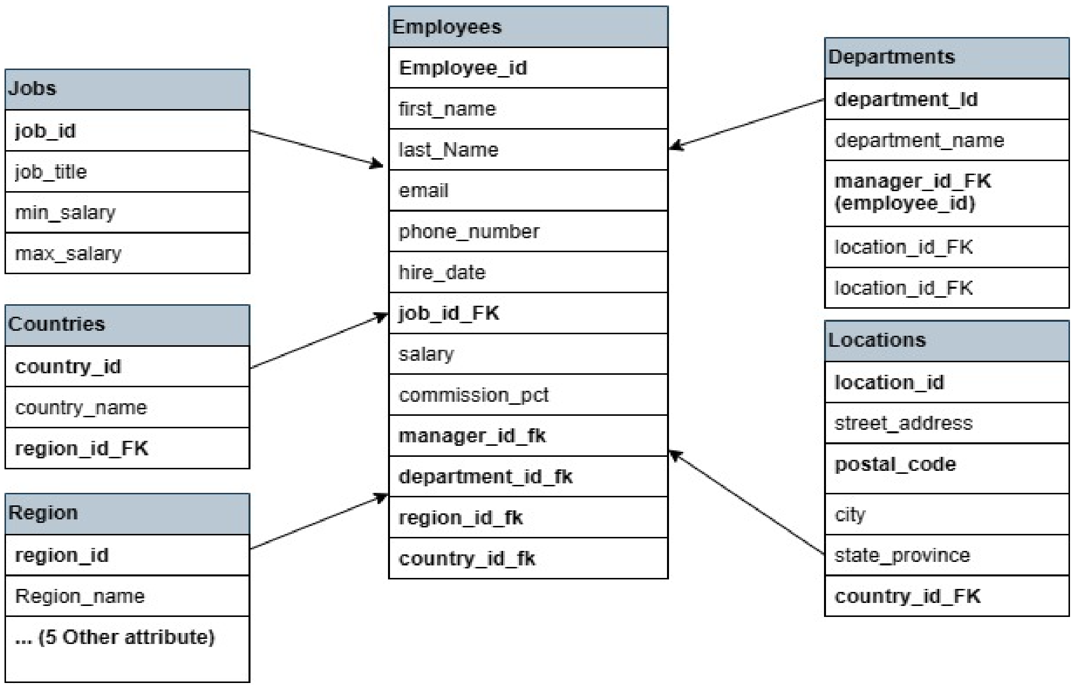


Figure 2. HR-schema benchmark

According to the relational database application most queries are executed for analytical purpose and more projected on the data attribute(s). To check the performance of row and column-oriented approaches; we have deployed Oracle in-memory product to check the output of sample queries based on HR-schema. Based on the sample benchmark we have evaluated sample queries which are divided into three categories according to the star-schema dimension attribute(s) sets to check the performance of row-oriented and column-oriented. A fundamental 'scale factor' is available for adjusting the benchmark's size. The proportions of the tables are determined in relation to this scale factor, with a value of 10 chosen for selectivity. The size by which we make the larger is described by its scale factor. For example, the size of the employee's table is 4,000,000 which represents a scale factor × 4,00,000. Selectivity primarily refers to the characterization of a predicate. Selectivity can be calculated based on a table as "[*No. of rows filter based on predicate value/No. of rows in the table*]". Table 8 demonstrate execution remarks for the HR-schema consists into ten queries divided into three categories.

Runtime for any selectivity (similar to the given sample example) based on CT-1, CT-2, and CT-3, the query execution and comparing the performance of a “column store” called C-store and Row-store on HR-schema shown in Figure 3. C-store performs better than the best case of Row-store, though even they access the same amount of I/O is similar. So, based on the experimental we have chosen the storage technique as C-store.

Table 8. Category wise query analysis for HR-schema

| Category | Solution | Remark |
|---|---|---|
| CT-1: Contain three queries, consists restriction with one-dimension attribute, *salary* and *department_id* (employees table). | *SELECT first_name FROM employees WHERE department_id < 40 AND salary between 20000 and 60000 ORDER BY department_id.* | The employee's selectivity for the three queries are $0.6\times10^{-2}$, $2.3\times10^{-3}$ and $3.5\times10^{-4}$. |
| CT-2: Contains three queries, consisting of restriction on two-dimension attributes are *min_salary*, *max_salary* for each group of rows with the same job code in the employee's table. | *SELECT e.last_name, m.last_name manager, m.salary, j.job_title FROM employees e, employees m, jobs j WHERE e.manager_id = m.employee_id AND m.salary BETWEEN j.min_salary AND j.max_salary AND m.salary > 15000.* | The employee's selectivity for the three queries are $5.0\times10^{-3}$, $1.2\times10^{-3}$ and $1.8\times10^{-4}$. |
| CT-3: Contain four queries, consists restriction on three-dimension attributes are *departments*, *employees*, and *locations*. | *SELECT l.city, d.department_name, e.job_id, SUM(e.salary) FROM locations l, employees e, departments d WHERE d.location_id = l.location_id AND e.department_id = d.department_id AND e.department_id > 80 GROUP BY CUBE (l.city, d.department_name, e.job_id)* | The employee's selectivity for the four queries is $2.5\times10^{-2}$, $1.1\times10^{-3}$, $3.5\times10^{-4}$ and $5.7\times10^{-6}$. |

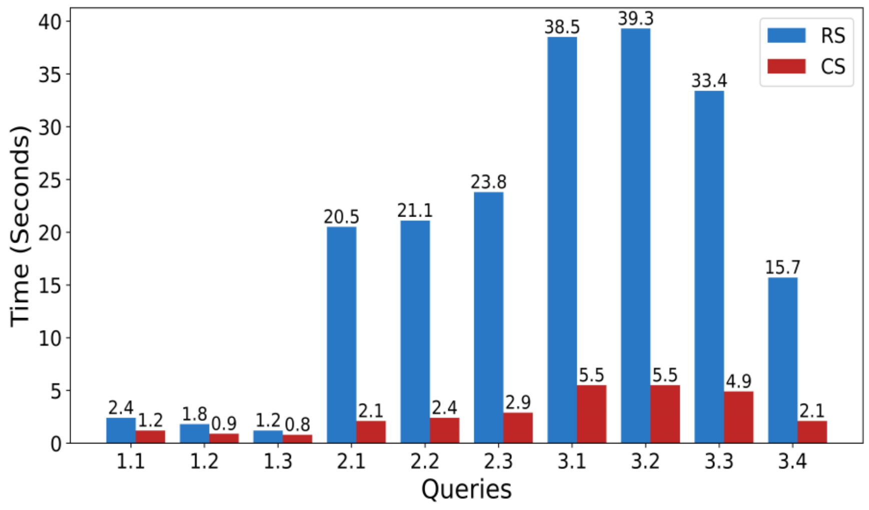


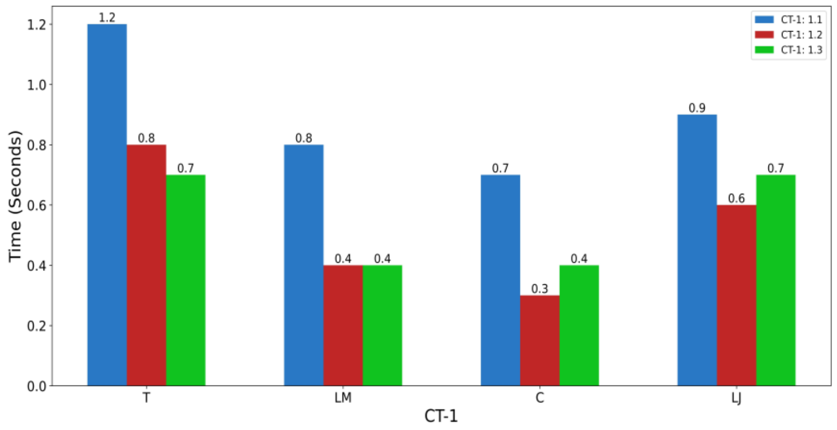
Figure 3. Performances of c-store (column-store) and row-store

## 4.2. Execution of column-oriented approach

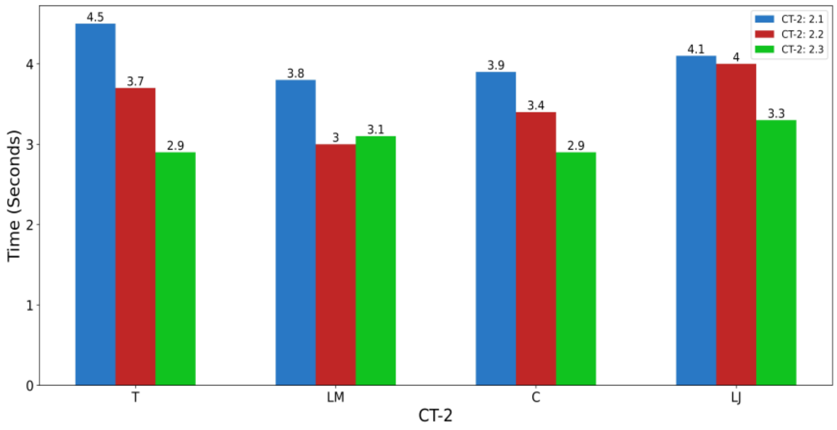
Now, we further tested datasets with different optimization dimension(s) are traditional approach, late-materialization, compression and invisible join on C-store and Row-based (HR-schema). These experiment(s) are carried out on same set of queries (mentioned as CT-1, CT-2, and CT-3). Applying C-store approach, we have demonstrated the impact on the performance of an expansion of column-oriented execution strategies, along with vectored query processing, compression, and a join. Column-oriented approach produces some critical system improvements with primary attributes, decreased tuple overhead, rapid merge joins of looked after records, run-duration encoding over multiple tuples.

In traditional approach tuple representation interface to extract the selected attributes from dataset resulting tuple-at-a-time processing; in which one or more function calls are to be required for extracting the data set during each tuple operation. In the case of C-store approach, blocks values of the same column are return to an interface using single function call. In addition, necessity exists for attribute extraction, and when dealing with a fixed-width column, these values can be sequentially accessed as an array. Late materialization is more suitable for column-oriented approach. Therefore, in most query dataset with multiple attributes jointly forms a "row" of information for that entity. Thus, such kind of join based materialization of tuple is common for column storage operation.

Figure 4(a) and (b) show different optimization dimension(s): traditional (T), late materialization (LM), compression (c), and invisible join (IJ) are processed on query set CT-1 and CT-2. When compression and late materialization are enabled in a query set; it improves the overall performance. Invisible join improves the performance at some extent and compression with late materialization improved the performance by 60-75%. In conclusion, the most beneficial optimization dimensions are "compression" and "late materialization" on similar benchmark schema with identical scale factor.



(a)



(b)

Figure 4. Optimization dimension on the query set (a) with one-dimension attribute(s) (b) with two-dimension attribute(s)

## 4.3. Predicting performance of in-memory over tenancy

To maintain effective data separation between tenants, every query or transaction have to carry tenant-specific information. This can be accomplished by including explicit tenant identifiers directly within SQL queries, or alternatively, by utilizing an intermediary middleware layer responsible for managing and enforcing tenant context. So, an important key factor to access SQL queries [C-Store] in optimized time which supports as per our recommended in-memory product with tenancy model. Based on (9), defines the workload incurred by a tenant as a function of its request rate and size; where m and n can be estimated using least-squares method.

$$w(t_r, t_s) := \frac{t_r\ t_s^{-m}}{10^n} \tag{9}$$

The predicated workload is calculated using (10). Typically, sizes and request rated of tenant are different on a server. So, total workload *T* on server as the sum of the workloads over the set of all tenants.

$$w(T) := \sum_{t \in T} \frac{t_r\ t_s^{-m}}{10^n} \tag{10}$$

To investigate in-memory database performance, an examination is conducted on the relationship between workload and response time within a relational context. Tests are performed using varying dataset sizes, ranging from 1 to 2.5 GB, employing a single database instance. As depicted in Figure 5, the projected workload is computed. The graph exhibits a linear increase up to a workload of 0.8, beyond which it transitions into exponential growth.

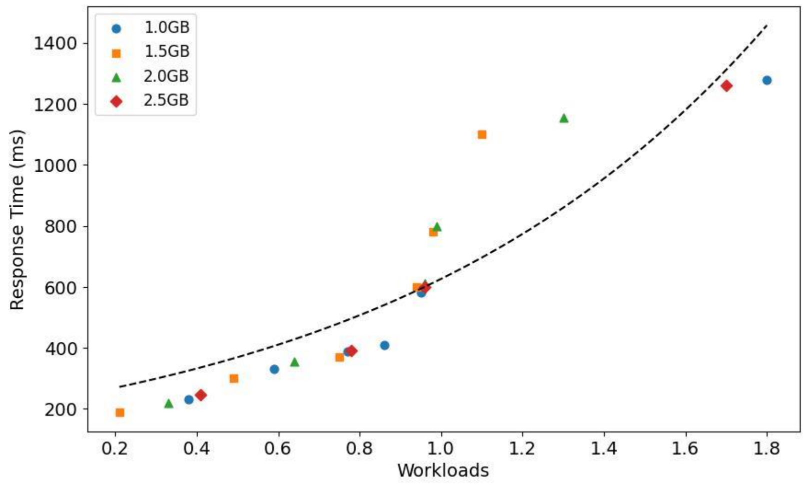


Figure 5. Single instance capacity

## 5. CONCLUSION

Multi-tenancy represents an innovative software architectural approach in which a singular application instance (or customized data and configuration) operates within the infrastructure of a service provider. In this analysis, we have conducted a comprehensive review of research studies pertaining to the multi-tenancy approach on cloud platforms and in-memory databases. The results demonstrate the most suitable storage approach is column-oriented for the given query sets which is the most suitable approach for the relational model. Based on our study, the selection of record storage is also a challenging task. Combined with multi-tenancy and database solutions, gives better performance. Multi-tenancy approach for column-based dataset has faster execution compare to row-based dataset. Applications that utilize a relational database with column storage should consider employing compression and late materialization based on simulated result(s). Future work of proposed study, evaluate hybrid storage models where allowing data processing model to choose between row-oriented or column-oriented per query or table. Additional study is required for dynamic approach to optimize data storage based on specific use cases.

## REFERENCES

[1] J. Krueger, F. Huebner, J. Wust, M. Boissier, A. Zeier, and H. Plattner, “Main memory databases for enterprise applications,” in *2011 {IEEE} 18th International Conference on Industrial Engineering and Engineering Management*, Sep. 2011, pp. 547–557, doi: 10.1109/icieem.2011.6035219.

[2] A. Magalhaes, J. M. Monteiro, and A. Brayner, “Main memory database recovery: A survey,” *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–36, Mar. 2021, doi: 10.1145/3442197.

[3] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. A. Wood, “Implementation techniques for main memory database systems,” *ACM SIGMOD Record*, vol. 14, no. 2, pp. 1–8, Jun. 1984, doi: 10.1145/971697.602261.

[4] J. Ru, J. Grundy, Y. Yang, J. Keung, and L. Hao, “Providing fairer resource allocation for multi-tenant cloud-based systems,” in *Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015*, Nov. 2016, pp. 306–313, doi: 10.1109/CloudCom.2015.30.

[5] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper, “Extensibility and data sharing in evolving multi-tenant databases,” in *Proceedings - International Conference on Data Engineering*, Apr. 2011, pp. 99–110, doi: 10.1109/ICDE.2011.5767872.

[6] G. Karataş, F. Can, G. Doğan, C. Konca, and A. Akbulut, “Multi-tenant architectures in the cloud: A systematic mapping study,” Sep. 2017, doi: 10.1109/IDAP.2017.8090268.

[7] H. Yaish and M. Goyal, “A multi-tenant database architecture design for software applications,” in *Proceedings - 16th IEEE International Conference on Computational Science and Engineering, CSE 2013*, Dec. 2013, pp. 933–940, doi: 10.1109/CSE.2013.139.

[8] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang, “Native support of multi-tenancy in RDBMS for Software as a Service,” in *ACM International Conference Proceeding Series*, Mar. 2011, pp. 117–128, doi: 10.1145/1951365.1951382.

[9] A. F. Pakpahan and I. S. Hwang, “Flexible access network multi-tenancy using NFV/SDN in TWDM-PON,” *IEEE Access*, vol. 11, pp. 42937–42948, 2023, doi: 10.1109/ACCESS.2023.3271142.

[10] H. Aljahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, “Multi-tenancy in cloud computing,” in *Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014*, Apr. 2014, pp. 344–351, doi: 10.1109/SOSE.2014.50.

[11] M. Kumar, A. Kishor, J. Abawajy, P. Agarwal, A. Singh, and A. Y. Zomaya, “ARPS: An autonomic resource provisioning and scheduling framework for cloud platforms,” *IEEE Transactions on Sustainable Computing*, vol. 7, no. 2, pp. 386–399, Apr. 2022, doi: 10.1109/TSUSC.2021.3110245.

[12] T. Kwok and A. Mohindra, “Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture*

*Notes in Bioinformatics)*, vol. 5364 LNCS, Springer Berlin Heidelberg, 2008, pp. 633–648, doi: 10.1007/978-3-540-89652-4_57.
[13] S. Fisher, "The architecture of the apex platform, salesforce.com's platform for building on-demand applications," in *Proceedings - International Conference on Software Engineering*, May 2007, p. 3, doi: 10.1109/ICSECOMPANION.2007.76.
[14] A. R. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, Feb. 2019, doi: 10.1016/j.future.2018.09.014.
[15] M. Kumar, K. Dubey, S. Singh, J. K. Samriya, and S. S. Gill, "Experimental performance analysis of cloud resource allocation framework using spider monkey optimization algorithm," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 2, Nov. 2023, doi: 10.1002/cpe.7469.
[16] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "HYRISE-A main memory hybrid storage engine," *Proceedings of the VLDB Endowment*, vol. 4, no. 2, pp. 105–116, Nov. 2010, doi: 10.14778/1921071.1921077.
[17] A. J. Elmore, D. Agrawal, S. Das, A. El Abbadi, A. Pucher, and X. Yan, "Characterizing tenant behavior for placement and crisis mitigation in multitenant DBMSs," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Jun. 2013, pp. 517–528, doi: 10.1145/2463676.2465308.
[18] Z. Liu, H. Hacigümüş, H. J. Moon, Y. Chi, and W. P. Hsiung, "PMAX: Tenant placement in multitenant databases for profit maximization," in *ACM International Conference Proceeding Series*, Mar. 2013, pp. 442–453, doi: 10.1145/2452376.2452428.
[19] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan, "Towards multi-tenant performance SLOs," in *Proceedings - International Conference on Data Engineering*, Apr. 2012, pp. 702–713, doi: 10.1109/ICDE.2012.101.
[20] R. Kallman *et al.*, "H-Store: A high-performance, distributed main memory transaction processing system," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1496–1499, Aug. 2008, doi: 10.14778/1454159.1454211.
[21] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *Proceedings - International Conference on Data Engineering*, Apr. 2011, pp. 195–206, doi: 10.1109/ICDE.2011.5767867.
[22] V. Narasayya and S. Chaudhuri, "Multi-tenant cloud data services: state-of-the-art, challenges and opportunities," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Jun. 2022, pp. 2465–2473, doi: 10.1145/3514221.3522566.
[23] C. P. Bezemer and A. Zaidman, "Multi-tenant SaaS applications: Maintenance dream or nightmare?," in *ACM International Conference Proceeding Series*, Sep. 2010, pp. 88–92, doi: 10.1145/1862372.1862393.
[24] F. Chen, A. Lu, H. Wu, and M. Li, "Compensation and pricing strategies in cloud service SLAs: Considering participants' risk attitudes and consumer quality perception," *Electronic Commerce Research and Applications*, vol. 56, Art. no. 101215, Nov. 2022, doi: 10.1016/j.elerap.2022.101215.
[25] R. Engel, S. Rajamoni, B. Chen, H. Ludwig, and A. Keller, "Ysla: Reusable and configurable SLAs for large-scale SLA management," in *Proceedings - 4th IEEE International Conference on Collaboration and Internet Computing, CIC 2018*, Oct. 2018, pp. 317–325, doi: 10.1109/CIC.2018.00050.
[26] A. Shah and N. Patel, "Efficient and scalable multitenant placement approach for in-memory database over supple architecture," *Computer Science and Information Technologies*, vol. 1, no. 2, pp. 39–46, Jul. 2020, doi: 10.11591/csit.v1i2.p39-46.
[27] F. Z. Belkadi and R. Esbai, "Model-driven engineering: from SQL relational database to column—oriented database in big data context," in *Smart Innovation, Systems and Technologies*, vol. 237, Springer Singapore, 2022, pp. 667–678, doi: 10.1007/978-981-16-3637-0_47.
[28] M. Kumar and S. C. Sharma, "PSO-based novel resource scheduling technique to improve QoS parameters in cloud computing," *Neural Computing and Applications*, vol. 32, no. 16, pp. 12103–12126, Jun. 2020, doi: 10.1007/s00521-019-04266-x.

## BIOGRAPHIES OF AUTHORS

**Arpita Shah** is a PhD student in the field of computer engineering at Charotar University of Science and Technology (CHARUSAT), Anand, Gujarat, India. She has received her master's degree in the field computer engineering from Sardar Patel University in 2009. Her major area of research includes cloud computing, database, distributed commuting and operating system. She can be contacted at email: arpitashah.ce@charusat.ac.in.

**Nikita Bhatt** obtained his PhD degree in the area of machine learning from Charotar University of Science and Technology (CHARUSAT), Gujarat, India in 2021 and master's degree in computer engineering in 2012 Gujarat, India. She is a professor at Department of Computer Engineering, Charotar University of Science and Technology (CHARUSAT), Anand, Gujarat, India. Her research interest includes data mining, machine learning, deep learning, information retrieval, meta learning, and active meta learning. She can be contacted at email: nikitabhatt.ce@charusat.ac.in.