

Optimizing multi-tenant database architecture for efficient software as a service delivery

Sanjeev Kumar Pippal¹, Sumit Kumar², Ruchi Rani³

¹GL Bajaj Institute of Technology and Management, Uttar Pradesh, India

²Symbiosis Institute of Technology, Pune Campus, Symbiosis International (Deemed University), Pune, India

³Department of Computer Science and Engineering, School of Computer Engineering and Technology, Dr.Vishwanath Karad MIT World Peace University, Pune, India

Article Info

Article history:

Received Jun 8, 2024

Revised Jul 15, 2024

Accepted Aug 5, 2024

Keywords:

Cloud

Memory efficient

Multi-tenant database design

Query performance

Software as a service

ABSTRACT

A multi-tenant database (MTDB) is the backbone for any cloud app that employs a software as a service (SaaS) delivery paradigm. Every cloud-based SaaS delivery strategy relies heavily on the architecture of multitenant databases. The hardware and performance costs for quicker query execution and space savings provided by the architecture of MTDBs are implementation costs. All tenants' data may be kept in a single table with a common schema and database format, making it the most cost-effective MTDB configuration. The arrangement becomes congested if tenants have varying storage needs. In this research, we present a space-saving architecture that improves transactional query execution while avoiding the waste of space due to different attribute needs. Extensible markup language (XML) and JavaScript object notation (JSON) compare the proposed system against the state of the art. The suggested multitenant database architecture reduces unnecessary space and improves query performance. The experimental findings show that the suggested system outperforms the state-of-the-art extension table method.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Sumit Kumar

Symbiosis Institute of Technology, Pune Campus, Symbiosis International (Deemed University)

Pune 412115, Maharashtra, India

Email: er.sumitkumar21@gmail.com

1. INTRODUCTION

Cloud computing [1] can provide software and hardware as a service using technologies such as virtualization and service-oriented architecture. Software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) are the three most common types of cloud service delivery (infrastructure as service). The core of any successful SaaS is multitenancy. It describes a highly customizable software solution that can be utilized by several companies (tenants) and is highly adaptable to the needs of each company and employee. Cost and efficiency benefits are realized by the cloud in the SaaS model thanks to the multitenant database. Cloud computing has several advantages over conventional applications. These include a "pay as you go" approach, more business agility, a lower total cost of ownership, a more fault-tolerant system, quicker startup time, and greater scalability. For a SaaS-based cloud service [2], a multi-tenant database (MTDB) [3] meeting the above criteria is essential. Cloud-compliant services for specialized applications like enterprise resource planning (ERP), customer relationship management (CRM), project management (PM), banking, and email need an MTDB that is both space-efficient and extremely query-efficient. Many tables would be nec-

essary to accommodate the various functional needs of the concerned tenants with a subset of shared aims. The aforementioned programme necessitates a MTDB to store its data. The cost of resources may be reduced, resource usage can be improved, and efficiency can be increased by designing databases to accommodate many users.

The amount of isolation is maximum due to the lack of resource sharing in the case of a separate database design method to implement multitenancy, whereby each tenant uses their database at a high expense for maintenance, procurement, and installation. However, with a shared database with a different schema architecture, the database is shared across distinct schema. Thus, there is still some isolation. This architecture is easy to create and very inexpensive. The cost savings from the shared database and shared schema model of multitenancy are especially noticeable since the sharing occurs at the schema level, allowing many tenants to use a single database server instance. Moreover, it reduces the cost of hardware and software while offering the same degree of security, but software may be used to establish isolation.

Section 1 represents a brief introduction, and section 2 iterates through the short literature review report with different types of tables proposed in the literature. Section 3 outlines the proposed method and design. Section 4 analyzes and discusses performance evaluation. Finally, section 5 is the conclusion, briefly summarizing the proposed model.

2. LITERATURE REVIEW

Follows are some of MTDB's features that were used to showcase the case study:

2.1. Private table

There is a separate table for each renter. According to Aulbach *et al.* [4], the method used by private tables makes it possible for each renter to have private tables. While all three tenants in Figure 1 are businesses, they all have unique needs. As a result, all tables serve distinct purposes yet are kept together to accommodate tenants' requirements. The private table approach of MTDB ensures that each tenant has their unique schema, eliminating the need for any data exchange and allowing for a high meta-data/data ratio and minimal buffer use.

Account No. 15			
Account ID	Name	Hotel	Rooms
1	KUMAR	TAJ	130
2	SUMIT	CROWN	1000

Account No. 40		
Account ID	Name	Agents
1	MINO	63
2	BOSH	20

Account No. 30	
Account ID	Name
1	BAT
2	MAT

Figure 1. Private table

2.2. Universal table layout

In this method, which universal relation developed [5], [6], each column in a table is stored in its separate row. The table is structured with a tenant column, a table column, and a fixed number of generic data columns (e.g., 250). The idea for this method was first presented in a study by Maier and Ullman in 1983. The primary benefit of this method is that it eliminates the need to generate fresh DDL when implementing new changes. Unfortunately, indexing is not an option with these generic columns. Another drawback is that this database will have many rows with empty data. The database management system (DBMS) needs to cope with this well. The design needs to be revised due to including numerous NULL values, which are not type-safe and need casting, and the need for indexing capability.

2.3. Extension tables

Extension tables are a hybrid of the basic table and private table designs. Tables are shared across tenants. However, tenants with unique data needs may create extension tables. In addition, it is possible to share extension tables. Separate tables for extensions are now available. It has a better design than the private table structure. However, the number of tables still rises as the number of tenants increases because of the need for an additional join at runtime and the row column needed for rebuilding the row. There are fewer tables in this configuration than in the private table layout. Extension table illustrates how the hospitality *ACCOUNT* and sports *ACCOUNT* tables, which hold the relevant data for each tenant, are linked through the tenant id attribute to accommodate the various needs of each tenant. Since row and column are both utilized to retrieve records, decomposing extension tables into several tables necessitates an additional join at runtime. It provides a more accurate depiction than the private table arrangement, and the number of tables may increase proportionally as the number of renters does.

2.4. Pivot tables

Each attribute value in the logical table has its row, making this a generic and type-safe multitenant database architecture. Because of how it was made, the need to deal with NULL values was removed entirely. A pivot table is where each field's value is kept in its row. Each data type has a specific pivot table, such as "Pivot int" and "Pivot string". There is an improvement over UTL in that it requires actual typing instead of just a table. Thus, valuable indexes may be made for these tables. A pivot table is built for only one column [7].

2.5. Chunk folding

Since it is based on pivot tables, the general design works best when the dataset can be split into dense subgroups. If highly populated subsets can be removed, reconstruction will need fewer joins. The fact that it can be indexed is a bonus. This approach, dependent on chunk size, also reduces the meta-data/data ratio. In the chunk folding method [8], these chunks are folded in different tables and combined.

A single chunk table contains all extensions, while the accounts table is the primary data storage, as shown in Figure 2. An integral part of the tables is a "Tenantid" property that links each row to its corresponding tenant. In all, four different fields may be manipulated in the AccountROW table. Data values of type integer or string may be persisted for individual logical columns in the logical table using the "number" and "string" attributes, respectively. Chunk folding needs to consider the need for extensibility in programmes like CRM and ERP.

Account ^{ROW}			
Tenant ID	Row Id	Account ID	Name
15	0	1	KUMAR
15	1	2	SUMIT
30	0	1	MINO
30	1	2	BOSH
40	0	1	BAT
40	1	2	MAT

Chunk ^{ROW}					
Tenant ID	Table	Chunk	Row ID	Number	String
15	0	0	0	130	TAJ
15	0	0	1	1000	CROWN
40	2	0	0	63	--
40	2	0	1	20	--

Figure 2. Chunk folding

2.6. Multi-tenant shared table

This method of multitenant database architecture [9] divides the storage needs of features common to all tenants. In a single schema application, just one schema explores the difficulty of scaling a multi-tenant shared table. Jacobs suggested methods like the shared machine, process, and table. The multitenant design presents several difficulties discussed in [10], [11]. Application-level security for multimedia applications is detailed in [12]. For example, Kerberos [13], [14] may be used for security, and mutual authentication trust can be established using the methods described in [15], [16].

2.7. Improved extension table

According to Chong *et al.* [17], the most popular industrial MTDB is an enhanced form of an extension table. The system holds each tenant's data in its table, and the metadata associated with that table is stored in a different database. Figure 3 shows an improved extension table.

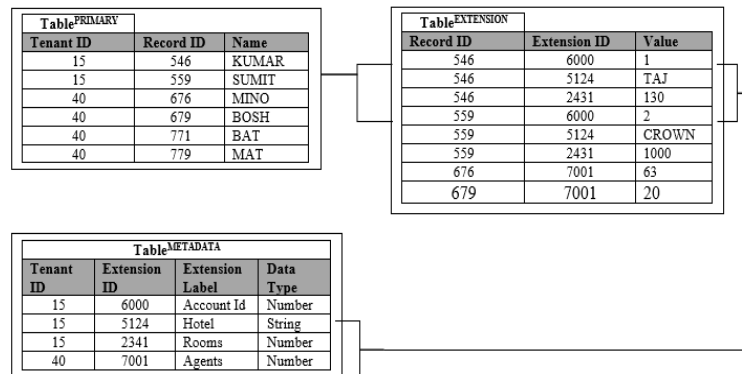


Figure 3. Improved extension table

Most hosted services employ query transformation to transfer several logical tenant schemas in the application to a single multi-tenant physical schema in the database, using features such as tenant ID. Limits may be set on the total database queries (80-100 user sessions). This strategy's scalability is restricted by the number of tables that can be stored in the database, which is constrained by the amount of accessible main memory. Because each table in MYSQL is allotted 8 KB of RAM, 800 MB of storage space is needed to store 1,000,000 tables. Additionally, there is intense competition for the remaining cache space since buffer pool pages are allotted individually for each table. Therefore, once a database server has more than about 50,000 tables, performance tends to suffer. Reducing the number of tables should be a top focus to minimize performance issues.

Any software delivered as a service must have highly available databases [18]. SaaS applications use multitenant database systems as their backend has grown exponentially [19]. The MTDB system is accessible through the SaaS distribution mechanism. MTDB architectures provide CRM, supplier relationship management (SRM), and business intelligence (BI). Elastic extension table (EET) is a MTDB architecture, and Yaish and Goyal [20] presented a framework with an intermediary database layer allowing software applications and relational database management systems (RDBMS) to obtain and store data in EET. The suggested design needs a thorough evaluation of its performance. Matthew *et al.* [21] discuss the need for MTD databases, the difficulties of deploying MTD, the need for database migration to achieve MTD's flexibility, and the many elements that go into deciding the MTD models to use. MTDB, difficulties in implementing MTD, database migration for flexibility in MTD, and variables impacting the selection of MTD models are all identified and summarised in [21]. Elmore *et al.* [22] imagine a DBMS in which multitenancy is viewed as virtualization and where models of database multitenancy are analyzed from the perspectives of availability and scalability, with further standardization of the forms of operation of MTDB in the cloud for an autonomous, elastic, and manageable multitenant database. Data security, isolation, query performance, and response time speed are only some challenges addressed in the novel MTDB schema architecture suggested in [23]. The suggested technique has the drawback of sacrificing one aspect of performance for the other: storage capacity.

Yaish *et al.* [24] have expanded their previous work on EET to show how the tables of tenants may be modified to meet the needs of specific users. Significant performance gains were seen when compared to Salesforce.com's commercial MTDB. Curino *et al.* [25] outline three main challenges that must be overcome before companies can effectively outsource their database software and management. This outsourcing approach should appeal to many users and be cost-effective for the service providers. They introduce a new way of handling databases called "database-as-a-service" (DBaaS), which ensures that everything from setting up and scaling the database to managing its performance, backups, privacy, and who can access it is handled by the service provider. To make this new approach even better than existing methods, the DBaaS model focuses on handling different types of tasks that the database needs to handle. It also allows users to run SQL queries

on encrypted data, meaning sensitive information stays secure even while processing. Additionally, it uses a smart way of dividing the data, called graph-based data partitioning, to ensure the database can handle many tasks without slowing down. To guarantee data integrity with privacy and performance isolation in multi-tenant blockchain-based systems, Weber *et al.* [26] have developed a scalable platform design. The suggested architecture provides each tenant with their authorized blockchain, where they may store their data and run smart contracts independently. Data integrity, performance isolation, data privacy, configuration flexibility, availability, cost-effectiveness, and scalability are all attained by the suggested architecture, as shown by the assessment results. Ma and Yang [27] analyze the characteristics of multitenant data and suggest multitenant multiple wide tables that are vertically scalable. This layout strikes an ideal middle ground between tenant preferences and overall efficiency. The experimental findings show that in terms of spatial intensity and read performance, multiple-wide tables with vertical scalability are superior to both single-wide tables and single-wide tables with vertical scalability. According to Song *et al.* [28], there are two ways to personalize a multi-tenant SaaS using microservices: invasive and non-intrusive. With its ability to strike a good mix between independence, integration, and economies of scale, the microservice-based customization solution shows promise for meeting general customization needs. Using foundation DB, Chrysafis *et al.* [29] suggest record layer is a free and open-source framework that implements a relational database-like data store in the form of a record-oriented data layer on top of foundation DB. They also show how Apple's cloud backend service, cloud kit, uses the record layer to provide apps that serve hundreds of millions of people with a strong abstraction layer. The aim of [30] is to highlight the trade-offs, resemblances, and distinctions that need to be considered when determining and implementing the required level of separation between tenants. This study evaluates different levels of tenant isolation through a cross-case analysis of selected open-source cloud-hosted software engineering tools. Our findings reveal that factors such as minimal client latency and bandwidth during the transfer of multiple files between repositories, as well as the substantial size of generated data, intensive resource usage by specific software operations, and varying workload levels, can diminish the extent of isolation. They also imply that software architects should consider the compromises, similarities, and differences we highlight to meet the tenants' demands for privacy. Examples of several approaches to database architecture for the Account tables of three tenants with identifiers 15, 30, and 40. Tenant 15 has a hotel-related extension, tenant 30 has a sports-related extension, and tenant 40 has an expansion for agents. Follows are some of MTDB's features that were used to showcase the case study:

3. METHOD

The suggested method, seen in Figure 4, involves using a single table to hold all characteristics, an extension of the shared database common schema approach. As illustrated in Figure 4, XML/JSON fields are used as table attributes to handle the varied needs, and these fields contain all tenancy-specific attributes given as XML/JSON tagged objects. All 4th-generation programming languages come equipped with XML/JSON file parsers, such as simple API for XML (SAXn), document object model (DOM), or JSON parsers, making reading and processing these files simple. Space savings are built into the MTDB for cloud applications under the SaaS delivery paradigm since the number of required characteristics decreases as the number of unique attributes across tenants decreases. The design also improves the basic recommended approach's query performance significantly. All queries, except select ones that need parsing, benefit from the suggested layout. Our implementation of a caching system for select queries, shown schematically in Figure 5 allows us to boost speed significantly.

Table ^{PROPOSED}				
Tenant ID	Account ID	Name	JSON/XML	
			Attribute	or File
15	1	KUMAR	<Hotel>TAJ</Hotel><Rooms>130</Rooms>	15.1.xml
15	2	SUMIT	<Hotel>CROWN</Hotel><Rooms>1000</Rooms>	15.2.xml
30	1	MINO	<Agents>63</Agents>	30.1.xml
30	2	BOSH	<Agents>20</Agents>	30.2.xml
40	1	BAT	--	40.1.xml
40	2	MAT	--	40.2.xml

Figure 4. Proposed XML table

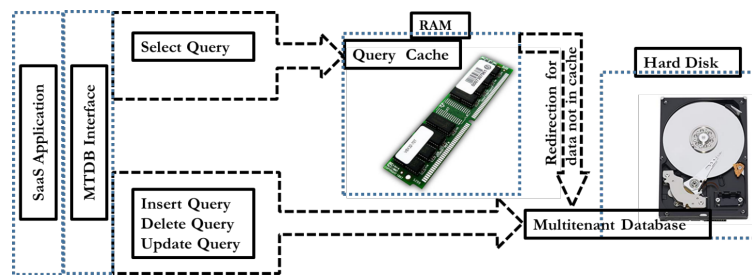


Figure 5. Query processing framework

3.1. Method

The test environment uses an ERP-CRM-hosted service. Customer requests are made through their browsers. The service is used to benchmark MTDB designs and is described as a standard MTDB for SaaS cloud distribution. The following permutations have been implemented:

- Using XML or JSON for the XML attribute
- Using XML or JSON file as a link in the attribute

We've built up an experiment with 10 nodes, all hosted on the cloud. All of the data at these vertices reflects queries made by database users. The use of resources by these nodes is seen in Figure 6. In the configuration, we keep track of how much space each node takes up on its memory, CPU, network, and disc.

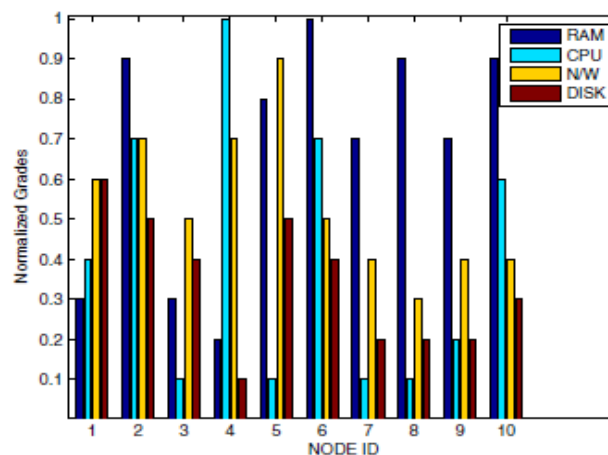


Figure 6. Testbed configuration

3.2. Taking into account the extension table method

Using the setup mentioned above, we evaluate the efficiency of our proposed multitenant-based database queries to that of an extension table-based method. XML and JSON implementations, with and without application-level caching, are compared for speed. Both methods' efficiencies are evaluated in metrics like the number of general SQL queries run per unit time (i.e., insertion, deletion, updation, and selection). Finally, we compare the two methods' capacity for storing newly added tenant-specific attributes, measuring it against the average query execution time for queries linked to adding attributes to the database.

Figure 7 shows the results of running some sample SQL queries using both methods. Compared to the current method, the chart clearly shows that the suggested method executes 318 (using JSON) and 124 (using XML) SQL insertion queries faster. Additionally, more than 750 (JSON) and 564 (XML) SQL deletion queries are conducted using the suggested technique instead of the extension table strategy. Similarly, update queries show a performance boost over an extension table-based strategy. The proposed approach provides faster query execution rates than the extension table-based approach, but the extension table-based approach outperforms the proposed approach in SQL selection queries. However, nearly the same performance is achieved when both approaches are used with caching. The algorithm for selecting caching is shown in Algorithm 1.

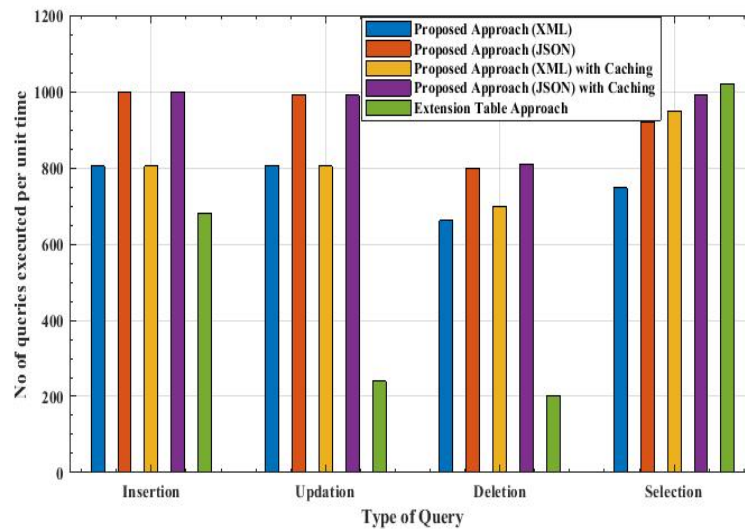


Figure 7. Query performances

Algorithm 1: Algorithm for selecting caching

Input: query_type, query_txt
Output: Query Execution

```

1  switch query_type do
2      case 1 do
3          | Insertion
4      end
5      Query Execution for insertion Query;
6          1. User supplied information is inserted for all common attributes between the tenants in a conventional way.
7          2. XML/JSON attribute: For the XML or JSON attribute, an XML/JSON file is formed with user-supplied information.
8          3. All user-supplied information is tagged with appropriate tags and inserted further.
9
10     case 2 do
11         | Updation
12     end
13     Query Execution for updation Query;
14         1. User supplied information is inserted for all common attributes between the tenants in a conventional way.
15         2. XML/JSON attribute: For the XML or JSON attribute, the attribute to be updated is fetched from XML/JSON attribute using its tag.
16         3. After modification to tagged value, the complete modified XML file is pushed back into the database.
17         3. Cached data is invalidated and marked dirty.
18
19     case 3 do
20         | Deletion
21     end
22     Query Execution for Deletion Query;
23     Information is deleted for all common attributes between the tenants as well as XML or JSON attributes in a conventional way.
24     case 4 do
25         | Selection
26     end
27     Query Execution for Selection Query without caching;
28         1. Information is retrieved for all common attributes between the tenants in a conventional way.
29         2. Information stored in the XML or JSON attribute is parsed with SAX, DOM, or JSON parser.
30     Query Execution for Selection Query with caching;
31         1. A cache is checked for a non-dirty copy of information, and if not present in the cache, the below steps are followed.
32         2. Information is retrieved for all common attributes between the tenants conventionally.
33         3. Information stored in the XML or JSON attribute is parsed with SAX, DOM, or JSON parser.
34 end

```

4. PERFORMANCE EVALUATION

To evaluate how well the two approaches work in light of adding new characteristics to the database, the execution time of equivalent queries in both approaches is tracked—no more than 15. Figure 8 displays the outcomes of these two approaches. The graph reveals that our recommended approach finishes each query with 75 additional attributes in around 5.382 ms. In contrast, the average query execution time for the extension table-based approach rises to 20.708 ms.

More research is needed to estimate the space needed in the database to accommodate the tenant-specific new properties. Here, we track how much room is required for new features in the table-based techniques we propose and develop. Figure 8 demonstrates that the proposed approach can accommodate far more data than the extension table approach, meaning there is no typical growth in storage needs as the number of attributes grows. The proposed method uses XML to store tenant-specific attributes, allowing for more attributes to be added with minimal storage size (which is further improved by using JSON). In contrast, the extension-based method requires the creation of a linked and mapped table, which increases the storage space required for the newly added attributes. Figure 9 shows how the suggested method uses much less space than the extension table-based alternative to store the same amount of data. As seen in Figure 10, the suggested method reduces the time it takes to run several queries simultaneously compared to the extension table method.

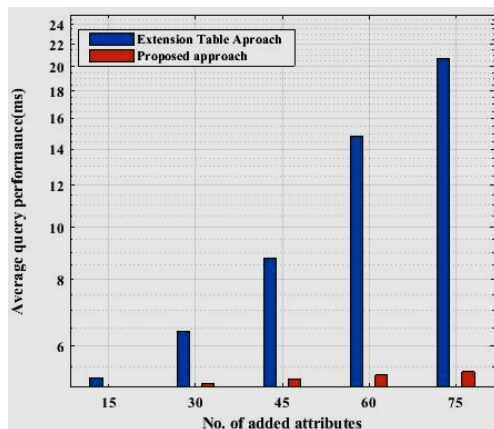


Figure 8. Performance for added attributes

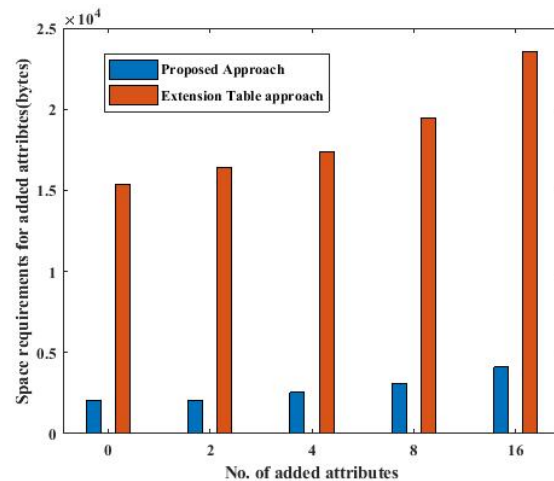


Figure 9. Storage requirements for added attributes

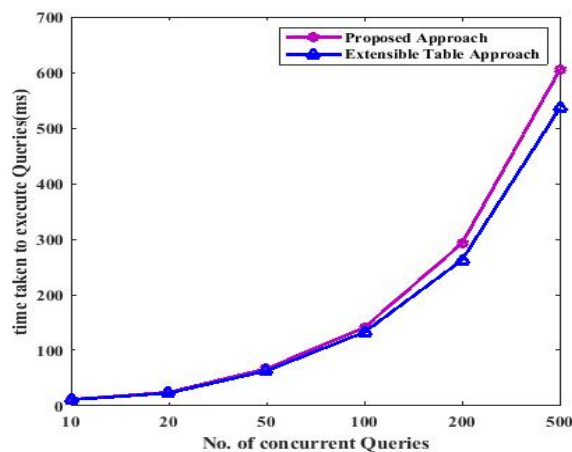


Figure 10. Concurrent query execution




5. CONCLUSION

In the proposed system, the MTDB is the back end. XML field attributes store several attributes in a single field as an XML/JSON file, reducing storage needs while increasing transactional efficiency. The experimental results demonstrate that the proposed method not only achieves efficient query execution per unit time but also needs significantly less query execution time for added attributes and can accommodate a greater number of attributes with a correspondingly smaller increase in storage space. The suggested method consistently outperforms the state-of-the-art in all experiments. The design's query speed and ability to save storage space are strong suits.




REFERENCES

- [1] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST*, Sep. 2011, doi: 10.6028/nist.sp.800-145.
- [2] M. Hui, D. Jiang, G. Li, and Y. Zhou, "Supporting Database Applications as a Service," *2009 IEEE 25th International Conference on Data Engineering*, Shanghai, China, 2009, pp. 832-843, doi: 10.1109/ICDE.2009.82.
- [3] D. Jacobs and S. Aulbach, "Ruminations on Multi-Tenant Databases," *Datenbanksysteme in Business, Technologie und Web (BTW 2007)*, 12. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Aachen, Germany, Proceedings, 2007, pp 514-521.
- [4] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques," *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1195-1206, doi: 10.1145/1376616.137673.
- [5] H. Hacigumus, B. Iyer and S. Mehrotra, "Providing database as a service," *Proceedings 18th International Conference on Data Engineering*, San Jose, CA, USA, 2002, pp. 29-38, doi: 10.1109/ICDE.2002.994695.
- [6] D. Maier and J. D. Ullman, "Maximal objects and the semantics of universal relation databases," *ACM Transactions on Database Systems (TODS)*, vol. 8, no. 1, pp. 1 - 14, 1983, doi: 10.1145/319830.319831.
- [7] E. J. Domingo, J. T. Nino, A. L. Lemos, M. L. Lemos, R. C. Palacios and J. M. G. Berbís, "CLOUDIO: A Cloud Computing-Oriented Multi-tenant Architecture for Business Information Systems," *2010 IEEE 3rd International Conference on Cloud Computing*, Miami, FL, USA, 2010, pp. 532-533, doi: 10.1109/CLOUD.2010.88.
- [8] G. P. Copeland and S. N. Khoshafian, "A decomposition storage model," *ACM SIGMOD Record*, vol. 14, no. 4, pp. 268-279, 1985, doi: 10.1145/971699.318923.
- [9] M. Grund, M. Schapranow, J. Krueger, J. Schaffner, and A. Bog, "Shared Table Access Pattern Analysis for Multi-Tenant Applications," *2008 IEEE Symposium on Advanced Management of Information for Globalized Enterprises (AMIGE)*, Tianjin, China, 2008, pp. 1-5, doi: 10.1109/AMIGE.2008.ECP.37.
- [10] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An, "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing," *2008 IEEE International Conference on e-Business Engineering*, Xi'an, China, 2008, pp. 94-101, doi: 10.1109/ICEBE.2008.60.
- [11] C. -P. Bezemer and A. Zaidman, "Challenges of Reengineering into Multi-Tenant SaaS Applications", Report TUD-SERG-2010-012, Delft University of Technology Software Engineering Research Group Technical Report Series, pp. 1-12, 2010.
- [12] P. C. Chapin, C. Skalka, and X. S. Wang, "Authorization in trust Management: Features and Foundations," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, pp. 1 - 48, 2008, doi: 10.1145/1380584.1380587.
- [13] Neuman C. RFC 1510, "The Kerberos network authentication service (V5)," 1993.
- [14] P. L. Hellewell, T. W. van der Horst and K. E. Seamons, "Extensible Pre-authentication Kerberos," *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, Miami Beach, FL, USA, 2007, pp. 201-210, doi: 10.1109/ACSAC.2007.33.
- [15] C. Lin and V. Varadharajan, "Trust based risk management for distributed system security - a new approach," *First International Conference on Availability, Reliability and Security (ARES'06)*, Vienna, Austria, 2006, pp. 8-13, doi: 10.1109/ARES.2006.139.
- [16] P. Liu, R. Zong, and S. Liu, "A New Model for Authentication and Authorization across Heterogeneous Trust-Domain," *2008 International Conference on Computer Science and Software Engineering*, Wuhan, China, 2008, pp. 789-792, doi: 10.1109/CSSE.2008.1152.
- [17] F. Chong, G. Carraro, R. Wolter. Multi-tenant data architecture. Available online: <http://msdn.microsoft.com/en-us/library/aa479086.aspx>. (Date Accessed 6 March, 2020).
- [18] S. Pippal, S. Singh, R. K. Sachan, and D. S. Kushwaha, "High availability of databases for cloud," *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2015, pp. 1716-1722.
- [19] S. K. Pippal and D. S. Kushwaha, "A simple, adaptable and efficient heterogeneous multi-tenant database architecture for ad hoc cloud," *Journal of Cloud Computing: Advances, Systems and Applications*, 2013, doi: 10.1186/2192-113X-2-5.
- [20] H. Yaish and M. Goyal, "A Multi-tenant Database Architecture Design for Software Applications," *2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, NSW, Australia*, 2013, pp. 933-940, doi: 10.1109/CSE.2013.139.
- [21] O. Matthew, C. Dudley, and R. Moreton, "A review of multi-tenant database and factors that influence its adoption," *UK Academy for Information Systems Conference Proceedings 2014*, 2014.
- [22] A. J. Elmore, S. Das, D. Agrawal, and Amr El Abbadi, "Towards an Elastic and Autonomic Multitenant Database," *Proceedings of the Network Database Workshop*, 2011.
- [23] A. I. Saleh, M. A. Fouad, and M. Abu-Elkheir, "A Hybrid Multi-Tenant Database Schema for MultiLevel Quality of Service," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 5, no. 11, pp. 132-139, 2014.
- [24] H. Yaish, M. Goyal, and G. Feuerlicht, "Evaluating the Performance of Multi-tenant Elastic Extension Tables," *Procedia Computer Science*, vol. 29, pp. 614-626, 2014, doi: 10.1016/j.procs.2014.05.055.
- [25] C. A. Curino *et al.*, "Relational Cloud: A Database-as-a-Service for the Cloud," *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA, 2011, 235-240.
- [26] I. Weber, Q. Lu, A. B. Tran, A. Deshmukh, M. Gorski, and M. Strazds, "A Platform Architecture for Multi-Tenant Blockchain-Based Systems," *2019 IEEE International Conference on Software Architecture (ICSA)*, Hamburg, Germany, 2019, pp. 101-110, doi: 10.1109/ICSA.2019.00019
- [27] K. Ma and B. Yang, "Multiple Wide Tables with Vertical Scalability in Multitenant Sensor Cloud Systems," *International Journal of Distributed Sensor Networks*, vol. 2014, pp. 1-10, 2014, doi: 10.1155/2014/583686.
- [28] H. Song, P. H. Nguyen, and F. Chauvel, "Using Microservices to Customize Multi-Tenant SaaS: From Intrusive to Non-Intrusive," *Joint Post-proceedings of the First and Second International Conference on Microservices (Microservices 2017/2019)*, 2019, pp. 1:1-1:18, doi: 10.4230/OASfcs.Microservices.2017-2019.1.
- [29] C. Chrysafis *et al.*, "FoundationDB Record Layer: A Multi-Tenant Structured Datastore," *SIGMOD '19: Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1787-1802, doi: 10.1145/3299869.3314039.
- [30] L. C. Ochei, J. M. Bass, and A. Petrovski, "Degrees of tenant isolation for cloud-hosted software services: a cross-case analysis," *Journal of Cloud Computing*, 2018, doi: 10.1186/s13677-018-0121-8.




BIOGRAPHIES OF AUTHORS

Sanjeev Kumar Pippal    received B.Tech degree from MJP Rohilkhand University, M.Tech, and Ph.D. from MNNIT Allahabad. His area of interest is cloud computing, distributed computing, and blockchain. He is certified in machine learning and deep learning. He has published over 30 research papers in SCI/ Scopus international journals and conferences. He has filed four patents and published 02 patents. He can be contacted at email: Sanpippalin@gmail.com.



Sumit Kumar    received a Bachelor's degree in Electronics and Telecommunication from Kurukshetra University, Kurukshetra, India in 2005, the Master's degree from Guru Jambheshwar University of Science and Technology, Haryana, India in 2008, and a Ph.D. degree from Jamia Millia Islamia, Delhi, India in 2017. He works as a Professor at the Electronics and Telecommunication Department of Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, Maharashtra, India. His research areas are machine learning and deep learning. He can be contacted at email: er.sumitkumar21@gmail.com.



Ruchi Rani    received a Bachelor's degree in Computer Science Engineering from Kurukshetra University, Kurukshetra, India in 2008, the Master's degree from Maharshi Dayanand University, Haryana, India in 2012. She is currently pursuing a Ph.D. degree from the Department of Computer Science Engineering, Indian Institute of Information Technology, Kottayam, Kerala, and working as an Assistant professor at the Department of Computer Engineering and Technology, School of Computer Engineering and Technology, Dr.Vishwanath Karad MIT World Peace University, Pune 411038, Maharashtra, India. Her research interests include machine learning and deep learning. She can be contacted at email: ruchiasija20@gmail.com.