International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

TRANSFORMING ACTIVE-ACTIVE ARCHITECTURES: LEVERAGING MVCC FOR SAAS SCALABILITY AND CONSISTENCY

Nikhil Sagar Miriyala[1*]
[1]Senior Software Engineer, Oracle America., USA
*Corresponding Author: nmiriya7@gmail.com

Abstract:

With the state at which modern highly complex distributed systems and global SaaS platforms are, maintaining strong consistency and scalability when running the applications in active-active architectural patterns comes with significant challenges. Active-active systems, which distribute traffic among various nodes across multiple regions, while help with maintaining high availability, face difficulties to ensure data consistency, handle high levels of concurrency, and stay within real-time Service Level Agreements (SLAs). For systems dealing with data persistence use cases, which is the case with majority of the large-scale SaaS platforms, traditional database methods like two-phase commit (2PC) and optimistic locking mechanisms often fail to meet the expected performance at scale, resulting in bottlenecks and delays when dealing with large volumes of incoming traffic, especially in real-time systems. To improve these scenarios, this paper explores how Multi-Version Concurrency Control (MVCC) databases present an efficient solution to these issues by allowing non-blocking, concurrent writes while preserving transactional consistency among geographically distributed nodes of the database. MVCC ensures that records undergoing constant state changes, such as those found in financial transactions and e-commerce platforms, can be managed without conflicts, even when numerous users or processes are interacting with the same data simultaneously. Additionally, the article aims to discuss the unique benefits that MVCC databases can offer towards active-active architectures, such as enhanced throughput, dynamic scaling, and improved conflict resolution in real-time scenarios. It also discusses how MVCC can enhance complex workflows found in multi-tenancy systems such as ticketing platforms, while maintaining strong consistency and high availability.

Keywords: Active-Active Architecture, MVCC (Multi-Version Concurrency Control), Global SaaS Platforms, Transactional Consistency, Workflow Management

1. Introduction:

Software-as-a-Service (SaaS) platforms have become an essential part of modern business operations, providing scalable, cost-efficient, and easily accessible applications hosted on the cloud [1]. These platforms are able to serve a wide variety of businesses, from providing customer support tools to financial applications, allowing companies the flexibility to use critical day-to-day software without the need for significant in-house development and on-site infrastructure.

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

All SaaS offerings typically try to achieve the fundamental goals, which include high-scalability, high-availability, and multi-tenancy. SaaS solutions are built to expand horizontally, support an increasing number of users, and manage growing demand by distributing workloads across various servers or geographic regions [2]. Furthermore, they are designed to be highly resilient, ensuring that systems can handle failures smoothly and provide uninterrupted access to their services.

Certain SaaS products, particularly those related to financial transactions or customer support systems, require a higher level of resiliency and availability because of their business models [3]. For instances, applications like payment processing systems or ticketing platforms function in a manner where a significant downtime or discrepancy could result in severe business and/or legal repercussions. For such use cases, in addition to being highly available across multiple regions simultaneously, the systems should be highly resilient, to ensure quick mitigation in the event of a failure. Active-active configurations, i.e. multiple regions operating concurrently, are essential for maintaining service continuity and reducing downtime, although they come with increased complexity in ensuring such high consistency across distributed systems.

Conventional databases that support active-active architectures have historically faced challenges with respect to finding a balance between maintaining strong consistency with being highly available and scalable [4]. As a trade-off, real-time transactional systems have fallen back to active-passive architectures, where one node in the primary region handles all write operations while all the other nodes with the same region serve only as read replicas, with the secondary region being synchronized on a timely basis for disaster recovery procedures . While this approach can provide strong consistency and quick disaster recovery, it eventually effects the performance of the system with increasing demand over time. The other approach to achieve better performance and scalability, is to use eventually consistent architectures, in which case the updates performed might not be immediately visible across all nodes across multiple regions. While this method can enhance performance, it compromises consistency, which is often unacceptable, particularly in critical systems such as payment gateways [5].

The recent growth of Multi-Version Concurrency Control (MVCC) databases has provided a solution to the limitations of traditional distributed database systems. MVCC database provides strong consistency across multiple regions while preserving high availability and scalability. By allowing non-blocking concurrent writes and ensuring that each transaction functions on its own version of the data, MVCC overcomes many of the issues related to managing concurrency and consistency in active-active setups [6]. This paper will delve into how MVCC databases are transforming SaaS platforms, allowing businesses to satisfy the requirements of multi-region consistency, dynamic rule management, and real-time system monitoring, and potentially creating a new trend for designing highly reliable and scalable applications.

2. Baseline Architecture & Limitations

Before we explore how MVCC databases can enhance SaaS products, it is essential to analyze the baseline architecture of a typical SaaS platform deployed in a single region. This serves as

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

the foundation for understanding how the system operates without the additional complexities that usually occur, once we move to a multi-region distributed architecture.

2.1 High-Level Overview of a Single-Region Architecture

At a high level, a standard SaaS platform typically consists of a web application, a bunch of microservices, and a transactional database that collaboratively provide the expected services to multiple customers. The platform supports multiple tenants and is designed to ensure scalability, availability, and data consistency while accommodating customer-specific features and customizations [7]. Below is a quick summary of what each component is supposed to deliver:

1. Web Application Layer (UI Layer): The web application acts as the interface that users engage with, where tenants can log in and use the SaaS product. Each tenant accesses a distinct interface that is driven by tenant-specific business rules and configurations.
    a. Admin Access: For each tenant, an Admin page is provided with restricted access, using specific RBAC (Role-Based Access Control) policies, which can be used for customization, in order to set up tenant-specific rules, business logic, and UI configurations. Admins have the ability to modify certain elements such as pricing structures, workflow rules, feature toggles, etc., depending on the product.
    b. End-User Access: Regular users (the tenants' customers) engage with the product interface, which provides tenant-specific custom features that the product is developed for.
2. Control Plane: The control plane maintains the backend for managing all customizations and tenant settings.
    a. It provides configurations specific to each tenant, including pricing models, feature flags, UI designs, and business logic.
    b. For each update performed by the admin users, for example, activating a feature, the control plane validates and saves these updates to the central database.
3. Data Plane: The data plane is responsible for handling each incoming request based on the business logic.
    a. It interacts with the central database to read and write data, controlled using the configurations provided through the control plane.
    b. The services also handle asynchronous background tasks, for example, email notifications, using Kafka or similar streaming services.
4. Transactional Database Layer:
    a. The database stores operational data for each request processed, along with rules configured through the control plane.
    b. The database guarantees strong consistency across all operations, making sure that all transactions are executed without any discrepancies.

2.2 Example Use Case: E-Commerce SaaS Platform

To understand the architecture better, let’s take an e-commerce platform as an example and see what each component is responsible for:

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

1. Web Application: The tenant, the merchant in this case, uses the admin UI to set up their pricing structures, discount rules, product catalog, and payment gateway endpoints. And, the end-users, shoppers in the case, use the customized UI, where they browse through the products and place their orders. Additional enhancements include features such as a search window, a wishlist screen, etc.
2. Control Plane: The control plane takes care of the processing and persisting settings, which includes, pricing models, feature flags such as holiday promotions, and brand specific settings, such as logos, colors, and theme.
3. Data Plane: The data plane handles requests such as listing products, providing product specific information and handle order placements and can grow based on the additional features configured. It also takes care of inventory updates and invokes any asynchronous tasks like sending a confirmation email, order updates, etc.
4. Database: The database stores all the data, which includes tenant specific configurations such as pricing models, discount rules, etc., and operational data such as orders, product inventory, etc.

2.3 Multi-Region Distributed Architecture and its Limitations

The need for a multi-region deployment comes from the requirement to be highly available and efficient with increasing traffic growth over time. By deploying services and data across multiple regions, SaaS platforms can ensure low-latency response times for users across the globe and increase overall resiliency. However, switching to a multi-region architecture introduces new issues with respect to data consistency, disaster recovery, and conflict resolution in distributed systems [8]. Below are two standard architectures used for multi-region deployments.

2.3.1 Active-Passive Architecture with Primary and Standby Regions

An active-passive architecture is designed such that one of the two regions acts as the primary region for handling all the incoming operations and the other regions acts as a standby, which does not handle any traffic until a region failover is performed as part of a disaster recovery protocol. This setup is comparatively simple, but still faces the following challenges:

1. Disaster Recovery: While failover is secondary region is how this architecture primarily works, it is possible that the standby region may not be fully up to date with the primary region. The replication lag between these two regions can lead to data loss or potential inconsistencies when failover happens. A different approach would be to synchronously replicate data to the secondary region, but that comes at a cost of increased overall latency for each incoming request.
2. Recovery Time Objective (RTO): The time taken to failover from the primary to the standby region is very critical and needs thorough testing and strict protocols setup in prior [9]. Any delay in this process can lead to service interruptions and SLA breaches.
3. Recovery Point Objective (RPO): Even with synchronous replication, there is potential for some data loss, if the standby region is not properly synced during failover.

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

2.3.2 Active-Active Architecture with Global Database

The other architectural pattern is to use an active-active setup, in which case both the regions handle incoming requests, using a global database in the background, through which both read and write operations can be performed from both the regions. While this approach helps with achieving high availability, it comes with its own challenges:

1. Eventual Consistency: Replication of data from the master node to the read replicas in an asynchronous fashion can lead to stale data, resulting in discrepancies.
2. Synchronous Replication Overhead: The alternative approach to fix the issue mentioned above is to instead perform synchronous replication, which makes sure that data is replicated to at least one node in the other region before a transaction is marked as committed. However, synchronous replication introduces additional latency overhead, minimizing system performance.
3. Data Conflicts: In a scenario where parallel write calls are performed for the same records from both regions, data conflicts may occur. In order to resolve these conflicts without any data loss, advanced conflict resolution or optimistic locking mechanisms are required, which could also affect the latency.

3. Mutli-Region Architecture using MVCC

3.1 MVCC Databases and Core Principles

Multi-Version Concurrency Control (MVCC) is a database management technique developed to handle concurrent read and write operations without the need for external conflict management or locking mechanism [10]. It is highly beneficial in scenarios where high concurrency is required, where multiple users or applications interact with the database at the same time, and comparatively provides better throughput and scalability.

3.1.1 Core Principles

1. Versioning: The most important aspect of MVCC databases is it use of versioning. Instead of locking the records during write operations, the database stores multiple versions of each record [11]. And for each transaction, the database works such that every node is aware of the latest version of the record and does not interfere with other transactions, handled by other nodes at the same time. For each write operation, a new version of the record is created, instead of overwriting the existing one. This helps with concurrent read and write operations without blocking, since each transaction can access the version of the data that was available to the specific transaction when initialized.
2. Snapshot Isolation: MVCC comes with snapshot isolation, which makes sure that each transaction has a stable view of the overall database. This isolation avoids problems such as corrupted reads, where a transaction might access uncommitted data from another transaction, where the same record displays different values across consecutive reads [12]. Using snapshot isolation, the transaction uses a version of the database, as it was when the transaction started, irrespective of any changes that occur simultaneously across other nodes.

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

3. Time Synchronization: MVCC databases depend on a highly synchronized physical clock throughout the system to identify the order of transactions and its versions. This synchronization makes sure that newer versions of records remain consistent across distributed nodes, avoiding conflicts and ensuring the system stays uniform, even across multiple nodes or regions.
4. Conflict Detection and Resolution: MVCC databases can identify conflicts among transactions that try to update the same records. Before a transaction commits, the system verified whether any conflicts have occurred, such as two transactions altering the same record. If conflicts are found, the system generally reverts the transaction and performs a retry based on the configurations, thus preventing data anomalies [13].

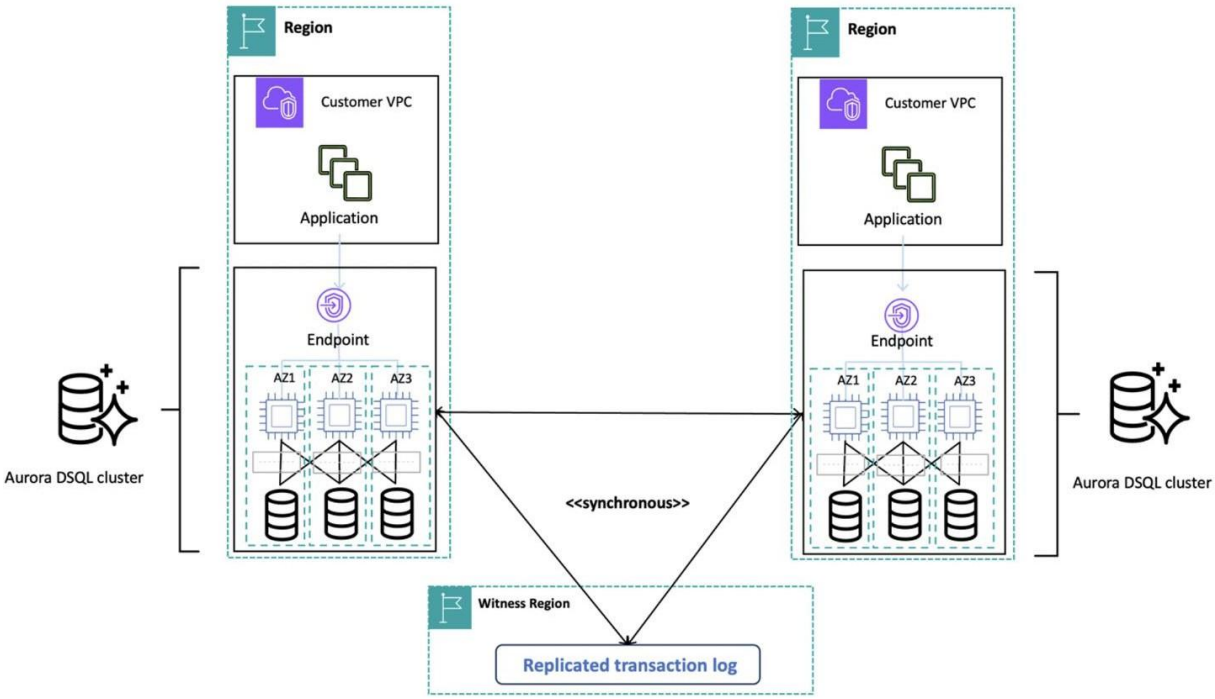

Figure1: Multi-Region Aurora DSQL Cluster [14]

3.1.2 Amazon DSQL Architecture

Amazon DSQL (Distributed SQL) is a cloud-native relational database designed to scale horizontally while providing strong consistency for transactional workloads [14]. Built on Amazon's Aurora platform, DSQL is designed to handle global-scale databases where the needs for high availability, scalability, and transactional consistency are critical. DSQL's architecture makes extensive use of MVCC to handle concurrent transactions in a distributed environment while ensuring strong consistency across regions. Figure 1 shows the high-level architecture of a multi-region Aurora DSQL cluster and below are the key components:

1. Transaction Router and Session Router: DSQL relies on a transaction router and a session router to direct incoming database requests to the appropriate region or node. These

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

components ensure that transactions are routed efficiently to minimize latency, while also ensuring that the state of transactions is consistent across regions.

2. Journal Service for Atomicity and Durability: One of the core components of DSQL is its journal service, which guarantees atomicity and durability by logging every transaction. This ensures that in the event of a failure, all changes made during a transaction are either fully committed or rolled back, maintaining data integrity.

3. Conflict Resolution Adjudicator: Since DSQL uses an active-active architecture where multiple regions may be writing to the same data concurrently, a conflict resolution adjudicator is used to handle conflicting writes. This component ensures that any conflicts between transactions (e.g., two regions updating the same record) are resolved according to the system's predefined conflict resolution strategy, ensuring that the final data remains consistent.

4. Global Database Synchronization: DSQL uses MVCC to ensure that data is synchronized across multiple regions while maintaining strong consistency. Data written in one region is asynchronously replicated to other regions. However, DSQL guarantees that even though the system is eventually consistent in terms of replication, the snapshot isolation provided by MVCC ensures that reads in all regions are consistent, even in the presence of concurrent writes.

3.2 Using MVCC Databases for SaaS Systems

The demand for high availability and resilience has pushed SaaS platforms to adopt multi-region architectures. Historically, active-passive architectures with primary and standby regions were the preferred model for high availability. Despite the benefits of active-active global databases for certain use cases, particularly when low-latency and scalability are the priority, transactional systems requiring high consistency, like financial transaction systems or ticketing platforms, often continue to rely on active-passive configurations [15]. This is because active-passive setups provide a strong consistency model and immediate failover with guaranteed data consistency during failover events. However, this often comes at the cost of higher latency and the need for quick disaster recovery procedures.

In this context, MVCC (Multi-Version Concurrency Control) databases offer a modern solution that directly addresses these challenges. MVCC enables strong consistency in an active-active setup by allowing concurrent reads and writes without locking data, thereby eliminating the need for disaster recovery procedures in the event of failure. This is achieved by maintaining multiple versions of data, ensuring that all regions or nodes in a multi-region system can operate independently while still maintaining global consistency. In addition to the core advantage MVCC databases provide with respect to adjusting the system topology to be setup in an active-active fashion, while maintaining high availability, scalability and performance, the strongly consistent nature of the databases provides additional advantages that can affect the entire architecture of a SaaS platform and minimize the footprint with respect to the additional setup that would otherwise be required with the prior options available and below two subsections describe two such major functionalities.

3.3 Dynamic Rule and Configuration Updates

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

One of the significant advantages of using MVCC in SaaS platforms is its ability to handle dynamic rule updates in a multi-tenant environment without introducing downtime or inconsistency. SaaS platforms, especially those serving multiple tenants, often require the ability to update business rules, pricing models, and feature configurations dynamically [16]. These updates should occur without disrupting service or causing tenants to see outdated data.

1. Seamless Rule Updates Across Regions: MVCC enables non-blocking updates, meaning that when a new rule or configuration is introduced, it can be applied dynamically without causing downtime or forcing a global shutdown. For example, if a tenant's pricing model is updated or a workflow rule is modified, MVCC ensures that all regions receive the updated configuration without disrupting ongoing transactions. The previous version of the data is retained for other regions that have not yet received the update, ensuring transactional consistency.
2. Ensuring Consistency in Rule Propagation: In multi-region setups, rule updates must be consistently propagated across regions. MVCC helps by maintaining multiple versions of the configuration data, where updates are versioned and asynchronously propagated, allowing regions to seamlessly transition to the new rule version once it is fully replicated.
3. Real-Time Application of Business Logic: For SaaS platforms, especially in use cases like e-commerce or financial transactions, the ability to apply business rules (e.g., tax calculations, discount logic) in real time is essential. MVCC ensures that these rules are dynamically applied across multiple regions without the risk of applying outdated logic, providing a consistent user experience for tenants and ensuring that data remains up-to-date across all regions.

3.4 Optimizing Asynchronous Processes and SLA Monitoring

Using MVCC databases within SaaS platforms can also help with optimizing complex asynchronous operations. Traditionally, many SaaS architectures are built around event-driven systems, where certain background tasks, SLA monitoring for example, are performed using queuing mechanisms, without interrupting real-time operations [17]. MVCC databases can enhance these features, by allowing these tasks to run concurrently with database operations without causing delays or inconsistencies.

1. Parallel Execution Without Blocking: As discussed, MVCC database allows for parallel execution of read and write operations while maintaining consistency. For example, in systems like ticketing platforms or payment gateways, multiple events (e.g., payment processing and user updates) can be handled asynchronously without blocking the main transaction flows.
2. Efficient SLA Monitoring: SLA monitoring in a multi-region active-active system is one such use case where MVCC shines. In high-performance systems like ticketing platforms, where data integrity and real-time processing are crucial, the ability to read the latest version of a transaction without requiring object locks ensures accurate SLA tracking, enabling the enterprises to adapt efficiently.
3. Event-Driven Workflows: Asynchronous tasks, such as notifications, inventory management, and order processing, are some of the trivial features of many SaaS platforms.

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

International Journal of Applied Mathematics Volume 38 No.

10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

MVCC ensures that these tasks can run independently of other operations without affecting the integrity of the data being processed. Even if a region fails or experiences delay, MVCC ensures that each region operates with the latest available data, improving resiliency and responsiveness.

4. Challenges and Limitations

1. The use of MVCC databases within multi-region systems comes with increased costs compared to traditional databases. For example, in Amazon DSQL, to ensure data is strongly synchronized across regions, an internal third region may be required to manage replication and synchronization. This third region acts as an intermediary for data synchronization and serves as a fallback if a region goes down, and this extra region setup increases the overall infrastructure and operational costs of the system.
2. While MVCC can perform non-blocking reads and writes, it can introduce a performance overhead, particularly in systems that deal with a lot of write calls. Since each write operation updating an existing record produces a new version, it leads to creating and retaining all these versions both in memory and on disk, potentially leading to increased storage demands and higher I/O requirements. This situation could negatively impact system performance, especially under high levels of concurrency or during large-scale transactions.

5. Conclusion

In conclusion, the article presents how MVCC databases can enhance and transform SaaS products, especially when using active-active deployment strategies. For any SaaS platform that continues to expand on a global scale, it is highly essential that we ensure that the systems are high available and strongly consistent. As discussed, conventional strategies, such as active-passive or eventually consistent active-active database setups, often face difficulties in meeting the requirements of transactional systems that demand both high performance and strong consistency.

As part of the analysis, the article explains how MVCC databases facilitate active-active architectures to operate with strong consistency, overcoming the performance limitations and data discrepancies that may occur in conventional database systems. At a high-level, MVCC accomplishes this by allowing simultaneous reads and writes across various database nodes across more than one region, while ensuring that the data stays consistent without requiring complex disaster recovery processes that could result in potential data loss.

Additionally, the article explores how MVCC databases can improve SaaS systems, especially in scenarios where dynamic rule updates and asynchronous workflow handling are needed. The ability to implement real-time modifications to the business rules without service interruption, and to be able to efficiently manage complex asynchronous operations, makes MVCC an excellent option for modern, adaptable SaaS platforms. MVCC databases do come with certain challenges related to cost, latency, and complexity. The incorporation of MVCC with SaaS platforms does require a thorough evaluation of these trade-offs, along with innovative strategies to enhance performance and cost efficiency.

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

7. References

[1] Tsai, W.-T., Tsai, W.-T., Bai, X., & Huang, Y. (2014). Software-as-a-service (SaaS): perspectives and challenges. *Science in China Series F: Information Sciences*. https://doi.org/10.1007/S11432-013-5050-Z

[2] Morakos, P., & Meliones, A. (2014, July 7). Design and implementation of a Cloud SaaS framework for Multi-Tenant applications. *International Conference on Information, Intelligence, Systems and Applications*. https://doi.org/10.1109/IISA.2014.6878755

[3] Dedase, A. (2024). Scalable Software as a Service Architecture. *arXiv.Org*. https://doi.org/10.48550/arxiv.2403.05377

[4] Domaschka, J., Hauser, C. B., & Erb, B. (2014, September 1). Reliability and Availability Properties of Distributed Database Systems. *Enterprise Distributed Object Computing*. https://doi.org/10.1109/EDOC.2014.38

[5] Sun, H., Bang, X., Wang, X., & Liu, X. (2017). Adaptive trade-off between consistency and performance in data replication. *Software - Practice and Experience*. https://doi.org/10.1002/SPE.2462

[6] Malkhi, D., & Martin, J.-P. (2013). Spanner's concurrency control. *Sigact News*. https://doi.org/10.1145/2527748.2527767

[7] Kulkarni, G., Shelke, R., Palwe, R., Khatawkar, P., Bhuse, S., & Bankar, H. (2013, July 4). *Multi-tenant SaaS cloud*. https://doi.org/10.1109/ICCCNT.2013.6726487

[8] Tsai, W.-T., Huang, Y., Bai, X., & Gao, J. (2012, April 11). Scalable Architectures for SaaS. *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. https://doi.org/10.1109/ISORCW.2012.44

[9] Parui, U., & Sanil, V. (2016). *High Availability and Disaster Recovery Concepts*. https://doi.org/10.1007/978-1-4842-2071-9_1

[10] Lomet, D. B., Fekete, A., Wang, R., & Ward, P. (2012, April 1). Multi-version Concurrency via Timestamp Range Conflict Management. *International Conference on Data Engineering*. https://doi.org/10.1109/ICDE.2012.10

[11] Sippu, S., Sippu, S., & Soisalon-Soininen, E. (2014). *Concurrency Control by Versioning*. https://doi.org/10.1007/978-3-319-12292-2_12

[12] Elnikety, S., Pedone, F., & Zwaenepoel, W. (2005, October 26). Database replication using generalized snapshot isolation. *Symposium on Reliable Distributed Systems*. https://doi.org/10.1109/RELDIS.2005.14

[13] Sadoghi, M., Canim, M., Bhattacharjee, B., Nagel, F., & Ross, K. A. (2014). *Reducing database locking contention through multi-version concurrency*. https://doi.org/10.14778/2733004.2733006

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[14] Amazon Web Services, "Introducing Amazon Aurora DSQL," Dec. 2024. Accessed: July, 2025. [Online]. Available: https://aws.amazon.com/blogs/database/introducing-amazon-aurora-dsql/

[15] Chirumamilla, S. K. (2021). Designing with High Availability: Achieving Fault Tolerance in Software Engineering through AWS Multi-Region Architectures. *International Journal For Multidisciplinary Research*. https://doi.org/10.36948/ijfmr.2021.v03i06.40677

[16] Walraven, S., Van Landuyt, D., Truyen, E., Handekyn, K., & Joosen, W. (2014). Efficient customization of multi-tenant Software-as-a-Service applications with service lines. *Journal of Systems and Software*. https://doi.org/10.1016/J.JSS.2014.01.021

[17] Aghera, P., Chaudhary, S., & Kumar, V. (2012, May 11). An Approach to Build Multi-tenant SaaS Application with Monitoring and SLA. *International Conference on Communication Systems and Network Technologies*. https://doi.org/10.1109/CSNT.2012.146

Received: Aug 04, 2025; Accepted: Nov 04, 2025; Published: Nov 12, 2025