Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Streamlined and Efficient Management of Non-Relational Cloud Database Transactions

Dr. Shantanu S Bose[1], Dr. Sachin Deshmukh[2], Devdatta Tare[3], Anup Suchak[4], Rajashree Joshi[5], Lalit Agrawal[6]

[1]Assistant Professior, Amity Business School, Amity University Chhattisgarh, shantanu@rediffmail.com
[2]Associate Professor, MIT WPU
[3]Assistant Professor, Post Graduate Teaching, Department of Commerce, Gondwana University, Gadchiroli.
[4]Central Institute of Business Management Research and Development Nagpur
[5]Associate professor, Don Bosco Institute of Management Studies and Computer Applications
[6]Assistant Professor, ShriRamdeobaba College of Engineering and Management

As cloud-based non-relational databases continue to gain traction for their scalability and flexibility, the management of transactions within these databases becomes increasingly crucial. Unlike traditional relational databases, non-relational databases often lack built-in transactional support, posing challenges for maintaining data integrity and consistency in distributed environments. This paper proposes a streamlined and efficient approach to anaging transactions in non-relational cloud databases. The proposed approach leverages a combination of architectural principles and advanced transaction management techniques tailored for non-relational databases. It focuses on optimizing transactional workflows, ensuring atomicity, consistency, isolation, and durability (ACID properties) while minimizing overhead and latency. Key components of the approach include distributed transaction coordination, optimistic concurrency control, and versioning mechanisms.
Furthermore, the paper explores the integration of transaction management with cloud-native technologies such as microservices architecture and container orchestration platforms. This integration facilitates seamless scalability, fault tolerance, and resource efficiency in transaction processing.

Keywords: Non-relational databases, Cloud computing, Transaction management, Scalability Flexibility, Atomicity.

1. Introduction

Cloud-based non-relational databases have emerged as a cornerstone of modern data management systems, offering unparalleled scalability and flexibility for storing and processing vast amounts of data. These databases, often referred to as NoSQL databases, have gained widespread adoption across various industries due to their ability to handle diverse data types and support distributed architectures[1].

Unlike traditional relational databases, which adhere to the principles of ACID (Atomicity, Consistency, Isolation, Durability), non-relational databases typically prioritize scalability and performance over strict transactional guarantees[2]. As a result, developers face the task of devising efficient strategies for managing transactions in non-relational cloud databases without compromising on these critical properties[3].

This paper presents a novel approach to addressing the transaction management challenges inherent in non-relational cloud databases. Our approach is grounded in a synthesis of architectural principles and advanced transaction management techniques tailored specifically for the characteristics of non-relational databases[4]. By leveraging distributed transaction coordination, optimistic concurrency control, and versioning mechanisms, we aim to optimize transactional workflows while minimizing overhead and latency[5].

A vast network of independent computer nodes is the basis of distributed and cloud computing systems. A storage area network (SAN), local area network (LAN), or wide area network (WAN) hierarchically connects these node devices. At now, a functional cluster consisting of hundreds of machines may be established with just a handful of switches[6]. Multiple smaller clusters can be linked together via a WAN to create one or more massive clusters. Massive systems are capable of reaching physically or logically web-scale connectivity and are hence highly scalable. There are four main types of massive systems: computing grids, internet clouds, clusters, and peer-to-peer (P2P) networks. Collectively, cooperatively, or collaboratively, these devices operate on several levels.

Computers in a cluster are actually a collection of interconnected personal computers that share resources and can process massive amounts of data quickly and efficiently. A typical server cluster, as shown in Figure 1, is designed around an interconnection network that has low latency and high bandwidth. Virtual Private Network (VPN) gateways allow for the construction of interconnection networks with numerous levels of Ethernet or switches, allowing for larger clusters with more nodes to join[7]. Server operating systems often handle resource management in clusters with weakly linked nodes. If a cluster is perfect, it will combine many system images into one (SSI).

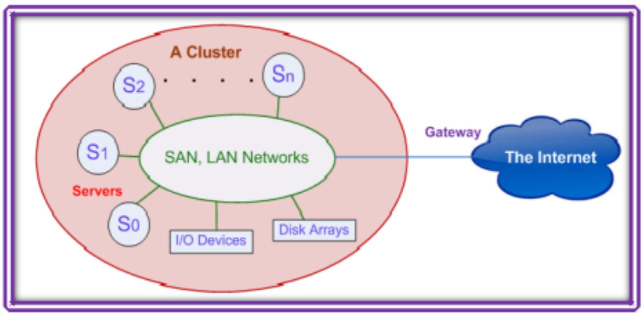

Figure 1: A Cluster of Servers

Cloud server systems, as seen in Figure 2, need a large number of storage-rich servers running virtualization software to enable the simultaneous execution of several application programming interfaces (APIs). By enabling the simultaneous operation of many operating systems, virtualization is the crucial technology for optimizing server resources[8]. In application programming interfaces, this may be a game-changer since it gives consumers a benchmark for performance on the cloud. Keep in mind that the majority of server setups will employ an open source operating system like Linux, which is great for developers due to its reliability[9].

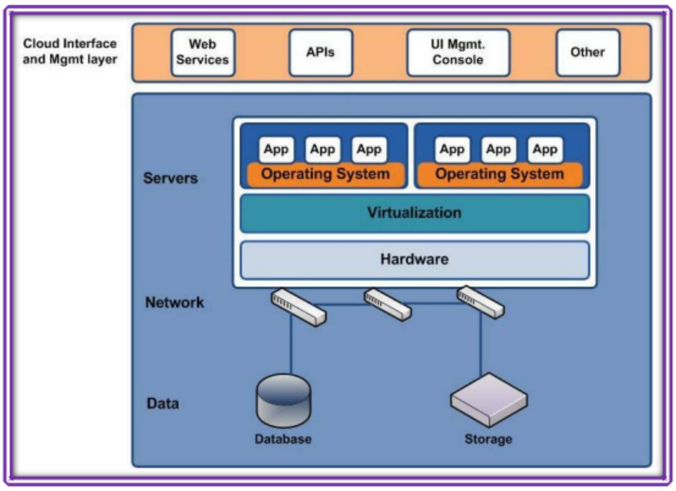

Figure 2: Cloud Constitution Elements

2. Cloud Computing Components:

Clients, a data center, and distributed servers are the building blocks of a cloud computing system. Users control their information in the cloud via clients, which are devices they communicate with. The portability of these devices—desktops, laptops, tablets, smartphones, and personal digital assistants—is a major factor propelling the growth of cloud computing. Customers may often be thin or thick, and they might be movable. Thin clients are often lauded for their many desirable qualities, such as reduced hardware prices, IT expenses, noise, power consumption, and security. Users' subscription applications are stored in the datacenter, which is a network of servers[10]. Server virtualization is becoming more popular in the IT industry. One reason virtualization is important to cloud computing is because it allows users to access cloud services. Virtualization comes in two varieties.

When one machine's installation is run entirely on another, this is called full virtualization. The end result is a setup where every server-side program operates within its own virtual machine. Clients see the server-side applications in a fully virtualized setup. Both distinct apps and operating systems may coexist in such a setup. A certain set of hardware configurations was required to enable complete virtualization. It wasn't until 2005 that it was put into action, but with the release of AMD-Virtualization (AMDV) and Intel Virtualization Technology (IVT), everything about virtualization became much simpler[11]. Several uses have found success with full virtualization, including: allowing several users to use a single computer system; isolating users from one another and the control application; and creating hardware emulators on other machines.

Through the use of para virtualization, a single physical device may run many operating systems simultaneously while making better use of system resources such as memory and CPUs. An OS that has been modified to run in a virtual machine is used by the management module. Due of the need to replicate every component in the full virtualization approach, para virtualization is superior[12].

Distributed Servers: The servers will be situated in different parts of the world. These servers may not really be physically located together, but to the cloud subscriber, it seems as if they are. Because of this, the service provider has greater leeway in terms of customization and safety. No more servers should be stashed away in the event that the cloud requires additional hardware[13]. They may just upload them to the cloud and use them from another location.

3. Rudiments of Transactions

Each operation against a database is called a transaction. Whether performed manually by an end user or automatically by a database software, transactions are units or sequences of work that are completed in a logical order. The process of updating a database with one or more modifications is called a transaction. Users are making transactions on the table whenever they do things like adding, editing, or removing records[14]. In order to manage database mistakes and maintain data integrity, transaction management is crucial. Moving funds between bank accounts is a typical example. That can only be accomplished by transferring funds from the user's original account to their new one. Complete success of the procedure is required. Money will be lost if the procedure stops midway, and the bank risks losing the client due to irregular transactions.

Some additional things that modern database transactions perform include making sure that data cannot be accessed after someone else has written it partially. The underlying principle is the same, though: transactions guarantee that one's data is always in a reasonable state. They promise that funds will never be transferred from one account to another without the other party's knowledge. Distributed transactions are those that include several applications or hosts and are implemented by database systems[15]. Any number of systems, including databases, file systems, messaging systems, and others, may be involved in a distributed transaction that enforces the ACID characteristics. A coordinating service makes sure that all the necessary systems are updated with the details of a distributed transaction. If any portion of the transaction fails, all impacted systems will be rolled back, just as with database and other types of transactions. Thus, a transaction is just a set of actions. Database management systems' (DBMS's) transaction management capabilities allow them to oversee the many system transactions. Database management systems should guarantee correct transaction execution; in other words, the transaction must execute in whole or in part. A transaction is a logical unit of database operations that may access several data objects. In most cases, it is the product of a program coded in a programming language or data manipulation language with a high degree of abstraction. The ACID transaction characteristics should be followed by DBMS to guarantee data integrity. The four principles of ACID are A=Atomicity, C=Consistency, I=Isolation, and D=Durability. Atomicity: The database correctly reflects all of the transaction's actions, or none of them. The term "all or nothing" describes this kind of feature.

Each database transaction is executed as a single unit using a "all or nothing" approach, thanks to atomicity of transactions. The whole transaction is reversed if even a single statement fails. For a transaction to be considered consistent, it must change the database state from one that is consistent to another. In addition, relational databases check that all transactions follow the rules set down by the database administration. If any one part of an atomic transaction could compromise the database's integrity, the whole transaction would fail.

4. Non-relational cloud databases differ from traditional databases

Non-relational cloud databases differ from traditional databases in several ways:

Data Structure: Traditional databases are primarily designed for structured data, while non-relational databases can handle unstructured, semi-structured, and structured data

Scalability: Non-relational databases are designed for horizontal scalability, allowing them to handle large volumes of data and sudden surges in demand more efficiently than traditional databases

Flexibility: Non-relational databases offer greater flexibility, with schema-on-read or schema-free options, allowing them to accommodate changing business requirements more easily than traditional databases

Data Storage: Non-relational databases can store large amounts of data with little structure, while traditional databases are table and row-oriented, requiring a predefined schema

Query Language: Traditional databases use SQL (Structured Query Language) for shaping and manipulating data, while non-relational databases use Object-relational-mapping (ORM) or other query languages

Cost and Maintenance: Non-relational cloud databases typically have lower upfront costs, as they don't require purchasing and maintaining physical servers, and offer flexible subscription-based models

Data Integrity: Traditional databases provide more control over data integrity, as users can design data types, restrictions, relationships, and rules that govern the data

Data Access: Traditional databases can provide faster data access and delivery, as they are not affected by network latency or congestion.

5. Data consistency

Non-relational cloud databases handle data consistency using different approaches compared to traditional relational databases. While relational databases use a structured approach to data management, ensuring data accuracy and consistency, non-relational databases offer flexibility in handling unstructured or semi-structured data with ease

Non-relational databases often employ the BASE consistency model, which stands for Basically Available, Soft state, Eventually consistent. This model prioritizes availability and partition tolerance over immediate consistency, allowing for high performance and scalability

BASE databases do not guarantee immediate consistency across all replicas, but they ensure that data will eventually become consistent, either at read time or for certain processed past snapshots

In contrast, relational databases typically use the ACID consistency model, which ensures Atomicity, Consistency, Isolation, and Durability. This model guarantees that all operations in a transaction either succeed or are rolled back, maintaining a consistent state and isolating transactions from one another

The choice between ACID and BASE consistency models depends on the specific use case and the trade-offs between data consistency and performance. For applications requiring strong data consistency and integrity, such as financial or healthcare systems, ACID-compliant databases are often preferred. However, for applications where availability and scalability are more important, such as social media platforms or real-time analytics, non-relational databases with BASE consistency models may be more suitable

Non-relational cloud databases handle data consistency using the BASE consistency model, which prioritizes availability and partition tolerance over immediate consistency. This approach allows for high performance and scalability but may not provide the same level of data consistency as traditional relational databases using the ACID consistency model

6. Challenges in maintaining data consistency in non-relational cloud databases

Indian economy has witnessed an uprising in the futures market since 2003 as a result of government removing the ban on futures trading. There are a range of reasons responsible for the unproductive growth of commodity futures market in India. But one among the major reason is price risk in commodity market, especially towards agriculture commodity. The production, supply and distribution of many agricultural commodities are restricted by the government and futures trading are allowed in certain agriculture commodity. Instability in prices of agriculture commodity is a major issue of the producers as well as the end users in an agriculture subjugated country like India. Generally, traders are interested to trade on instruments that provide safe returns for their investment[16]. The thumb rules of the market are low risk will provide low return. But now diversification of portfolio helps to reduce risk and increase return. Commodity market has attracted many investors, producers and farmers to trade and hedge their risk.

Isolation: Actually, DBMS executes several transactions simultaneously. But the system makes sure that no two transactions are running in parallel with each other. If more than one transaction is about to happen at the same moment, the database engine will compel them to be isolated. No two transactions happen at the same time, and the database view that a transaction starts with is the only one that is changed before it ends. It is unacceptable for one transaction to ever touch the byproduct of another.

Best practices for ensuring data consistency in non-relational cloud databases include:

Balancing Strong and Eventual Consistency: Non-relational databases like Google Cloud Datastore offer a balance between strong and eventual consistency to provide a positive user experience while handling large quantities of data and users

Understanding Data Models: Choosing the appropriate data model, such as key-value, document, or graph databases, can help optimize performance and data consistency for specific use cases

Implementing Replication Strategies: Implementing replication strategies can ensure data durability and fault tolerance, and can help maintain consistency in the event of network failures or concurrent updates

Using Indexing and Data Structures: Efficient indexing and data structures can support efficient querying and optimize read and write performance

Defining Consistency Requirements: It is critical to choose the right consistency level according to the application's data availability, accuracy, and performance needs.

Implementing Robust Security Measures: To safeguard the NoSQL database and prevent data corruption, alteration, or loss, authentication methods, encryption, and access control lists may be used.

Regularly Monitoring and Optimizing: Regularly monitoring and optimizing the database, including indexes, queries, and security measures, can help ensure optimal performance, scalability, and data integrity

Table 1: Transaction Schedules

BEGIN-TRANSACTION Y=0;Y=Y+1; END-TRANSACTION	BEGIN-TRANSACTION Y=0; Y=Y+2; END-TRANSACTION	BEGIN-TRANSACTION Y=0;Y=Y+3; END-TRANSACTION

Three different transactions are illustrated in table 1. If all the three are executed serially, the result of Y will be 3 at the end. The arrangements of the transactions as

1. Y=0;Y=Y+1; Y=0;X=Y+2; Y=0;Y=Y+3;

2. Y=0;Y=0;Y=Y+1;Y=Y+2;Y=0;Y=Y+3;

3. Y=0;Y=0;Y=0;Y=Y+1;Y=Y+2;Y=Y+3; Of the above arrangements, 1 and 2 will be correct since Y=3 at the end. But 3 will be illegal since Y=5. So serilizability is the important aspect of isolation

7. Conclusion

This paper has presented a comprehensive approach to achieving streamlined and efficient management of transactions in non-relational cloud databases. We have addressed the challenges posed by the absence of built-in transactional support in non-relational databases, focusing on optimizing transactional workflows while ensuring atomicity, consistency, isolation, and durability (ACID properties).Our proposed approach leverages a combination of architectural principles and advanced transaction management techniques tailored specifically for non-relational databases. By incorporating distributed transaction coordination, optimistic concurrency control, and versioning mechanisms, we have demonstrated the ability to maintain data integrity and consistency in distributed cloud environments.

Conflicts of Interest

The authors declare that they have no competing interests.

References

1. A. Raut, "NOSQL database and its comparison with RDBMS," International Journal of Computational Intelligence Research, vol. 13, no. 7, pp. 1645–1651, 2017.
2. J. Pokorny, "NoSQL databases: a step to database scalability in web environment," International Journal of Web Information Systems, vol. 9, no. 1, pp. 69–82, 2013.
3. R. Kanwar, P. Trivedi, and K. Singh, "NoSQL, a solution for distributed database management system," International Journal of Computer Applications, vol. 67, no. 2, pp. 6–9, 2013.
4. D. McCreary and A. Kelly, Making Sense of NoSQL, Manning, Shelter Island, NY, USA, 2014.
5. Y. Li and S. Manoharan, A Performance Comparison of SQL and NoSQL Databases, IEEE, Piscataway, NJ, USA, 2013.
6. Z. Bicevska and I. Oditis, "Towards NoSQL-based data warehouse solutions," Procedia Computer Science, vol. 104, pp. 104–111, 2017.
7. M. Stonebraker, "SQL databases v. NoSQL databases," Communications of the ACM, vol. 53, no. 4, pp. 10-11, 2010.
8. J. Han, H. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in Proceedings of the 2011 6th International Conference on Pervasive Computing and Applications, Port Elizabeth, South Africa, October 2011.
9. D. G. Chandra, "Base analysis of NoSQL database," Future Generation Computer Systems, vol. 52, pp. 13–21, 2015.
10. D. Bermbach and S. Tai, "Eventual consistency: how soon is eventual? An evaluation of Amazon S3's consistency behavior," in Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, pp. 1–6, Lisbon, Portugal, December 2011.
11. P. Vassiliadis, "A survey of extract-transform-load technology," International Journal of Data Warehousing and Mining, vol. 5, no. 3, pp. 1–27, 2009.
12. P. Vassiliadis and A. Simitsis, "Extraction, transformation, and loading," Encyclopedia of Database Systems, Springer, Boston, MA, USA, 2018.
13. S. K. Bansal and S. Kagemann, "Integrating big data: a semantic extract-transform-load framework," Computer, vol. 48, no. 3, pp. 42–50, 2015.
14. Nejrs, Salwa Mohammed(2023) Medical images utilization for significant data hiding based on machine learning, Journal of Discrete Mathematical Sciences and Cryptography, 26:7, 1971–1979, DOI: 10.47974/JDMSC-1785
15. Lin, Lon, Lee, Chun-Chang, Yeh, Wen-Chih& Yu, Zheng(2022) The influence of ethical climate and personality traits on the performance of housing agents, Journal of Information and Optimization Sciences, 43:2, 371-399, DOI: 10.1080/02522667.2021.2016986
16. Johri, P., Khatri, S.K., Al-Taani, A.T., Sabharwal, M., Suvanov, S., Kumar, A. (2021). Natural Language Processing: History, Evolution, Application, and Future Work. In: Abraham, A., Castillo, O., Virmani, D. (eds) Proceedings of 3rd International Conference on Computing Informatics and Networks. Lecture Notes in Networks and Systems, vol 167. Springer, Singapore. https://doi.org/10.1007/978-981-15-9712-1_31