ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Parallel and Distributed Mechanisms for Data Access in Enciphered Clouds

T.Supraja Pavithra
M.Tech Student,
Department of CSE,
B V Raju Institute of Technology,
Narsapur, India.

Mrs.J.Manjula
Assistant Professor,
Department of CSE,
B V Raju Institute of Technology,
Narsapur, India.

Abstract:

Cloud database environments are very attractive for the deployment of large scale applications due totheir highly scalable and available infrastructure. The main reason for the users deploying different types of applications in the cloud is its pay-for-use cost model. This survey contains the most prominent concurrency control protocols that can be used in the encrypted cloud database. The degree of data consistency and cost requirements varies according to the concurrency control protocols.

Index Terms:

Cloud; database; data consistency; concurrency control.

1. INTRODUCTION:

Cloud based services are becoming popular as they focus on high availability and scalability at low cost. While providing high availability and scalability, placing critical data to cloud poses many security issues. For avoiding these security issues the data are stored in the cloud database in an encrypted format. The encrypted cloud database allows the execution of SQL operations by selecting the encryption schemes that support SQL operators. Encrypted cloud database permits different types of accesses such as distributed, concurrent, and independent. One of the architecture that supports these three kinds of access is SecureDBaaS, which was proposed by Luca Ferretti et al [1]. The SecureDBaaS architecture supports multiple and independent clients to execute concurrent SQL operations on encrypted data. Data consistency should be maintained by leveraging concurrency control mechanisms used in DBMS engines.This survey explains the various concurrency control protocols that can be used in the encrypted cloud database. The applications need 1SR if data is replicated.

Hence, to guarantee the merits of cloud, it is essential to provide high scalability, availability, low cost and data with strong consistency, which is able to dynamically adapt to system conditions. Self-optimizing one copy serializability (SO-1SR) is the concurrency control protocol that dynamically optimizes all stages of transaction execution on replicated data in the cloud database [2]. Current DBMSs supported by cloud providers allows relaxed consistency guarantees which in turn increase the design complexity of applications [3].The second concurrency control protocol is the snapshot isolation (SI) which provides increased concurrency in cloud environment when compared to 1SR [4]. Transactions are read from the snapshot, reads are never blocked because of write locks which in turn increases concurrency. SI does not allow many of the inconsistencies, but allows write skew anomalies. SI allows transaction inversions. To avoid transaction inversions strong consistency guarantee is required, i.e. strong SI (SSI).The third concurrency control protocol is the session consistency (SC) [5]. Session consistency is a different variety of eventual consistency. The system provides read your writes consistency inside each session. Session consistency is at a low cost while considering response time and transaction cost.The cost based concurrency control in the cloud is the C3 i.e. cost-based adaptive concurrency control in cloud [6]. C3 dynamically switch between strong consistency level and weak consistency level of transactions in a cloud database according to the cost at runtime. It is built on the top of 1SR and SSI.

2. SECUREDBAAS:

SecureDBaaS (Secure database as a service) architecture proposed by Luca Ferretti et al supports multiple clients and clients which are geographically distributed to execute the independent and concurrent operation on encrypted data in the remote database [1]. SecureDBaaS also guarantees data confidentiality and cloud level consistency.

This architecture eliminates the intermediate server between the cloud database and client in order to provide availability and scalability [7].SecureDBaaS is the architecture that supports the concurrent execution of operations in the encrypted cloud database. The existing proxy based architecture constraints the multiple and distributed clients to access data concurrently from the same database. The data consistency during the concurrent access of data and metadata can be assured by using some isolation mechanisms or the concurrency control protocols in the cloud database. SecureDBaaS allows the execution of concurrent SQL operations (INSERT, DELETE, SELECT, UPDATE) from multiple and distributed clients. In order to provide data confidentiality the tenant data and metadata should be in an encrypted format. For this reason, clients convert plaintext SQL statements into SQL statements that support transactions and isolation mechanisms allowed in cloud databases [8]. The solutions for the consistency issues lies in the five contexts: (1) data manipulation (2) modification of structures (3) altering table (4) modification of secure type (5) unrestricted operations.

2.1. Architecture design:

The architecture design of SecureDBaaS consists of one or more client machines with SecureDBaaS installed and cloud database. This client is responsible for the connection of a user to the cloud DBaaS to perform SQL operations. The SecureDBaaS manages plaintext data, metadata, encrypted data and encrypted metadata. The plaintext data includes the data user wants to save in cloud DBaaS [9]. In order to avoid the confidentiality issues, multiple cryptographic approaches are used to convert plaintext data to encrypted form for storing in cloud database. The metadata includes information needed to encrypt or decrypt data. Moreover, metadata is also stored in an encrypted format [10].

Encryption Schemes:

The encryption schemes supported by SecureDBaaS [11] are:(1) Plain: it supports the storage of unencrypted data in the cloud and allows all types of SQL operations. (2) OPE: order preserving encryption permits the execution of inequality and range queries on encrypted data. (3)Det: it permits the execution of equality and aggregation operators on encrypted data. (4)Random: it assures highest confidentiality level. But it restricts all SQL operators.

2.2. Implementation:
SecureDBaaS client consists of five components:

Operation parser software:
Is responsible for the conversion of receiving plain text SQL command into intermediate form which is processed later by other modules.

Encryption engine:
Is responsible for all kinds of encryption and decryption operations specified in the metadata of SecureDBaaS.

Metadata manager:
it manages metadata local copies and assures its consistency.

Query writer:
it translates the query in intermediate form from the operation parser into SQL statements that can be executed by the cloud database over encrypted data.

Database connector:
it acts as an interface between client and remote DBMS.

3.CONCURRENCY CONTROL PROTOCOLS:

In what follows, we briefly present the most prominent concurrency control protocols that can be used in cloud database.

3.1. Self-optimizing One Copy Serializability (SO-1SR):

1SR is the strongest and well known correctness criterion for applications that are newly deployed in the cloud. It assures the serializable execution of concurrent transactions and a one copy view of the data. The most commonly used approaches to implement 1SR is to use lock based protocols such as strict two-phase locking (S2PL) for providing serializable transaction execution and two-phase commit (2PC) for synchronous updating all replicas.

3.1.1. Transaction model:

In a system providing 1SR, each transaction which writes to a data object must update all copies of the data object. In case of update transactions the replicated data increases the response time and thus decreases the overall scalability of the system. In order to exploit the merits of the cloud, it is essential to provide scalability, availability, low cost and strongly consistent data management. Under distributed systems, it is not possible to provide consistency and availability. The stronger consistency level decreases the availability and scalability.

In cloud environments, the cost of guaranteeing a certain consistency level on top of replicated data is to be considered. Strong consistency is costly; on the other hand, weak consistency is cheaper, but may lead to high operational costs of compensating the effects of anomalies and access to stale data. The first generation cloud DBMS's provide on the weak consistency in order to provide maximum scalability and availability. It is sufficient for satisfying requirements related to consistency of simple cloud applications.

However, more sophisticated like web shops, online stores and credit card services requires strong consistency levels. The advantages of cloud such as availability and scalability are not yet exploited by existing commercial and open source DBMS's which provide strong consistency [12].SO-1SR (self-optimizing 1SR) is a novel protocol for replicated data in a cloud that dynamically optimize all phases of transaction executions. System model of SO-1SR assumes that applications are built on the top of a cloud data environment.

3.1.2. Implementation:

The SO-1SR middleware should be present at each replica node. The transactions that are submitted by the client to the application servers are forwarded to the SO-1SR middleware for optimal execution. The SO-1SR is based on a fully replicated system and flat transaction model. Protocols like 2PC or Paxos are needed to provide strong consistency guarantees. The main goal of SO-1SR is to decrease latency by using dynamic optimization technique at different phases of transaction life cycle, not to replace protocols like 2PC or Paxos. .

3.2. Snapshot Isolation:

The transactional guarantees of SI are weaker than 1SR, such that the database system can achieve increased concurrency by relaxing isolation requirements on transaction. In SI, the transaction attempting read is never blocked. The tradeoff between transaction isolation and performance is that higher degrees of transaction isolation assure fewer anomalies. Anomalies avoided by 1SR are also avoided in SI. Under SI, write skew anomaly is possible if two transactions concurrently update one or more common data item. For example, consider two transactions Tm and Tn. Transaction Tm reads data items p and q and then updates concurrently with other transaction Tn that reads data item p and q and then updates q. Here transaction Tm and Tn do not have a write-write conflict because none of the transaction updates a common data item.

Different variations of SI exist for replicated systems like cloud which provide different consistency guarantees. In a lazily synchronized replicated database system; if two transactions Ts and Tv do not have a write–write conflict under SI, then their updates may be committed in the order Ts followed by Tv at a site S1 but in reverse order at another site S2 in which each site individually guarantees SI. In this case, consider a transaction Tk that reads x and y at site S1 and view database state from the commit of Ts will not view this same database state if it were to be executed on the database replica at site S2.But this kind of replica in consistency will not occur in a centralized database system that guarantees SI.

SI was introduced by Berenson et al [13]. SI is defined as; it does not allow dirty reads, dirty writes, non-repeatable reads, phantoms or lost updates. Write skew anomalies are possible in SI. By the definition of SI, when the transaction starts the system assigns a transaction Ta start timestamp called start (T). The database state seen by T is determined by start (T). The system can choose any time less than or equal to the actual start time of T to start (T). The update transactions made by Tl that commit after start (T) will not be visible to T. Only update transaction that commits before start (T) will be visible to T. Each transaction T is able to see its own updates are also a requirement in SI. Thus, if T updates a database item and reads that item, then T will see the updating even though the update occurred after the start (T).

3.2.1. Transaction model:

Commit timestamp, commit (T) is assigned to a transaction when a transaction is to commit. The time commit (T) is more recent than any other start or commit timestamp assigned to any transaction. If no other committed transaction Tk with lifespan [start (Tk), commit (Tk)] that overlaps with a T's lifespan of [start (T), commit (T)] write data that T has also written then only T commits. Otherwise, to prevent lost updates T is getting aborted. This technique of preventing lost updates is called the first-committer-wins (FCW) rule.Transaction inversions are possible in SI, i.e. for every pair of transactions T1 and T2, if T2 executes after T1 then T1 will view T1's updates. This is because the actual start time of T2 can be larger than that of a start (T2). In particular, if T2 starts after T1 has finished, then T2 will see a database state that does not contain the effects of T1. In order to prevent these kinds of transaction inversions, strong SI is introduced.In the definition of strong SI (SSI), if for every pair of committed transactions Tp and Tq in transaction history TH such that Tp's commit precedes the first operation of Tq, start (Tq) > commit (Tp) and it is SI then we can say that the transaction execution history TH is strong SI.

3.2.2. Implementation:

The decentralized model of SI based transactions consists of some mechanisms such as: (a) Keeping a consistent, committed snapshot for reading (b) a global sequencer is used for arranging the transactions by allocating commit timestamps (c) detection of write-write anomalies in concurrent transactions and(d) atomically commit the updates and make them durable. In the model, each transaction goes through a sequence of phases during execution. The main phase is the active phase in which all read/write on data item is performed in this phase. The remaining phases are part of the commit of the transaction. Validation phase is required for detecting the conflicts among transactions that are executed concurrently.

3.3. Session Consistency:

Session Consistency is considered to be the minimum consistency level in a distributed environment that does not result in complexities for application developers. Under Session Consistency, the application will not see its own updates and may get inconsistent data from successive accesses. The key idea is that, all data does not need the same level of consistency. There is a term called consistency rationing i.e. the data is divided into three categories A, B, C and each type of data is treated differently depending on the consistency level provided.The category A contains data in which consistency violations may result in large penalty costs. The category B includes data where the consistency requirements change over time.

Category C comprises data in which inconsistency is acceptable. Session consistency considers data under category C. C category is always a preferred category for placing data in the cloud database [14]. By considering a transaction cost and response time the session consistency is very cheap; because only few messages are needed as compared to strong consistency guarantees. The performance level can be increased by providing extensive caching mechanisms which in turn lowers the cost.

3.3.1. Transaction model:

By sessions, the client connects to the system. The system assures read your own writes monotonicity as the session ends. A new session cannot view the writes of the last executed session, immediately. The updates in sessions of different clients are not always visible to each other. As the time passes, the system converges and acquires consistency called eventual consistency. The conflicts for concurrent updates in the C category data depends upon the type of update. In case of commutative and non-commutative updates, the former is solved by the last update wins and the latter is solved by performing the updates one after the other. But the inconsistencies are possible in both cases.

3.3.2. Implementation:

The implementation is done on top of the Amazon's simple storage service (S3). The key idea is, each page's highest commit timestamp is recorded that is cached by the server in the past. The server can check if a server receives an outdated copy of the page from S3 and can update the page from S3. The session consistency can be guaranteed by forwarding all requests from the same client to the same server under a session. The session ID is used for the routing mechanism and the request is redirected accordingly.

3.3. Cost-Based Adaptive Concurrency Control (C3):

Cost plays an important role in the cloud environment along with the performance [15]. The strongconsistency leads to high cost, whereas weak consistency leads to high operational costs [16]. In C3 approach, a consistency rationing model is used which categorized the data into three: the first category contains data which require ISR, the second category data require SC and the third category data handled with adaptive consistency. At the data level, specific policy will be defined based on that policy consistency level is selected between 1SR and SC at the time of running. Moreover, C3 is implemented on the top of 1SR, SC and SSI concurrency protocols by utilizing the resources provided by the cloud providers.The update anywhere and full replication procedure are the basis for the C3 system model. The updating of all replicas will be carried out in ISR and SSI transactions using 2PC, while SC transactions only commits at the remote local replicas. The C3 model does not introduce any hindrance for the replication strategy. Each and every replica in the system is known to all other replicas. The C3 procedure uses an adaptive layer, which allows the dynamic switching between the different CCPs at runtime. Thus the reduction of operational costs and transaction response time is possible [17].

3.4.1. Transaction model:

An object-id is used for identifying an object uniquely for performing operations under transactions. Only read operations are included in the read-only transaction, where update transactions should contain minimum one update operation. In the transaction model of C3, provides a unique timestamp for transactions at the start and commit time based on their arrival order. The highest start timestamp is assigned to the transaction which started more recently and the highest timestamp for commit is the most recently committed transaction.

3.4.2. Implementation:

All the middleware components are implemented as web services and allow deployment in possible configurations. The components of C3 middleware are: (1)Transaction Manager: Manages every transactions and responsible for the implementation of C3 protocol. (2)Site Manager: provision of an abstract layer for the management of local data access.
(3)Timestamp Manager: provides timestamps for transactions based on the arrival order and the management of timestamps.
(4)Lock Manager: Is responsible for management of locks.
(5)Replica Manager: provides replica management.
(6)Freshness Manager: manages the freshness data.

Under logical architecture of C3, each replica includes a Transaction Manager and Site Manager. Moreover, each replica also includes a local datastore where the Site Manager utilizes the datastore for managing real data and Transaction Manager stores data regarding its functionality.Avoidance of Anomalies:

The transactions with read and write sets are required for avoiding anomalies under consistency mixes. The implementation of C3 consists of different types of CCPs, when the different concurrent transactions, access the same data item with different consistency levels for the reasons such as:

First, the design of the application supports the access of the same data item by transactions with different consistency levels. Second, consistency requirements will be different for different applications that use the same data [18]. Third, based on the cost model different replicas execute transactions adaptively that accesses the same data object [19]. The possible inconsistencies are:

(1)Inconsistencies arise because of the isolation level between transactions that run on same CCP.

(2)Inconsistencies arise because of the isolation level between transactions that run on different CCP.

(3)Data staleness is also a reason for the inconsistency.

We analyze these concurrency control protocols in Table 1.

Table 1: Comparison of different concurrency control protocols

Properties	SO-1SR	SI	SC	C^3
Definition	Integration of 1SR with the merits of cloud such as availability, scalability, and strong consistency of data.	SI is a concurrency control protocol. It avoids many inconsistencies, but allows write skew anomaly.	It is the form of eventual consistency. Data is accessed under sessions. In sessions system assures read your writes consistency.	It is implemented on the top of ISR, SC, SSI. It is based on the full replication and update anywhere approaches.
Consistency level	Strong consistency	Weak consistency	Read your writes consistency	Adaptive consistency
Scalability	Higher scalability	Higher scalability	Higher scalability	Adaptive scalability
Availability	Higher availability	Higher availability	Higher availability	Adaptive availability
Cost	Optimized cost	High penalty costs	Low cost	Low cost

4. CONCLUSION:

In this paper, the different concurrency controls in the encrypted cloud database such as SO-ISR, SI, SC and C3 is discussed. These protocols provide different data consistency levels at different costs. The concurrency and performance varies according to the concurrency protocols used in the cloud environment. The architecture which supports the distributed, concurrent and independent access to the encrypted cloud database is SecureDBaaS. SecureDBaaS uses the isolation mechanisms for providing concurrent access to its geographically distributed clients.

Acknowledgments:

I am thankful to my guide Dr. M. NewlinRajkumar& Co-guide Dr. V. Venkatesakumar for their guidance and encouragement for the paper work.

REFERENCES:

[1]L. Ferretti, M. Colajanni, and M. Marchetti, “Distributed, Concurrent, and Independent Access to Encrypted Cloud Databases,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 2, pp. 437–446, Feb. 2014.

[2]I. Fetai and H. Schuldt, “SO-1SR: towards a self-optimizing one-copy serializability protocol for data management in the cloud,” in Proceedings of the fifth international workshop on Cloud data management, 2013, pp. 11–18.

[3]C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, “Relational cloud: A database-as-a-service for the cloud,” 2011.

[4]K. Daudjee and K. Salem, “Lazy database replication with snapshot isolation,”inProceedings of the 32nd international conference on Very large data bases, 2006, pp. 715–726.

[5]T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, “Consistency Rationing in the Cloud: Pay only when it matters,” Proc. VLDB Endow., vol. 2, no. 1, pp. 253–264, 2009.

[6]I. Fetai and H. Schuldt, “Cost-based data consistency in a data-as-a-service cloud environment,” in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012, pp. 526–533.

[7]Y. Lu and G. Tsudik, “Enhancing data privacy in the cloud,” in Trust Management V, Springer, 2011, pp. 117–132.

[8]L. Ferretti, M. Colajanni, and M. Marchetti, “Supporting security and consistency for cloud database,” in Cyberspace Safety and Security, Springer, 2012, pp. 179–193.

[9]H. Hacigumus, B. Iyer, and S. Mehrotra, “Providing database as a service,” in Data Engineering, 2002. Proceedings. 18th International Conference on, 2002, pp. 29–38.

[10]K. P. Puttaswamy, C. Kruegel, and B. Y. Zhao, "Silverline: toward data confidentiality in storage-intensive cloud applications," in Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011, p. 10.

[11]L. Ferretti, F. Pierazzi, M. Colajanni, and M. Marchetti, "Security and confidentiality solutions for public cloud database services," in SECURWARE 2013, The Seventh International Conference on Emerging Security Information, Systems and Technologies, 2013, pp. 36–42.

[12]L. Ferretti, M. Colajanni, M. Marchetti, and A. E. Scaruffi, "Transparent Access on Encrypted Data Distributed over Multiple Cloud Infrastructures," in CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, 2013, pp. 201–207.

[13]J. G. U. Berkeley and others, "A Critique of ANSI SQL Isolation Levels," Online Verfügbar Http131107, vol. 65.

[14]A. J. Feldman, W. P. Zeller, M. J. Freedman, and E.W. Felten, "SPORC: Group Collaboration using Untrusted Cloud Resources.," in OSDI, 2010, vol. 10, pp. 337–350.

[15]M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and others, "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, 2010.

[16]W. Jansen, T. Grance, and others, "Guidelines on security and privacy in public cloud computing," NIST Spec. Publ., vol. 800, p. 144, 2011.

[17]C. Almond, "A practical guide to cloud computing security," White Pap. Accent. Microsoft, 2009.

[18]S. Hildenbrand, D. Kossmann, T. Sanamrad, C. Binnig, F. Faerber, J. Woehler, D. Kossmann, and D. Kossmann, Query Processing on Encrypted Data in the Cloud by. ETH, Department of Computer Science, 2011.

[19]Y. Sun, J. Zhang, Y. Xiong, and G. Zhu, "Data Security and Privacy in Cloud Computing," Int. J. Distrib. Sens. Netw., vol. 2014, pp. 1–9, 2014.