Reduction of Monetary Cost in Cloud Storage System by Using Extended Strict Timed Causal Consistency

Hesam Nejati Sharif Aldin\textsuperscript{a,∗}, Mostafa Razavi Ghods\textsuperscript{a,∗}, Hossein Deldari\textsuperscript{1}

\textsuperscript{a}Department of Computer Engineering, Mashhad Branch, Islamic Azad University, Mashhad, Iran

Abstract

Cloud storage systems have been introduced to provide a scalable, secure, reliable, and highly available data storage environment for the organizations and end-users. Therefore, the service provider should grow in a geographical extent. Consequently, extensive storage service provision requires a replication mechanism. Replication imposes many costs on the cloud storage, including the synchronization, communications, storage, etc., costs among the replicas. Moreover, the synchronization process among replicas is a major challenge in cloud storage. Therefore, consistency can be defined as the coordination among the replicas. In this paper, we propose an extension to the Strict Timed Causal Consistency (STCC) by adding the considerations for the monetary costs and the quantity of violations in the cloud storage systems and call it the Extended Strict Timed Causal Consistency (X-STCC). Our proposed supports monotonic read, read your write, monotonic write, and write follow read models by taking into account the causal relations between users’ operations, at the client-side. Besides, it supports timed causal at the server-side. We employed the Cassandra cloud database that supports various consistencies such as ALL, ONE, Quorum, etc. Our method performs better in reducing staleness rate, the severity of violations, and monetary cost in comparison with ALL, ONE, Quorum, and Causal.

Keywords: Replication, Causal Consistency, Timed Causal Consistency, Cloud Computing, Monotonic Read, Monotonic Write, Read your write, Write follow read

1. Introduction

Context. Big data involves the data generated from social networks, Internet of Things (IoT), multimedia, etc. over the Internet [1–3]. Big data helps researchers make precise and valuable decisions in their researches and applications such as business, science, and engineering [4]. Storage and data access are two of the major problems of big data.

Today, organizations apply cloud computing technology for storage and easy access to their data. Cloud computing is one of the most popular distributed systems that provide users with a pay-per-use model [5]. One of the cloud computing services is the Storage as a Service (SaaS). This service can easily solve big data storage problems [6].

Existing challenges. Cloud storage systems utilize the replication mechanism in order to avoid failure and improve their performance in data storage [7]. This mechanism places the replica at the nearest data-center to the user so that the user can easily access the replica [8]. Replication and synchronization among replicas lead to a consistency problem [9]. However, replication brings about the costs such as network overhead, bandwidth, storage space, etc. on cloud storage systems.

Replication is a key factor that needs consistency to work appropriately in cloud storage systems. Consistency has a variety of levels that vary from weak [10] to strong [11]. In fact, different criteria and services have been considered for data-sharing in the distributed systems, out of which, the five most important criteria are such as consistency, concurrency, availability, visibility, and isolation [12]. Consistencies are divided into two categories: data-centric and client-centric [9]. The choice of consistency level required by the cloud storage systems directly affects the monetary costs in them [13–18].

Strong consistency by the means of synchronous replications may introduce high latencies due to the cross-sites communication and therefore will significantly increase the monetary cost of the services. High latency causes a high monetary cost. This is due to the fact that the cost of leasing a VM-instance is proportional to the latency, which in turn affects the throughput of the system resulting in high run-time, in addition to the increased cost of both the storage (e.g. number of requests to the copies) and the communication cost (e.g. number of cross-sites communication) due to the synchronous cross-site replication [19, 20].

Moreover, high latency causes significant financial losses for service providers that use such storage systems. For instance, the cost of a single hour of downtime for a system doing credit card sales authorizations has been estimated to be between 2.2M$–3.1M$ [21].

Researchers have shown that there is a high degree of convergence among replicas in cloud storage systems with strong (data-centric) consistencies such as linear, sequential, causal, etc. [22–24]. Especially, to maintain convergence (i.e., use-
fulness), causal is the strongest implementable consistency in a highly-available system [25]. Therefore, they spend more time on the coordination process among the replicas in comparison with the other consistencies [20]. As a result, the system faces a high network latency in order to the coordination among replicas for all nodes and an increased synchronization and communication costs among them [19]. Additionally, the cloud storage systems ensure that the replicas do not face with the stale read values and severity violations [19, 26, 27].

Conversely, weak consistencies (client-centric), such as eventual [28], monotonic read [29], read your write [29], etc., have the least degree of convergence in the cloud storage systems [22, 23]. As a result, the time spent on the synchronization among replicas for all nodes is reduced, but the severity of the violations and staleness rate of the weak consistencies is greater than the other ones [20, 27, 30–32].

One of the most popular data-centric models is causal consistency. According to the Consistency, Availability, and Convergence (CAC) theorem [22], the degree of convergence in a system with causal consistency is higher. Furthermore, based on the Consistency, Availability, and Partition tolerance (CAP) theorem [33, 34], the causal also has a high data availability and network partition tolerance. These theories show the benefits of causal consistency over the other ones. Besides, the causal+ and timed causal consistencies [23, 35] have also improved the performance of the causal by applying the CAC and CAP theorems in the cloud environment [36].

Nowadays, the major needs of cloud storage systems are high convergence and availability, as well as reduced monetary cost, staleness rate, and severity of violations among replicas, etc. On this basis, researchers offer a combination of data-centric and client-centric models to meet a large portion of cloud storage needs.

In this paper, we present an extension to the Strict Timed Causal Consistency (X-STCC), as a data-centric model. At the server-side, the model supports the Timed Causal Consistency (TCC), and at the client-side, it supports the monotonic read, read your write, monotonic write, and write follow read. Based on the CAC and CAP theorems, this model provides a high degree of convergence and partition tolerance at the server-side with high availability at the client-side.

**Proposed solution.** The proposed X-STCC is a data-centric model. However, when a session is held between the user and the Cloud Service Providers (CSPs), then this model supports the monotonic read, read your write, monotonic write, and write follow read. Supports at the client-side. Besides, the user requests (user operations) are sent to the CSPs at the server-side. This model also supports the timed causal while the users register their requests with respect to the logical time on the Distributed User Operations Table (DUOT). It also analyzes the causality among the requests on the DUOT and sends them to CSPs. All servers have the same view of the causality relations between the requests and their event times.

The similarities between the timed causal and the monotonic read, monotonic write, read your write, and write follow read is in their applications. This similarity is based on the event time and the causal relations between the operations of a user or multiple users in the same session or different sessions.

We have created the DUOT in order to arrange the requests in the correct order. When the users submit their requests to the CSPs, they are first registered in the DUOT. Consequently, based on the user’s identification (User-ID) and the logical time of the request, the operation is executed. Finally, thanks to the X-STCC, all servers have the same view of the users’ requests executions. Also, we create an operations dependency graph according to the logged requests on the server to determine the relations between the operations of a user or a number of users to calculate the severity of the violations.

**Main contributions.** The goal of this study is to define a consistency that comes with a high system throughput and reduce monetary costs, staleness, and the severity of violations.

Our major contributions in this paper are as follows:

- We have proposed X-STCC as a hybrid consistency that supports the timed causal consistency at the server-side.
- We have presented X-STCC that supports the (monotonic read, read your write, monotonic write, and write follow read) at the client-side.
- The proposed X-STCC not only provides the users with a satisfactory level of data availability but also reduces the stale read rate and severity of violations.
- The reduction of latencies due to cross-site communication. Therefore, the reduction of the number of requests to the copies and the number of cross-sites communication due to the synchronous cross-site replication results in the reduction of monetary costs such as communication, storage, and instance costs.

**Experimental setup.** Cloud database systems such as Mongo DB [37], Hadoop [38] and Cassandra [39], etc., have proven to be effective to store large bulks of data and in service provision on a large geographic scale. Most of these systems, such as Amazon Dynamo [40–42], have chosen the eventual consistency as the most efficient consistency in which the replicas gradually converge. Each of the above mentioned cloud storage systems have a specific purpose and application. Apache Cassandra [39] is an open source cloud storage system used by the applications such as AppScale [43], Instagram, Facebook [44], etc. Our proposed method is implemented on a Cassandra cluster. We used the Complete Replication and Propagation Protocol (CRP) and used NetworkTopologyStrategy in Cassandra to implement this protocol.

The remainder of this paper is organized as follows. In section 2, different consistency models are reviewed. The proposed model and the studied scenario are introduced in section 3. Section 4 deals with the evaluation of the proposed model and its results in comparison with other consistencies in Cassandra cluster. And finally, section 5 concludes the paper.

2. Related works

Depot is a cloud storage system with the Fork-Join-Causal (FJC) consistency to secure the system from malicious clients
and servers [16]. The FJC is a hybrid consistency and weaker than the causal. This model despite the malicious nodes, and ensures that the healthy node has the latest update. Our proposed consistency is a data-centric model that also supports the client-centric model. Additionally, this model reduces the staleness rates, monetary costs, and the severity of violations.

Harmony is offered as an adaptable consistency in Cassandra and provides consistency levels based on the application requirements [27]. This consistency can elastically tolerate the stale read value or significantly reduce the staleness rate by increasing or decreasing the number of replicas involved in the read operations. Our method significantly reduces the staleness rate at the client-side. At the server-side, all servers have the same view of users’ requests and decrease the severity of violations.

Bismar, a new consistency performed in Cassandra that argues the monetary costs must also be taken into account when evaluating or selecting consistency levels in Cassandra’s storage system. Accordingly, it has defined a new metric called consistency-cost. Therefore, this adaptive model has been introduced as an economic consistency model [19]. Our proposed consistency also pursues Bismar goals, with the exception that our consistency is a data-centric and supports both server and client sides, but Bismar is an adaptable consistency.

Eventual is a client-centric model that provides a relatively weak level of consistency and does not guarantee a reduction in the severity of violations. However, a cloud storage system with eventual consistency ensures that if the system does not have a continuous update, the system converges to the steady-state and all replicas are adapted [30]. In contrast, our approach reduces the staleness rate and the severity of violations at the client-side.

A combination of causal, monotonic read, and read your write consistencies which are designed for the cloud storage systems [31]. The system uses monotonic read and read your write to investigate the severity of violations in the implementation of read operations and local auditing. Similarly, it uses the causal in the global auditing. We have provided X-STCC at both server and client sides. Moreover, our proposed applies a global auditing schema in order to the reduction of staleness rate and the severity of violations. Furthermore, Our proposed supports monotonic read, read your write, monotonic write and write follow read.

Weak causal consistency strives to maintain the causal relations among operations. This model avoids the reduction of the convergence degree among the replicas that have been introduced in the causal convergence [45]. Moreover, this model is another variant of the causal consistency that provides both the weak causal and convergence.

The cloud storage system has been implemented in the causal consistency based on the partial and full replication protocols with respect to which the performance of the system is analyzed [46].

3. Proposed method

In this paper, our proposed method is implemented on a Cassandra cluster. We have evaluated our proposed method and the performance of different consistency levels in this environment. Our suggested method consists of 6 sections as follows:

As shown in Fig. 1, we have elaborated upon our proposed scenario in Section 3.1. This scenario illustrates how the proposed method behaves. We also introduced assumptions for the ease of implementation of our method. In Section 3.2, we present the DUOT which contains the users’ operations in which the information is registered in the table with a timestamp for each of their requests to the server. In Section 3.3, our strategy is to analyze the causal relationships among the operations listed in the DUOT at the client-side and the server-side. In Section 3.4, our proposed X-STCC is presented. Our proposed method runs on the operations listed in the DUOT. In section 3.4.1, we create an Operation Dependency Graph (ODG) from the operations listed in the DUOT. Based on the ODG, the causality relations between the operations are determined. We also use the garbage collection mechanism to remove operations performed in the DUOT. In Section 3.5, we will make a comparison between our proposed method and the other consistencies in Cassandra to evaluate the staleness rate, monetary costs, and severity of violations.

3.1. Scenario

Consider Fig. 2 in which several cloud servers in CSPs are available for Bob and Alice. Bob posts his tweet by connecting to the CSPs server. When Bob reconnects to the same cloud server or moves to another location and connects to another server, the following situations may occur:

- Bob might see his previous or the most recent tweet when connected to the new CSPs server.
- Bob retweets when connected to the new CSPs server.
- Bob sees the least tweet by connecting to a new CSPs server.
- After might read his previous tweet, retweet again upon connecting to a new CSPs server.
- Bob might post a tweet while connected to the CSPs server. Then, Alice reads the tweet’s content and posts a comment in response to his tweet.

In our previous work [47], we considered some assumptions in our implementation of the proposed method. i.e. the STCC. Similarly, in this paper, we intend to use the logical time as the timestamp in the DUOT. Besides, we used the NetworkTopologyStrategy in order to execute CRP in Cassandra. We considered three factors for the write operations that determine the causal relations among the operations listed in the DUOT and represent the consistency levels at the client and server sides.
1. We have presented our proposed scenario

2. Create a Distributed Users Operations Table (DUOT)
   - All clients inserted the operations in DUOT
   - Each client inserted the operations in the DUOT
   - Registered each operation with a timestamp

3. Auditing Strategy

4. X-STCC implemented on the operations listed in DUOT

5. Create the Operations Dependency Graph (ODG)
   - We used a garbage collection mechanism on DUOT

6. Running micro benchmark (YCSB)
   - The staleness rate was calculated
   - The monetary cost was calculated
   - The violations were calculated

End

Figure 1: Process steps of the proposed method (Extended Strict Timed Causal Consistency).

3.2. Distributed Users Operations Table (DUOT)

Each user has access to the distributed user operations table and records its operations there. Each operation recorded in this table contains elements such as the type of operation, the user ID of the applicant, the operation, the base name, which is the universal logical clock vector. The insertion of operations in this table is such that the user is entered with the type of operation and the source of the operation. For example, the operation $R(x)a$, indicates the read operation of value $a$ from the resource $x$ or the operation $W(x)a$, on the resource $x$, the read/write values can be either unique or common. The logical time is replaced with the physical clock in the DUOT. Therefore, users can have the same view of the order of the execution of operations.

In our proposed system [47], the audit is carried out based on a global strategy. As shown in Table 1, each client before executing the operations, registers them in the DUOT. All clients access the DUOT simultaneously and the availability to the DUOT is based on the timed sequential consistency which is used to manage the operations on the data in this table. The insertion of each client’s operation in this table is performed with a timestamp to arrange the view of the clients’ operations in the DUOT based on the timestamp. The audit is carried out glob-
users. Logical time increases as the user operations are be-

ally by all clients in order to execute the operations correctly. Each client’s operations will be viewed by its latest operations as well as the other clients’ operations on the shared resource.

Fig. 3 illustrates the read/write operation on a common resource. Logical time increases as the user operations are being performed. In case the first operation is carried out by $U_1 (W(x)a)$, the $U_i$ requests for the write operation of value $a$ on the common resource $x$ and the logical time $< 1,0,0 >$ in the DUOT will be registered. This logical time is the same timestamp which is stored in the DUOT and the other users like $U_2$ or $U_3$ register their requests in the DUOT as well.

### 3.3. Auditing strategy

In this strategy, each user inserts its operations in DUOT independently. Then, this required operations with his/previous operation, and other users required operations will be ana-
yzed on the same resources. In case the causality relation between the user’s new operations with his/her previous operations or the other users’ operations is analyzed. This strategy is performed globally and the user operations are merely the read/write operations. Consistency in the distributed storage systems is one of the major challenges as the accessibility to a resource by multiple users is performed simultaneously and therefore it will lose consistency in the execution of the operations. The considered operations are recorded in the DUOT based on the timestamp by the users. These operations might be recorded whether by a client or different clients. These executed operations are considered as follows:

\[
(O_1 = r(x)a) \land (O_2 = r(x)b) \quad (1a)
\]

\[
(O_1 = w(x)a) \land (O_2 = r(x)a) \quad (1b)
\]

\[
(O_1 = w(x)a) \land (O_2 = w(x)b) \quad (1c)
\]

\[
(O_1 = r(x)a) \land (O_2 = w(x)b) \quad (1d)
\]

In eq. 1a the read operation of the value $a$ from resource $x$ on server $S_i$ by $O_1$ and after that the read operation of the value $b$ from resource $x$ on server $S_j$ by $O_2$ at the client-side which indicates the execution of the operations on a resource with the monotonic read. Then, the read value $a$ on resource $x$ on server $S_i$ and the write value $a$ on resource $x$ on server $S_j$ before reading value $b$ on resource $x$ on server $S_j$.

In eq. 1b the write operation of the value $a$ from resource $x$ on server $S_i$ by $O_1$ and after that the read operation of the value $a$ from resource $x$ by $O_2$ on server $S_j$ at the client-side which indicates the execution of the operations on a resource with the read your write. Then, the write value $a$ on resource $x$ on server $S_j$ before reading value $a$ on resource $x$ on server $S_j$.

In eq. 1c the write operation of the value $a$ from resource $x$ by $O_1$ and after that the write operation of the value $b$ from resource $x$ by $O_2$ at the client-side which indicates the execution of the operations on a resource with the monotonic write. Then, the write value $a$ on resource $x$ on server $S_i$ and the write value $a$ on resource $x$ on server $S_j$ before writing value $b$ on resource $x$ on server $S_j$.

In eq. 1d the read operation of the value $a$ from resource $x$ by $O_i$ on server $S_i$ and after that the write operation of the value $b$ from resource $x$ by $O_2$ on server $S_j$ at the client-side which indicates the execution of the operations on a resource with the write follow read. Then, the write value $a$ on resource $x$ on server $S_i$ and the write value $a$ on resource $x$ on server $S_j$ before reading value $b$ on resource $x$ on server $S_j$.

The above-mentioned operations at the server-side indicates the resource has causal consistency.

We have considered the operations by the user $C_i$ based on the timestamp $T_{D_i} < T_{O_2}$ stored in the DUOT. Consequently, the user’s new operations are compared with its previous operations and the operations of the other users. The comparison criteria are as follows [49]: Causality between write operations on the same resource ($O_1 \rightarrow O_2 \Rightarrow O_1 S \rightarrow O_2$)

The middle operation $i, \exists_{o_{1}, \cdots, O_{2}}$ is the causal relation between operations $o_1$ and $o_2$ by one or two different clients.

The execution of the operations $o_1$ and $o_2$ by the same client $\exists_{o_{1}, \cdots, O_{2}}$

The operations are performed by the same client or two different clients concurrently. $\exists_{o_{1}, \cdots, O_{2}}$ $O_1 \rightarrow O_2 \rightarrow O_1 \Rightarrow O_1 \parallel O_2$, the operations which do not have the causality are executed at the same time.
Causal consistency could be defined using Rule 1, this rule indicates the behavior and the performance of this consistency model on the shared resource [49]:

\[
O_{x_1}, O_2 \in D \land \neg O_2 \Rightarrow O_1 \Rightarrow O_2
\]  

(Rule 1)

Rule 1 presents that in case the execution of the operation \(O_1\) evokes operation \(O_2\) on the replica existing in server \(S_{x_1}\), then the other processes should also first observe the operation \(O_1\) on their own server and then the operation \(O_2\) [49]. In other words, the execution process is performed according to the cause and effect relation between the operations.

The strategy in our previous work [47] was to analyze each client by inserting its operations in the DUOT based on its user-ID on the shared resource \(x\), the \(T_{O_1} < T_{O_2}\) timestamp, and the type of the operations are analyzed with their per-operation.

In case conflicting operations, e.g. \(O_1 = \text{write}(x)\) and \(O_2 = \text{read}(x)\) are preformed by the same user \(U_i\) on the same resource \(x\), then the X-STCC should be implemented at the client-side. Moreover, if conflicting operations are executed by different clients \(U_i\) and \(U_j\) in the DUOT, the X-STCC should be implemented at the Server-side. Finally, if the shared source is not the same, e.g. \(O_1 = \text{write}(y)\) and \(O_2 = \text{read}(x)\) a, or non-conflicting operations e.g.: \(O_1 = O_2 = \text{read}(x)\) a, can be executed simultaneously.

3.4. Extended Strict Timed Causal Consistency

Our previous work performed at both client and server sides [47]. As shown in Fig. 4, our proposed method supports four client-centric models at the client-side. Furthermore, it supports data-centric models based on the causal relations among the operations listed in the DUOT at the server-side.

Moreover, at the client side, X-STCC supports four client-centric consistencies i.e.: the monotonic read, read your write, monotonic write, and write follow read based on the user \(U_i\), type of operation (read/write), and operation event time. Also, at the server-side X-STCC supports TCC based on the aforementioned criteria.

In Fig. 4 we showed that the event time of an operation is important and the operations \(O_1\) and \(O_2\) are either new or old with respect to the occurrence time as an entry data. Also, we showed that the operation as well as its corresponding timestamp as a logical time are registered in the DUOT. Besides, X-STCC executes on the operations listed in the DUOT and analyze them based on the following five conditions:

- If the operations \(O_1\) and \(O_2\) are registered in the DUOT by the same client \(C_i = C_j\) on the same resource \(x = y\), and the operation \(O_1\) happened before \(O_2\), one of these four phases might be happen:
  - if the operations \(O_1 = \text{read}(x)\) and \(O_2 = \text{read}(x)a\), then phase a1: monotonic read is correct.
  - if the operations \(O_1 = \text{write}(x)a\) and \(O_2 = \text{write}(x)b\), then phase a2: monotonic write is correct.
  - if the operations \(O_1 = \text{write}(x)a\) and \(O_2 = \text{read}(x)a\), then phase a3: read your write is correct.

if the operations \(O_1 = \text{read}(x)a\) and \(O_2 = \text{write}(x)b\), then phase a4: write follow read is correct.

- If the operations \(O_1\) and \(O_2\) are registered in the DUOT by different clients \(C_i <> C_j\) on the same resource \(x = y\), and the operation \(O_1\) happened before \(O_2\), phase b1 should be executed and timed causal is correct.

- If the operations \(O_1\) and \(O_2\) are registered in the DUOT by the same client \(C_i = C_j\) on the same resource \(x = y\), but the operation \(O_1\) did not happen before \(O_2\), phase b2 should be executed.

Finally, we check the number of operations that are analyzed, and if an operation could not be the latest one, then X-STCC executes again on the operations listed in the DUOT. Otherwise, our proposed method could be terminated.

The performance of our proposed method is shown in Fig. 5. In the following items we will elaborate upon our suggested consistency:

- **Client-side**

  **Monotonic read:** As it can be seen in Fig. 5, our proposed consistency first detects if the user \(U_i\), wishes to read the value \(b\) correctly at time \(\theta\) when accessing the server \(S_j\). The user should write the value \(a\) at time \(\epsilon\) on the server \(S_i\), and at time \(\Delta + \epsilon\) read the value \(a\) at the server \(S_j\). Hence, the value \(a\) is written at time \(\delta\) on the server \(S_j\). Therefore, at the time \(\Delta + \delta\) the value \(b\) is written on the same server. Finally, the user can read the value \(b\) at time \(\theta\) from the server \(S_j\). Given the causality between operations, the X-STCC guarantees monotonic read consistency.

  **Monotonic write:** According to Fig. 5, our proposed method detects if the user \(U_i\), wants to write the value \(b\) correctly at time \(\theta\) to the server \(S_j\). Hence, the value \(a\) is written at time \(\epsilon\) on the server \(S_i\). Then, the value \(a\) at time \(\delta\) is written on the server \(S_j\). Finally, the value \(b\) is written on the server \(S_j\) at time \(\theta\) by the user \(U_i\). Given the causality between operations, the X-STCC guarantees monotonic write consistency.

  **Read your write:** As it can be seen in Fig. 5, our proposed consistency first detects if the user \(U_i\), wishes to read the value \(b\) correctly at time \(\theta\) when accessing the server \(S_j\). The user should write the value \(a\) at time \(\epsilon\) on the server \(S_j\). Hence, the value \(a\) is written at time \(\delta\) on the server \(S_j\). Therefore, at the time \(\Delta + \delta\) the value \(b\) is written on the same server. Finally, the user can read the value \(b\) at time \(\theta\) from the server \(S_j\). Given the causality between operations, X-STCC ensures read your write consistency.

  **Write follow read:** According to Fig. 5, our proposed method detects the user \(U_i\), if the user wants to write value \(b\) correctly at time \(\theta\) by accessing server \(S_j\), then the
timestamps inserted with operations in DUOT

Execute X-STCC

Phase a1: Monotonic Read
Phase a2: Monotonic Write
Phase a3: Read Your Write
Phase a4: Write follow Read

Figure 4: The flowchart of our proposed method.
value \( a \) at time \( \epsilon \) on the server \( S_i \) should also be written. The value \( a \) should be read at time \( \Delta + \epsilon \) from the server \( S_j \). Then, at time \( \delta \), the value \( a \) should be written on the server \( S_j \), and finally the value \( b \) at time \( \theta \) should be written on the server \( S_j \). Given the causality between operations, X-STCC ensures write follow read consistency.

- **Server-side**

  **Timed causal**: The last consistency supported by our proposed method is the TCC. As can be seen in Fig. 5, the user \( U_i \) writes the value \( a \) on the server \( S_i \) at time \( \epsilon \). Then user \( U_j \) wants to write the value \( b \) at time \( \theta \) on the server \( S_j \). Operations are executed correctly when the user \( U_j \) at time \( \delta \) writes the value \( a \) on the server \( S_j \). Then the \( U_j \) writes the value \( b \) on the server \( S_j \). Hence, the X-STCC guarantees the TCC.

### 3.4.1. Operations Dependency Graph (ODG)

In our implementation of the proposed method [47], it is necessary to determine which process observes the write operation. Therefore, an ODG is needed to determine which operation is related to other operations. In our proposed consistency timestamp is applied to build the ODG which helps us to implement our proposed consistency easily.

As shown in Fig. 6, the graph shows the dependency of operations, type of operations, event time of operations, and the type of dependency between them. We used three edges, Timed, Causal, and Data in this graph [31]. Timed edge shows the temporal priority between user operations, the Causal edge shows the causal relations between the user operations concerning himself or the other users, and the Data edge shows the causal relations between the data values updated by the same users and read by the others.

All these three edges help our proposed consistency support monotonic read, read your write, write follow read, monotonic write, at the client-side, and the timed causal consistency at the server-side.

- **Client-side**

  **Monotonic read**: User \( U_1 \), given the causal relations among the operations \( w(x)a \) and \( r(x)b \), reads the value \( b \) when value \( a \) is written before value \( b \). As shown in Fig. 6, the X-STCC guarantees that the monotonic read is not violated for the user \( U_1 \).

  **Monotonic write**: User \( U_1 \) given the causal relations among the operations \( w(x)a \) and \( w(x)b \). The user \( U_1 \) can write the value \( b \) without violations, if value \( a \) is written before value \( b \). As shown in Fig. 6, the X-STCC guarantees that the monotonic write is not violated for the user \( U_1 \).

  **Read your write**: User \( U_2 \) given the causal relations among the operations \( w(x)d \) and \( r(x)b \), the value \( d \) can read without violations when its value is written to the other servers. As shown in Fig. 6, the X-STCC guarantees the read your write for the \( U_2 \) user is not violated.

  **Write follow read**: User \( U_2 \) given the causal relations among the operations \( r(x)d \) and \( w(x)c \), value \( c \) is written without violations when value \( d \) is written before, then value \( d \) is read. Finally, the value \( c \) is written without violation. As shown in Fig. 6, the X-STCC guarantees that the write follow read is not violated for the user \( U_2 \).

- **Server-side**

  **Timed causal**: Based on the causal relations among the operations \( w(x)b \) and \( w(x)d \) performed by users \( U_1 \) and \( U_2 \), respectively. As a result, the value \( b \) should be written, then user \( U_2 \) reads it, and then writes the value \( d \). As shown in Fig. 6, the X-STCC guarantees that the timed causal is not violated.

3.5. Estimation of Stale Read Rate and Monetary Cost

#### 3.5.1. Estimation of Stale Read Rate

In our previous work [47], the staleness rate was calculated concerning the execution of the write/read operations rate on Cassandra. The calculation of this probability requires an examination of the network latency and access pattern in the cloud storage systems. Network latency is a key factor in calculating the staleness rate. Network latency is based on the period that it takes for a replica to be propagated to the other nodes. Moreover, the access pattern to the replica depends on two modes of write and read operations. Furthermore, the staleness rate is calculated based on two factors, if the client requests to the server to read a replica: a. The client will send its read request from the replica to the server while the replica is being updated.
locally. b. The client will send its read request from the replica to the server while the other replicas are being updated globally. Finally, the staleness rate is calculated based on the exponential distribution function [50]. (interested readers could refer to Appendix 5 for more details).

3.5.2. Monetary Cost

The synchronization among replicas imposes on a network latency in the cloud storage systems. Moreover, the high/low network latency depends on the different consistency levels in the cloud storage systems. The network latency affects the system throughput at run-time, as well as increasing storage cost (e.g., number of requests to the storage servers), and communication cost (e.g., cost of communication among VMs). Therefore, there is a trade-off between monetary cost and network latency. In this case, it can be said that if the network latency is increased, then the monetary costs will grow significantly.

For example, with respect to the monetary costs, the network latency is increased in the cloud storage system with strong consistency. Besides, by using strong consistency, the performance of the system would be more than the cloud storage system with the eventual. Consequently, the monetary costs in the cloud storage systems with strong consistency are sharply increased. In contrast, by applying the eventual consistency in the system, monetary costs will drop significantly. However, the risk of staleness rate sharply increases in the cloud storage system.

Strong consistencies like linearize increase the number of access requests the replicas. Moreover, the number of replicas that have involved in the replication process is increased. Besides, high network latency affects the execution time of the operations in the replica. Hence, by increasing the consistency levels, the network traffic grows rapidly. Therefore, more monetary cost should be paid to have a broader network bandwidth. Also, by increasing the consistency levels, the number of requests to the storage devices is increased which directly affects the storage cost.

In this paper, we considered three factors to calculate the monetary costs imposed on the cloud storage systems, these factors include a. The processing unit cost which includes the CPU, RAM on the virtual machines’ rent (e.g., the cost of VMs per hour to pay for sample medium on the Amazon EC2 is $0.0464). b. Storage cost which includes the amount of memory leased on 1GB per Month basis, and the number of I/O requests send/receive to the storage devices (e.g., the cost of 1GB memory per month to pay for Amazon EBS, $0.010). c. Network costs are based on the type of resource services, and the data transmission among nodes. In general, the internal communications in a data-center are much more expensive than external ones among data-centers. Finally, in our proposed method, we used the monetary cost model to calculate the cloud storage systems’ cost [19]. (interested readers could refer to Appendix 5 for more details).

4. Experimental setup

We evaluate our proposed consistency on three Cassandra clusters which are presented in Fig. 7. In these three clusters, a total of 24 nodes are applied. We dedicated 2 cores, and 4GBs of memory to each node. We also dedicated 12TiBs of memory to the storage devices in the Cassandra cluster. To be more specific, the share of each cluster is 4TiBs and the share of each node in it is 512GiBs. The local network is the Gigabyte Ethernet and the network connection among three data-centers is provided by Cisco routers. The average intra-data-center round trip latency is 0.115ms, whereas it is 45.7ms among three data-centers. The NetworkTopologyStrategy is applied as the replication strategy. By selecting this strategy, the data are stored in all clusters and racks. We employed the Cassandra-3.11.4 with the replication factor of 12 replicas: 4 replicas are located in each data-center.

4.1. Micro Benchmark

We conducted our experiments on cloud storage systems using the YCSB benchmark [51]. This benchmark shows the current services of the cloud servers [52]. This benchmark has been extended for the open source databases such as MongoDB [53], Hadoop HBase [38], and Cassandra [39]; among which different workloads with different read/write operations are available to be used by our proposed method.

We have applied the YCSB 0.14.0 for the Cassandra in order to analyze with different consistency levels during run-time. This benchmark has variant workloads (workload-A, workload-B, etc.) which can be varied. In workload-A is also called read-heavy, 50% of the operations are read and the half are write operations. In workload-B is also called write-heavy, 5% of the operations are read and 95% of the operations are write operations. In our experiments, these workloads consist of 8 million operations and 5 million rows with a total of 18.65GB data after replication. Besides, we have executed the workload-A and workload-B on 24 nodes from three different data-centers. This benchmark has been executed for 20 times on the ONE, Quorum, ALL, causal and our proposed X-STCC consistency levels (interested readers could refer to [47] for more details).

4.2. Evaluation

We used Cassandra-3.11.4 to implement the X-STCC and causal. In Cassandra, there are some consistencies such as Quorum, ONE, ALL, etc. In this paper, we used YCSB to evaluate the performance of the mentioned consistencies. The system runs the workload using the YCSB Benchmark. The stale-
ness rate is based on the rate of read/write operation at run-time, monetary costs (instances’ cost, storage cost, and network cost), and the severity of violations which are analyzed in order to be showed the dynamics of the system (e.g., the system throughput and read/write rates at run-time), we ran the workload-A on different number of threads (1, 16, 64, and finally 100 threads). Finally, with respect to the parameters listed above, we compared the X-STCC with Quorum, ONE, ALL, and causal (the interested read could refer to [47] for more details).

4.2.1. Throughput

Workload-A runs with 1, 16, 64, and 100 threads in a system with 24 nodes, and results are illustrated in Fig. 8. System throughput is one of the subjects that we investigated by considering different consistency levels mentioned in the previous section. Throughput is a ratio that shows the number of consistently executed operations in a second based on the promised consistency level. Workload-A is based on the number of threads that a client executes during the workload process. As can be seen in Fig. 8, the system throughput is shown considering workload-A in 24 nodes.

It can be clearly seen that the system throughput has an increasing trend up to 64 threads with the ALL, ONE, Quorum, causal, and X-STCC consistencies. However, as the number of nodes increases our proposed method slows down. Our X-STCC has shown better performance than ALL, ONE, Quorum, and causal.

X-STCC has improved the system throughput in comparison with ONE, 19%, Quorum, 23%, ALL 31% and Causal 14%. This improvement in the system throughput is due to the effect of workload-A which includes 50% read operations and 50% write operations; in our proposed method, consistency is important in an operation which there is a cause and effect relation within the operation.

Workload-B runs with 1, 16, 64, and 100 threads in a system with 24 nodes and results are illustrated in Fig. 9. It can be clearly seen that the system throughput has an increasing trend up to 64 threads with the ALL, ONE, Quorum, causal, and X-STCC consistencies. However, as the number of nodes increases our proposed method slows down. Our X-STCC has shown better performance than ALL, ONE, Quorum, and causal.

X-STCC has improved the system throughput in comparison with ONE, 13%, Quorum, 18%, ALL 26% and Causal 9%. This improvement in the system throughput is due to the effect of workload-B which includes 5% read operations and 95% write operations; in our proposed method, consistency is important in an operation which there is a cause and effect relation within the operation.

4.2.2. Staleness Rate

Fig. 10 shows the staleness rate based on workload-A with 24 nodes. Besides, concerning different consistency levels, the staleness rate of each consistency has changed. They also state that although the staleness rate in ALL is significantly less, it has the least system throughput in comparison with the other consistencies. Therefore, ALL can be neglected when there is increased system throughput. But, our X-STCC has decreased the staleness rate significantly in comparison with Quorum, ONE, and causal.

In our experiments, ONE, with more than 80%, has the highest staleness rate in comparison with the other consistencies. Using X-STCC the staleness rate of the system with 24 nodes decreases for approximately 70% in workload-A in comparison with ONE. Moreover, comparing with Quorum the staleness rate has about 5% reduced in workload-A by using X-STCC. Also, in comparison with causal the staleness rate drops down almost 25% when using X-STCC.

Although the ALL consistency has shown the best perfor-
performance to the staleness rate in comparison with other consistencies, it imposes the most monetary cost in the cloud storage systems.

Fig. 11 shows the staleness rate based on workload-B with 24 nodes. Besides, concerning different consistency levels, the staleness rate of each consistency has changed. They also state that although the staleness rate in ALL is significantly less, it has the least system throughput in comparison with the other consistencies. Therefore, ALL can be neglected when there is increased system throughput. But, our X-STCC has decreased the staleness rate significantly in comparison with Quorum, ONE, and causal.

In our experiments, ONE, with more than 20%, has the highest staleness rate in comparison with the other consistencies. Using X-STCC the staleness rate of the system with 24 nodes decreases for approximately 11% in workload-B in comparison with ONE. Moreover, comparing with Quorum the staleness rate has about 3% reduced in workload-B by using X-STCC. Also, in comparison with causal the staleness rate drops down almost 10% when using X-STCC.

Although the ALL consistency has shown the best performance to the staleness rate in comparison with other consistencies, it imposes the most monetary cost in the cloud storage systems. Moreover, reducing read operations in workload-B which has reason the reducing staleness rate by running its.

4.2.3. Violations

Fig. 12 shows the severity of violations based on workload-A with 24 nodes with different consistency levels. In our experiment, ONE has the most severe violations, with over 55% of in replicas in comparison with other consistencies. The reason for this is that in this consistency the least number of replicas are involved in the replication mechanism. In contrast, ALL comes without any severity of violations. This is because in this consistency all replicas are involved in the replication process.

In our experiment, the proposed X-STCC reduces the severity of violations among replicas for about 37% in comparison with the ONE consistency. Our X-STCC reduces the severity of violations for about 10% and 15% in comparison with the Quorum and causal consistencies respectively.

Fig. 13 shows the severity of violations based on workload-B.
with 24 nodes with different consistency levels. In our experiment, ONE has the most severe violations, with over 20% of in replicas in comparison with other consistencies.

Furthermore, the proposed X-STCC reduces the severity of violations among replicas for about 10% in comparison with the ONE consistency. Our X-STCC reduces the severity of violations for about 5% and 7% in comparison with the Quorum and causal consistencies respectively.

The communications among the replicas in their replication mechanism during the execution of the operations play the most pivotal role in the severity of violations. Therefore, by increasing the communications among the replicas their cost sharply grows. Besides, the number of replicas involved in the replication process significantly increases and therefore, the storage cost grows as well.

### 4.2.4. Monetary Cost

Table 2 describes the monetary costs, hourly rental of virtual machines on the Amazon EC2, rental storage unit per month on the Amazon EBS, amount of I/O requests, and the cost of communications. As stated in Section 3.5.2, the monetary costs include the cost of the virtual machine instance, the communication cost between nodes and data-centers, and the storage cost. The sum of these costs is the monetary cost imposed on the cloud.

As shown in Fig. 14, ALL imposes the highest monetary cost to the cloud. Our method has reduced it compared to ALL for $458.8, ONE $16.9, Quorum $324.25 and the causal for $356.75.

Compared to all consistencies in the Cassandra, the ALL consistency has shown the best performance thanks to its potential for the staleness rate and severity of violations; however, it imposes a significant amount of monetary cost to the cloud storage systems. Therefore, our proposed method performs better than the other consistencies thanks to its severity of violations, staleness rate, and the lowest monetary cost in comparison with others.

#### 4.2.5. Resource cost

Fig. 15 shows the details of system costs spent on different levels of consistency. The ALL consistency has spent most of its time on the sample VMs, network connections between the nodes, and data storage. With respect to the cost of sample VMs, our proposed method costs approximately 12% less than the ONE, and Quorum for approximately 20% and for the causal about 25%. In Fig. 15, in terms of the network communications, the X-STCC costs approximately 15% less than the ONE, and with respect to Quorum about 25% and the causal approximately 35%. It also has reduced storage costs for nearly 20% in comparison with the Quorum and 18% compared to the causal. Whereas, the storage costs in the ONE consistency decrease, as less replicas are involved in the replication process.

### 5. Conclusion

The monetary cost in cloud storage systems is one of the key factors in the determination of the consistency levels used by the CSPs. The servers often look for the consistencies that satisfy theirs as the end-users needs and reduce the costs of their service provisions to users. In this article, we showed that our proposed consistency supports the monotonic read, monotonic write, read your write, and write follow read at the client-side, and the timed causal consistency at the server-side. It might also reduce the monetary costs imposed on the cloud storage system by reducing the sum of instance, storage, and the network’s costs. It reduces the staleness rate and the severity of violations as well. Additionally, our method has a higher operational throughput than the other consistencies.

In the future, we would like to extend the proposed method by reducing the staleness rate as well as the severity of violations by improving the Quality of Service (QoS) in the cloud environment to have a better Service Level Agreement (SLA).

### Appendix A: Stale read calculations

If the start of \( X_i \) happens between the start of the last writing action \( X_w \) and the termination time of data propagation to other

---

Table 2: Pricing schemes used in our evaluation.

| Comp. unit | Storage unit | Storage Req | Intra Comm. | Inter Comm. |
|------------|--------------|-------------|-------------|-------------|
| $0.13/mo   | $0.01/GB     | $0.10/GB    | $0.10/GB    | $0.10/GB    |

Figure 14: Monetary costs of different consistency levels.

Figure 15: Resource cost of different consistency levels.
copies, the value may be called stale. This condition will be repeated the same way for all other write operations happening in the system. \( T_p \) is the time needed for replicate write operation or updating all replicas. Transaction inputs are generally the same as Poisson’s distribution process [50]. It is assumed that inputs of write and read are intended to Poisson’s distribution of parameters \( \lambda_r \) and \( \lambda_w \). These parameters are changed dynamically during storage system monitoring and running of incoming write or read calls.

The distribution of the waiting period between two inputs with Poisson’s distribution is exponential. Random variables \( X_r \) and \( X_w \) are read and write time with exponential distribution of parameters \( \lambda_r \) and \( \lambda_w \). The possibility that the next reading returns an stale value is calculated from the following simplified formula:

\[
Pr(\text{Stale}^{\text{read}}) = \sum_{n=0}^{\infty} \left( \frac{\lambda_w}{\lambda_r} \times Pr(X_r < X_w) + \frac{\lambda_r}{\lambda_w} \times Pr(X_w < X_r + T + T_p) \right) \tag{1}
\]

All writing times that may happen in the system have exponential distribution function. Duration of writing operation occurrence is shown by exponential distribution; sum of \( X_w \) with written gamma parameters of \( i \) and \( \lambda_w \).

All recorded times for write operation follow exponential distribution. Sum of \( X_w \) for all write operations follow gamma distribution of parameters \( i \) and \( \lambda_w \). Therefore, the probability in formula 1-8 is as follows:

\[
Pr(\text{Stale}^{\text{read}}) = \sum_{n=0}^{\infty} \left( \frac{\lambda_r}{\lambda_w} \times \int_0^T f_i(t) \lambda(t + T + T_p) - \lambda(t) \, dt \right) + \frac{\lambda_w}{\lambda_r} \times \int_0^T f_i(t) \lambda(t + T) - \lambda(t) \, dt \tag{2}
\]

Time \( T \) for local write is negligible compared with time \( T_p \) so we put it zero. The following probability shows the simple replacement of mass function of Poisson’s distribution probability and cumulative distribution function of exponential distribution:

\[
Pr(\text{Stale}^{\text{read}}) = \sum_{n=0}^{\infty} \left( \frac{\lambda_w}{\lambda_r} \times \int_0^T e^{-\lambda t} \lambda(t + T + T_p) - e^{-\lambda t} \, dt \right) + \frac{\lambda_r}{\lambda_w} \times \int_0^T e^{-\lambda t} \lambda(t + T) - e^{-\lambda t} \, dt \tag{3}
\]

Finally, the probability of the next read to be an old value is calculated from the following simplified formula:

\[
Pr(\text{Stale}^{\text{read}}) = \frac{(N-1)(1-e^{-\lambda T_p})(1+i)\lambda_w}{N\lambda_r\lambda_w} \tag{4}
\]

**Appendix B: Monetary cost calculation**

Formula .5 presents the overall cost for geo-replicated based services for a given consistency level \( cl \). Essentially, this cost is the combination of the VM instances cost \( Cost_{\text{inst}}(cl) \), the back-end storage cost \( Cost_{\text{st}}(cl) \), and network cost \( Cost_{\text{np}}(cl) \)

\[
Cost_{\text{cl}} = Cost_{\text{inst}}(cl) + Cost_{\text{st}}(cl) + Cost_{\text{np}}(cl) \tag{5}
\]

**Appendix 1. Computing unit: instances cost**

A common pricing scheme used by recent cloud providers is primarily based on virtual machine (VM) hours. Formula .6 presents the cost of leasing \( nb\text{Instances} \) VM-instances for a certain time \( \text{timeUnit} \).

\[
Cost_{\text{inst}}(cl) = nb\text{Instances} 	imes \text{price} 	imes \text{runtime} \times \text{timeUnit} \tag{6}
\]

Here the price is the dollar cost per \( \text{timeUnit} \) (e.g., In Amazon EC2 small instance the price is 0.464$ per hour).

**Appendix 2. Storage cost**

As mentioned earlier the storage cost includes the cost of leased storage volume (GB per month) and the cost of I/O requests to/from this attached storage volume. In Amazon EC2 for instance, this would be the cost of attaching Amazon EBS to VM-instances to increase the storage capacity using a highly durable and reliable way. The total storage cost is accordingly given by Formula .7:

\[
Cost_{\text{st}}(cl) = \text{costPhysicalHosting} + \text{costIORequests} \tag{7}
\]

**Appendix 3. Network cost**

The network cost varies in accordance to the service type of the source and destination (e.g., computational service and storage services) and whether the data transfer is within or across sites. In general, inter–datacenter communications are more expensive than intra–datacenter communications. Formula .8 shows the total cost of network communications as the sum of inter– and intra–datacenter communications (traffic\text{InterDC} and traffic\text{IntraDC}).

\[
Cost_{\text{np}}(cl) = \left( \text{price}\text{InterDC} \times \text{sizeUnit} \times \text{traffic}\text{InterDC} \right) \times \frac{\text{price}\text{IntraDC} \times \text{sizeUnit} \times \text{traffic}\text{IntraDC}}{\text{price}\text{IntraDC} \times \text{sizeUnit} \times \text{traffic}\text{IntraDC}} \tag{8}
\]

where price\text{InterDC} and price\text{IntraDC} are the dollar cost per sizeUnit.(interested readers could refer to [50] for more details)

**References**

[1] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, S. Ullah Khan, The rise of “big data” on cloud computing: Review and open research issues, Information Systems 47 (2015) 98–115. doi: 10.1016/j.is.2014.07.006. URL http://dx.doi.org/10.1016/j.is.2014.07.006

[2] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, N. B. Anuar, The rise of traffic classification in iot networks: A survey, Journal of Network and Computer Applications 88 (2017) 10–28.

[3] F. A. Alaba, M. Othman, I. A. T. Hashem, F. Aloataibi, Internet of things security: A survey, Journal of Network and Computer Applications 47 (2015) 98–115.

[4] C. Yang, Q. Huang, Z. Li, K. Liu, F. Hu, Big Data and cloud computing: innovation opportunities and challenges, International Journal of Digital Earth 10 (1) (2017) 13–53. doi:10.1080/17538947.2016.1239771.

[5] J. A. González-Martín, I. Linser, M. L. Bote-Lorenzo, E. Gómez-Sánchez, R. Cane-Parra, Cloud computing and education: A state-of-the-art survey, Computers & Education 80 (2015) 132–151.
[49] J. Brzezinski, C. Sobaniec, D. Wawrzyniak, From session causality to causal consistency., in: PDP, 2004, pp. 152–158.

[50] H.-E. Chihoub, Managing Consistency for Big Data Applications on Clouds: Tradeoffs and Self Adaptiveness. Distributed, Parallel, and Cluster Computing, Ph.D. thesis, PhD thesis, Université européenne de Bretagne (2013).

[51] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, R. Yerneni, PNUTS: Yahoo!’s hosted data serving platform, Proceedings of the VLDB Endowment 1 (2) (2008) 1277–1288.

[52] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with ycsb, in: Proceedings of the 1st ACM symposium on Cloud computing, ACM, 2010, pp. 143–154.

[53] M. Diogo, B. Cabral, J. Bernardino, Consistency Models of NoSQL Databases, Future Internet 11 (2) (2019) 43. doi:10.3390/fi11020043.