Convergent Causal Consistency for Social Media Posts

Ta-Yuan Hsu
University of Illinois at Chicago
Chicago, IL, USA
thsu4@uic.edu

Ajay D. Kshemkalyani
University of Illinois at Chicago
Chicago, IL, USA
ajay@uic.edu

Abstract

Geo-replicated services play a vital role in cloud storage management by providing enhanced availability, higher reliability, and lower latency on demand access to shared infrastructure and data resources. In such environments, consistency is a critical and essential consideration for distributed storage systems where it is required to make updates to the replicated data. Convergent causal consistency has become a popular consistency model by offering useful semantics for online human interaction services. Adaptive non-full replication strategies have potential benefit of lower message counts in social network systems. However, static replication is ineffective for time-varying workloads. This paper presents a causal+ consistency protocol, CaDRoP, to support adaptive dynamic replication with the convergence property for all comments following a post and the causal ordering between posts with explicit causality. We evaluate CaDRoP protocol with realistic workloads by different PUT rates in terms of the practical price of Amazon Web Service. The results show that CaDRoP can yield significantly lower cost than it is running in a statically replicated data store. We further evaluate CaDRoP by comparing it with a clairvoyant optimal replication solution. The findings indicate that with cache, CaDRoP incurs only around 6% $\sim$ 16% extra cost. Without cache, CaDRoP brings around 2% $\sim$ 4.5% extra cost in steady states.

CCS Concepts: • **Networks** → *Network simulations*; • **Computing methodologies** → **Modeling and simulation**; • **Information systems** → *Remote replication.*

Keywords: causal consistency, partial replication, social networks, performance, cost optimization

PaPoC'21, April 26, 2021, Online, United Kingdom
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8338-7/21/04…$15.00
https://doi.org/10.1145/3447865.3457967

ACM Reference Format:
Ta-Yuan Hsu and Ajay D. Kshemkalyani. 2021. Convergent Causal Consistency for Social Media Posts. In *8th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC'21), April 26, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3447865.3457967

1 Introduction

Online cloud storage is one of the most popular data management solutions and the fastest growing services in large-scale cloud computing. It has been widely deployed by businesses and enterprises to manage their data [8–10, 19, 20], including data-driven applications such as LinkedIn and Facebook. Data geo-replication is a critical part of any storage systems and a widely adopted approach to improve the availability and performance for massive scale. It is the process of maintaining copies of data at both locally and geographically dispersed stores closer to the users. Thus, the latency between end-users and the store servers can be effectively lowered, in addition to offering improvements in system scalability.

In geographically distributed environments, partial replication is an advanced measure, where data objects replicate to a subset of the system store nodes and their updates are propagated to fewer replicas. Hence, the system can avoid propagating unnecessary resources to improve storage utilization and reduce network transmission costs against full replication.

Replication gives rise to the problem of data consistency across different replicas. Linearizability is the strongest consistency model, requiring global synchronization to access data sequentially. Several well known cloud store platforms are satisfied with weaker consistency models to provide lower latency [3, 12, 13, 20]. Causal consistency (CC) has gained significant attention as an attractive consistency for geo-replicated cloud storage systems [1, 2, 11, 15, 21–24, 26–28, 30], since it supports the ordering of operations with respect to program and read-from order across store nodes. Moreover, it not only avoids the unpredictable execution status allowed by weaker consistency (e.g., eventual consistency), but provides lower latency than strong consistency models (e.g., linearizability).

CC preserves intuitive causal ascription, crucial in social networks (e.g., privacy policies). It improves user experience, because, with it, events appear to each user in the correct order. Moreover, CC+, CC with convergent conflict handling, integrates the CC model and eventual consistency [29]. CC+ not only supports the causality order for write/update operations, but also requires that all replicas converge to the same state under concurrent conflicting updates.

Contributions:. This paper presents an overview of CaDRoP (Causal Consistency under Dynamic Replication Protocol), a new cost-optimized protocol that ensures causal+ consistency (CC+) in a partially geo-replicated platform. CC+ protocol requires that data replicas converge to the same state under concurrent updates. Current existing approaches [1, 2, 11, 20–24, 26–28, 30] maintain CC+ in standard key-value storage configuration. Most of them are based on full replication, whereas some CC+ protocols [22, 27, 28, 30] support partial replication. There are some limitations when applying the current CC+ protocols to social media platforms.

- When users have access to a post (e.g., an image), all the replying comments return. Each comment corresponds to an update operation to a post. The existing CC+ protocols treat the post and its following comments as values to a variable. However, none of these CC+ approaches can achieve the convergence property for the values corresponding to the same post. Since they use the last-writer-wins reconciliation, only the value from the latest writer is kept around.
- Further, the current CC+ protocols rely on static underlying replication, i.e., the data replica placement is predetermined. However, static replication of data resources in dynamic environments with time-varying workloads is ineffective for cost management.

In contrast, CaDRoP has the following advantageous features.

1. CaDRoP is the first protocol to achieve CC+ for all replying comments (update operations) corresponding to an object (a post) with a unique key or for different objects with explicit happens-before relationships in social applications by a key-values store system. Users from different replica store can observe the same global causal ordering of all the text replies to a post.
2. CaDRoP is adapted to (proactive) dynamic data replication.
3. CaDRoP also integrates CC+ across storage layer replicas and caches to reduce network transmission costs.

We conduct an evaluation of cost-effectiveness of CaDRoP algorithm via trace-driven CloudSim simulator toolkit and realistic workload traces from Twitter in terms of the prices set on Amazon Web Service (AWS) as of 2019. Results show that the total system cost can be highly reduced by CaDRoP in a dynamic replication strategy [16] in comparison to the same protocol without caches. We further evaluate CaDRoP by comparing it with a clairvoyant optimal replication solution. The findings indicate that with cache, CaDRoP incurs only around 6% ∼ 16% extra cost. Without cache, CaDRoP brings around 2% ∼ 4.5% extra cost in steady states. Further details, including pseudocode of the CaDRoP algorithm and more experimental results, are given in [14].

This paper is organized as follows. Section 2 gives the design model of CaDRoP. Section 3 describes our proposed approach. Section 4 reports the simulation experiments along with the cost effectiveness evaluation of CaDRoP.

Section 5 summarizes our work.

2 Definitions and system model

2.1 Causal consistency (CC)

A CC system requires that clients observe the results returned from the data repository servers, consistent with the causality order. Causality is the happen-before relationship between two events. The two events must be visible to all clients in the same order, when they are causally related. In other words, when users in client A observe that event M1 happens before M2, other users in client B can perceive that the effects of M1 occurring are visible to M2. Otherwise, a (potential) causality violation has occurred. When a series of access operations occur on a single thread, they are serialized as a local history h. The set of local histories from all threads form the global history H. For potential causality , if there are two operations o_1 and o_2 in O_H, we say that o_2 causally depends on o_1, denoted as $o_1 \prec_{co} o_2$, if and only if one of the following conditions holds:

1. o_1 precedes another local operation o_2 in a single thread of execution (program order).
2. o_1 is a *write* operation and o_2 is a *read* operation that returns a value written by o_1, even if o_1 and o_2 are performed at distinct threads (read-from order).
3. there is some other operation o_3 in O_H such that $o_1 \prec_{co} o_3$ and $o_3 \prec_{co} o_2$ (transitive closure).

Especially, the causality order defines a strict partial order on the set of operations O_H. For a CC system, all the write operations that can be related by the potential causality have to be observed by each thread in the order defined by the causality order. Note that if there are multiple operations, o_1 and o'_1, updating the same variable being read by o_2 later, o_2 is dependent on both o_1 and o'_1.

2.2 System Design

CaDRoP runs in a distributed key-[values] data store that manages a set of data objects under an adaptive non-full replicated store system. Thus, CaDRoP implements a multiversion data store in social networks. [values] is a list of values corresponding to an item key. In our system, one post, such as a picture on Instagram, is viewed as an object item and is assigned a global unique number as the item key. The post object is always saved in the head of [values], denoted as v_0. Afterwards, when a comment (e.g., a list of strings) is posted out under a post, this comment text, denoted as v_i ($i >$

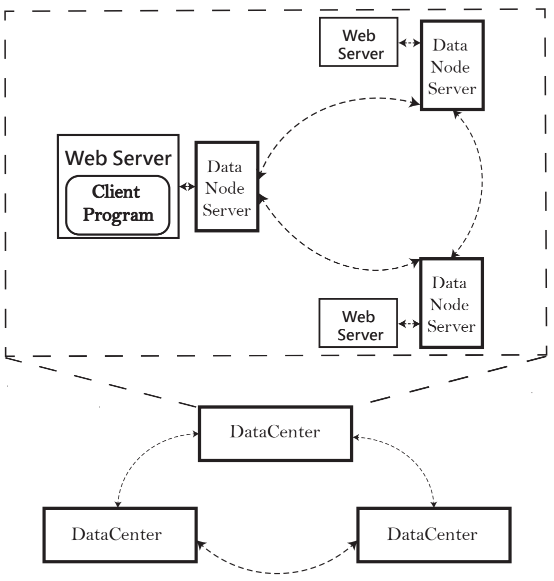

Figure 1. The system architecture.

0; i is the index of [values]), will be inserted to [values]. The value of each update operation is referred to as an immutable version of the access object. When users request access to a data object, [values] (i.e., a list of version values) is the result returned. Each entry in [values] corresponds to one update operation. In order to track causality, each version value needs to be associated with some metadata. [values] is also a causal list. For example, consider two entries v_i and v_j in [values] and $i < j$. Assume that v_i and v_j are created by update operations o_a and o_b, respectively. CaDRoP can guarantee that $o_b \not\prec_{co} o_a$.

Note that CaRRoP is aimed at social media networks. Each write operation corresponds to one text value. When there are multiple operations writing same value to the same key, all the duplicate values will be saved (i.e., no overwrite). For example, there is a landscape image (P1). Under P1, user A puts a comment - "Beautiful!" (c1) and user B also writes a comment - "Beautiful!" (c2). When user C reads P1, both c1 and c2 are presented under P1.

Although the potential causality allows to prevent any causal anomalies, it leads to higher costs to maintain many dependencies among different posts without any semantic coherency in social networks. For example, there is a cute dog photo posted in the morning and a blue sky image uploaded at noon. Tracking explicit causal order offers a more flexible solution. Under explicit causality, each application can have its own happens-before relationships between operations [4]. Because it tracks only customized relevant dependencies, explicit causality decreases the number of dependencies per modification and lowers metadata overhead. We have modeled a hybrid causality based on a column-based model. Our system maintains two types of columns: 1) key columns: they are used to store data item keys. 2) value column: each value column contains a [values] corresponding to a data item key.

CaDRoP supports the explicit causality in key columns and implements the potential causality for each value column. Explicit causality can be captured through application user interface. For example, user Bob can click @ symbol on Facebook to post an image content to reply a post done by user Alice before. Thus the client program can capture the causal dependency between the two posts, even if they are realized by different users. Otherwise, the causal relationship between different object keys will be ignored in CaDRoP.

The whole framework is a hierarchical geo-distributed cloud store system composed of multiple geographical *DCs* (see Fig. 1). All the *DCs* are fully connected by WANs with higher network access cost. They are deployed and dispersed across the world. In each *DC*, there are multiple web servers, each of which serves the data access demands from one geographical region and connects to its own data node server, which is called the host server of that connected web server. Data can be replicated asynchronously between different data servers within the same *DC* or in different *DCs*. When a data server s_r stores an object with key k, s_r is called a *replica* server of object o_k. Otherwise, s_r is a *non-replica* server. When a DC_r includes at least one replica server of object o_k, DC_r is called a *replica DC* of object o_k. Otherwise, DC_r is a *non-replica DC*. CaDRoP supports partial replication of data. Each data object is replicated in a subset of *DCs*.

CaDRoP consists of the client layer and the data store layer. They communicate with each other through the client library. The client layer implemented in web servers is responsible for storing or retrieving information to or from data node servers and presenting information to the application users. Note that the client layer has to wait for the corresponding response to the current request before sending the next access request. The underlying store layer controls the physical storage in data store servers and the data propagation between them. CaDRoP provides the following three operations to the clients: 1) POST(key, object): A POST operation assigns an object item o_k (e.g., a picture or a clip) with an item key. 2) PUT(key, value): A PUT operation assigns a text value (string) to an item key. Then, a new version value will be created. Note that if an object is visible to clients, the corresponding key always exists, unless the data object of an item key is removed from the whole system. 3) [values] $\leftarrow$ GET(key): The GET operation returns [values] corresponding to an item key in causality order.

2.3 Convergent conflict handling

CC does not establish a global order for operations in O_H. Therefore, there exist some causally independent operations, which are characterized as concurrent. Formally, two operations o_1 and o_2 in O_H are concurrent if $o_1 \not\prec_{co} o_2$ and $o_2 \not\prec_{co} o_1$. Concurrent write operations applied to the same data object very likely lead to inconsistent data states. Those are

Table 1. Definition of symbols and parameters used in the model.

Term	Meaning
D	The set of *datacenters* (*DCs*)
dm_c	Dependency meta-data *depm* set at client c
o_k	An object with a unique key k
$cvl\langle k\rangle$	A causal version list of data object k (o_k)
TS	the local Lamport timestamp for update operations
s_i	the data node server i
d	An item tuple $\langle\, k, v, dm \,\rangle$
$Dests$	A set of replica store servers

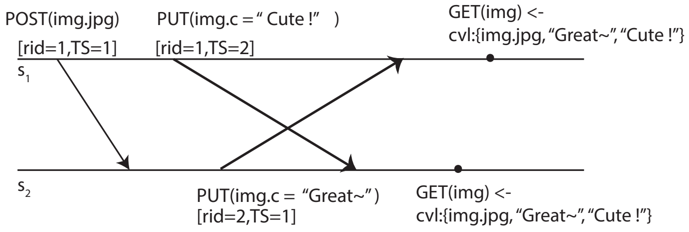

Figure 2. An example with the convergence property.

said to be in "$conflict$". Essentially, conflicts do not result in causal violation. However, when different concurrent versions of a data object are replicated to remote stores, this potentially leads to divergent undesired results to clients. Multiple concurrent versions of an object could be present in the system at the same time. In this work, CaDRoP uses the timestamp and the local data node identification to order the list of version values. This can achieve a global consistent state for different data replica nodes. Thus, CaDRoP can provide causal consistency with the convergence property.

3 Design

CaDRoP is adapted from Opt-Track protocol [15, 25], which adapts the KS algorithm [7, 17, 18] to a partially replicated causal distributed shared memory system. In Opt-Track, each data store site holds a collection of the most recent causal updates, which happened before under the $\prec_{co}$ relation. Each record in the collection includes a list of destinations, each of which consists of one replica site of the corresponding update. Whenever an update operation is initiated, a delivered multicast update message will piggyback the recently stored collection records. Once an update message is received and applied, the piggybacked collection of records is associated with the corresponding variable. If a later read operation has access to the update variable, the corresponding associated collection of records will be merged into the local collection of records. Opt-Track aims at reducing the dependency metadata size and storage cost for causal ordering. Though Opt-Track achieves CC with non-full replication across geo-distributed servers, it does not support *DC*-level partial replication and storage cache. CaDRoP is designed to achieve CC+ within and across *DCs*.

3.1 The client layer

The client library maintains for its session a dependency metadata, denoted as dm_c. dm_c consists of a set of $\langle rid, TS, Dests\rangle$ tuples, each of which indicates an update operation (POST or PUT) initiated by data node server rid at clock time TS in the causal past. $Dests$ includes replica data node servers for that update operation. Only necessary replica node information is stored. When PUT() or POST() is invoked, the client library retrieves the local dm_c and assigns POSTREQ or PUTREQ attribute to propagate a new object or a new value with dm_c to its host data node server. The host server is in charge of distributing requests to other replica node servers, handling responses from others, and returning feedback to the client. Although PUT and POST operations are very similar in the client layer, their corresponding functions in the storage layer are different. POST needs to implement CC for different objects, whereas PUT needs to enforce CC+ for the comments to an object. When GET() is invoked, the client library assigns GETREQ attribute to propagate an access request to its host data node.

3.2 The storage layer

The data storage layer is composed of multiple data node servers. Each data object can be replicated to one or more data node servers. As mentioned before, the CaDRoP data store layer exposes three main functions to the client library:

- $\langle$POSTREPLY $dmr\rangle \leftarrow \langle$POSTREQ $k, o_k, dm_c\rangle$.
- $\langle$PUTREPLY $dmr\rangle \leftarrow \langle$PUTREQ $k, v, dm_c\rangle$
- $\langle$GETREPLY $cvl\langle k\rangle\rangle \leftarrow \langle$GETREQ $k\rangle$

Note that dm denotes a dependency meatadata set and dmr indicates a returned dm. For a POSTREQ operation in a host data node server, it needs to update the local Lamport timestamp TS. When a host node server invokes a POSTREQ, it will realize two different packages, d and f. d is composed of four elements – dm_s, the set of replicas, TS, and o_k. f is a tuple of four elements – dm_s, the set of replicas, TS, and the key id k. d will be propagated to each other replica server, whereas f will be propagated to each non-replica server. In CaDRoP, we assume that the host server for the client initiating a post o_k is always a replica of object o_k. d may be inserted to the head of $cvl\langle k\rangle$. However, some entries in $cvl\langle k\rangle$ are concurrent with d. CaDRoP can sort those concurrent entries by their TS and rid, in ascending order. Thus, the text values of $cvl\langle k\rangle$ saved in different data servers can be present in the same convergent order. As shown in Figure 2, when two users retrieve the "img.jpg" from s_1 and s_2, respectively, they can obtain a consistent cvl result, {img.jpg, "*Great~*", "*Cute!*"},in causality order.

3.3 Dynamic Replication Model

Most of the existing CC protocols are based on static replication models in geo-replicated data stores. In other words, the numbers of replicas (*RF*) for a variety of data objects are predetermined. All replication decisions are made before the system is operational and replica configuration is invariant during operation. However, static replication of data resources in dynamic environments hosting time-varying workloads is obviously ineffective for optimizing system utilization, especially in social network systems. Dynamic replication strategies have been widely used as means of increasing the data availability of large-scale cloud store systems. **CORP** model, a proactive dynamic data replication strategy, has been proposed in [16] to effectively improve the total system cost in a social network system. According to the current data resource allocation and historical changes in workload patterns, **CORP** employs the autoregressive integrated moving average (ARIMA) model to predict data object access frequency in the near future. In order to optimize system cost, we incorporate **CORP** model as the underlying replication mechanism into CaDRoP protocol. Based on the requirement of **CORP**, a time slot system is required to realize the data migration process in CaDRoP.

CORP strategy runs at the end of each time slot and outputs a set of replicas for each data object. Then, the home server for that object triggers the migration process, based on the replica placement at the current time slot and that at the next time slot. It is noted that the regular CORP runs the ARIMA prediction model by an equal time interval. At runtime, the prediction is constantly updated. When new access requests arrive in the current time slot, they are getting involved into the time series and the information in the oldest time slot is removed from the time series. However, when a data object is created, there is not sufficient data in the time series initially (i.e., the training data set is not enough). Therefore, CaDRoP adopts cache mechanism, based on a PUSH model, to reduce the network transmission cost, especially in the initial time slot(s). When a non-replica s_i receives a requesting data package with key k by fetching cvl from another replica server s_r, $cvl\langle k\rangle$ may be cached in s_i with a sequence number seq assigned by s_r. For object o_k, s_i becomes a slave server of s_r. Afterwards, whenever s_r receives an update value, s_r relays the update value to s_i with a seq (increasing by one per PUT). Based on the seq, s_i can maintain a visible $cvl\langle k\rangle$ in causality order. When the migration process initiates, **CORP** outputs a new set of replicas of a key k (denoted as $k.replicas'$) for the next time slot t_h to the home server s_i. Based on different replica distributions, s_i will send the $replicas'$ or replicate $cvl\langle k\rangle$ + $replicas'$ to the other servers within the same DC. Similar to POST or PUT operations, the migration process utilizes the relay mechanism to reduce the network transmission cost across DCs. The home s_i may just send $k.replicas'$ to DC_j in the following three cases: 1) DC_j is not a replica DC in t_h. 2) DC_j was a replica DC or included a cache server in t_{h-1}, and is a replica DC in t_h (lines 13-18). 3) DC_j was not a replica DC in t_{h-1}, but will be a replica DC in t_h. After receiving $k.replicas'$ or $k.replicas$ + $cvl\langle k\rangle$ from other DCs, it needs to update the replica placement and store $cvl\langle k\rangle$ (if received), and then to relay them to other servers within the same DC.

4 Performance Evaluation

4.1 Experimental Setting

We evaluate the proposed **CaDRoP** protocol by real traces of requests to the web servers from Twitter workload and the CloudSim discrete event simulator[6]. These realistic traces contain a mixture of temporal and spatial information for each http request. The number of http requests received for each of the target data objects (e.g., photo images) is aggregated in 1000-secs intervals based on the dataset used in [16]. By implementing our approaches on the Amazon cloud provider, it allows us to evaluate the cost-effectiveness of request transaction, data store, and network transmission, and to explore the impact of workload characteristics. We also evaluate CaDRoP by a clairvoyant Optimal Placement (OPT) Solution, proposed in [16], based on the time slot system and object access patterns known in advance.

4.2 Data Object Workload

Our work focuses on the data store framework on image-based sharing in social media networks, where applications have geographically dispersed users who PUT and GET data, and fit straightforwardly into a key-[values] model. We use actual Twitter traces as a representation of the real world. PUT or POST, denoted as *Put*, to a timeline occurs when users post a tweet, retweet, or reply messages. We crawl the real Twitter traces as the evaluation input data. Since the Twitter traces do not contain information of reading the tweets (i.e., the records of *Gets*), we set five different ratios of Put/Get (P_{rate}: *Put* rate), where the patterns of *Gets* on the workloads follow Longtail distribution model [5]. The simulation workload contains several Tweet objects. The volume V of each target tweet in the workload is 2 MB. The simulation is performed for a period of 20 days. The results for each object show that they have similar tendency.

The experiment has been performed via simulation using the CloudSim toolkit [6] to evaluate the proposed system. CloudSim is a JAVA-based toolkit that contains a discrete event simulator and classes that allow users to model distributed cloud environments, from providers and their system resources (e.g., physical machines and networking) to customers and access requests. CloudSim can be easily developed by extending the classes, with customized changes to the CloudSim core. We figure out our own classes for simulation of the proposed framework and model 9 *DCs*

Table 2. Cost improvement rates in different *Put* rates and RF values.

P_{rate}	0.05	0.1	0.2	0.5	0.8
RF=9	3.46%	2.87%	5.24%	4.41%	4.16%
RF=5	72.62%	58.88%	55.05%	21.35%	6.74%
RF=2	79.46%	69.49%	56.33%	29.08%	11.95%

in CloudSim simulator. Each *DC* is composed of 4 pairs of web servers and data servers. Each data server incorporates a 50GB storage space and each web server is in charge of user's query processing from one (or a few) states in US or one country in Asia and in Europe. The price of the storage classes and network services are set in terms of Amazon Web Service (AWS) as of 2019.

4.3 Results and Discussion

The performance criteria we use are based on the monetary cost and the cost improvement rates under varying P_{rate}. *RF* is the number of replica *DC*, where it is randomly pre-decided and each replica *DC* includes one replica data node server. We vary *RF* to evaluate the cost effectiveness of our proposed approach. Furthermore, when *RF* and the replica placement for each key are predetermined, CaDRoP is simplified to 'CaS' (i.e., the underlying replication strategy is static). Cost is represented by the total system cost, which is composed of transaction cost (TC), network transmission cost (NTC), and storage cost (SC). We use the term 'transaction' to denote data query operations, such as *Put* or *Get*. NTC depends on the size of the packet (e.g., a *d* packet) transmitted. SC includes the costs of storing data items (including the *dm* data) and the bookkeeping management of data replication information.

CaS' Vs. CaS: To evaluate the cost effectiveness of the cache component, we examine the system performance with the comparisons between CaS′ (w/o cache) and CaS on cost improvement rate with respect to different RF, which is defined as:

$$\frac{cost(CaS') - cost(CaS)}{cost(CaS')} \tag{1}$$

Table 2 shows the cache effectiveness of different RF modes for different *Put* rates increases as RF decreases. As *Put* rate decreases, the cost improvement of CaS becomes higher except for full DC replication (RF=9).

CaS Vs. CaDRoP: We now evaluate the cost effectiveness of CaDRoP by comparing it with CaS. By running the same workloads as before, [*TC*] in Table 3 presents the TCs of various RF models in different *Put* rates. Lowering the number of transactions to fetch objects from remote data servers increases throughput in cloud environments, while an increased number of transactions would lead to an over-utilization of the underlying systems. Thus, the total TC is completely subject to the number of transactions. The results show that CaDRoP can achieve the best performance for TC under the same cache capacity, although it needs to bring additional transactions for the migration process. [*NTC*] in Table 3 presents the NTC of CaDRoP in comparison with various RF models in different *Put* rates. The smaller the NTC, the lower the network bandwidth consumption. Although NTC of CaDRoP is slightly higher than that of the full *DC* replication, it is much lower than others' NTCs. [*SC*] in Table 3 shows the results of SC of CaDRoP in comparison with other alternatives. It is noteworthy that the SC of CaDRoP falls in between the SCs of the replication models with RF=9 and RF=2. This implies that the proper number of replicas for CaDRoP is able to decrease TC and NTC. [*TSC*] in Table 3 presents the total system costs for CaDRoP and CaS in different RF values. It illustrates that CaDRoP can reduce TC and NTC at the slight cost of SC.

Table 3. The price cost comparisons between CaS and CaDRoP in different *Put* rates and RF models. 'SC' includes two costs: (i) storing data objects + (ii) storing *dm* data. Similarly, 'NTC' includes two costs: (i) transmitting data objects + (ii) transmitting *dm* data.

		Price Cost Comparison			
		[*SC*]	[*TC*]	[*NTC*]	[*TSC*]
P_{rate}=0.05	CaS + RF=9	0.182	35.29	1.363	36.84
	CaS + RF=5	0.106	31.62	23.43	55.16
	CaS + RF=2	0.048	18.85	37.63	56.53
	CaDRoP	0.102	13.97	2.333	16.4
P_{rate}=0.1	CaS + RF=9	0.180	33.50	1.367	35.05
	CaS + RF=5	0.103	29.29	16.61	46.00
	CaS + RF=2	0.045	18.04	26.55	44.63
	CaDRoP	0.096	13.36	2.163	15.62
P_{rate}=0.2	CaS + RF=9	0.181	28.39	1.387	29.95
	CaS + RF=5	0.102	26.84	16.60	43.54
	CaS + RF=2	0.044	17.09	22.60	39.73
	CaDRoP	0.041	10.93	2.693	13.66
P_{rate}=0.5	CaS + RF=9	0.179	27.71	1.381	29.27
	CaS + RF=5	0.102	26.19	12.31	38.59
	CaS + RF=2	0.043	16.98	17.53	34.55
	CaDRoP	0.062	10.87	2.183	13.11
P_{rate}=0.8	CaS + RF=9	0.181	27.81	1.40	29.39
	CaS + RF=5	0.1.02	25.99	11.13	37.22
	CaS + RF=2	0.043	16.93	15.57	32.54
	CaDRoP	0.057	10.33	2.191	12.58

CaDRoP VS. CaDRoP' (w/o cache) CaDRoP integrates cache functionality to improve the system costs. Thus, in this section we present experiments aimed at evaluating how the total costs are improved by CaDRoP against CaDRoP′. Table 4 presents the results of the cost saving ratio (Δ_{saving}) for different *Put* rates. Δ_{saving} is defined as

$$\frac{cost(CaDRoP') - cost(CaDRoP)}{cost(CaDRoP')} \tag{2}$$

Table 4. Δ_{saving}: The cost improvement results for different *Put* rates show that caching has taken an important step to improve the total system costs. Δ_{inc}: The performance evaluation of CaDRoP compared to CaDRoP+OPT. $\Delta_{inc'}$: The performance evaluation of CaDRoP' compared to CaDRoP'+OPT' in steady states.

P_{rate}	0.05	0.1	0.2	0.5	0.8
Δ_{saving}	91.95%	85.01%	75.14%	57.84%	49.02%
Δ_{inc}	16.08%	13.17%	10.21%	9.02%	6.17%
$\Delta_{inc'}$	1.72%	2.62%	1.6%	4.51%	3.31%

Since the evaluation data come from the social network, each individual data object brings a lot of requests in the initial time slots. It can be observed that the results indicate that the lower the P_{rate} (Get-intensive), the better the Δ_{saving} is.

CaDRoP evaluation: In order to evaluate the effectiveness of CaDRoP, we also implemented the Optimal Placement Solution (OPT) proposed in [16] as the clairvoyant replication strategy. CORP runs on the underlying replication layer of CaDRoP. Compared to CORP, OPT knows the exact temporal and spatial data object access patterns. OPT can figure out the optimal object placement for each time slot. CaDRoP+OPT means that the underlying layer of CaDRoP implements OPT rather than CORP. OPT uses the real object access (*Put* and *Get*) numbers as the inputs in different time slots. Δ_{inc} is defined as

$$\frac{cost(CaDRoP) - cost(CaDRoP + OPT)}{cost(CaDRoP)} \tag{3}$$

Δ_{inc} in Table 4 illustrates the comparisons between CaDRoP and CaDRoP+OPT. CaDRoP only increases 6% ∼ 16% of total system cost compared to CaDRoP+OPT.

In order to measure the cost effectiveness of CaDRoP in steady states (including enough training time slots), we also compare the cost of CaDRoP′ (w/o cache) to that of CaDRoP′+OPT′ (w/o cache) in steady states. $\Delta_{inc'}$ in Table 4 gives the cost increase ratios ($\Delta_{inc'}$) of CORP compared to OPT for different *Put* rates. We notice that $\Delta_{inc'}$ rates are around 2% ∼ 4.5%. $\Delta_{inc'}$ is defined as

$$\frac{cost(CaDRoP') - cost(CaDRoP' + OPT')}{cost(CaDRoP')} \tag{4}$$

5 Conclusion

We proposed CaDRoP to ensure CC+ between posts and for the comments under each post in social network systems. CaDRoP is adapted to a proposed dynamic replication algorithm CORP, which proactively deploys required data replicas in geo-replicated datastores. We presented an evaluation of the effect of the CaDRoP in terms of cost improvement via trace-driven CloudSim toolkit and realistic workload traces from Twitter. Simulations show that, with caching, as the RF increases, the TSC decreases. CaDRoP is around 55 ∼ 70% lower than CaS in different predetermined RF models, as shown in Table 3. In order to further evaluate CaDRoP, we compared it to an OPT replication solution based on known temporal and spatial access patterns. CaDRoP increases only 6 ∼ 16% of TSC of CaDRoP+OPT. Without cache, the TSC of CaDRoP′ is slighly higher than that of CaDRoP′+OPT′ in a steady state. The simulation results also showed that the TSC of CaDRoP is usually improved better in lower P_{rate} (i.e., CaDRoP is cost-effective for most social applications with Get-intensive workloads).

References

[1] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro. 2016. Cure: Strong Semantics Meets High Availability and Low Latency. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 405–414.

[2] Sérgio Almeida, João Leitão, and Luís Rodrigues. 2013. ChainReaction: A Causal+ Consistent Datastore Based on Chain Replication. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. ACM, New York, NY, USA, 85–98.

[3] H. Attiya, F. Ellen, and A. Morrison. 2017. Limitations of Highly-Available Eventually-Consistent Data Stores. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2017), 141–155.

[4] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2013. Bolt-on Causal Consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. ACM, New York, NY, USA, 761–772.

[5] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. 2010. Finding a Needle in Haystack: Facebook's Photo Storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Vancouver, BC, Canada) *(OSDI'10)*. 47–60.

[6] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Softw. Pract. Exper.* 41, 1 (Jan. 2011), 23–50. https://doi.org/10.1002/spe.995

[7] P. Chandra, P. Gambhire, and A. D. Kshemkalyani. 2004. Performance of the Optimal Causal Multicast Algorithm: A Statistical Analysis. *IEEE Transactions on Parallel and Distributed Systems* 15, 1 (2004), 40–52.

[8] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. 2008. PNUTS: Yahoo!'s Hosted Data Serving Platform. *Proc. VLDB Endow.* 1, 2 (Aug. 2008), 1277–1288.

[9] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.* 31, 3, Article 8 (Aug. 2013), 22 pages.

[10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. *SIGOPS Oper. Syst. Rev.* 41, 6 (Oct. 2007), 205–220.

[11] Diego Didona, Rachid Guerraoui, Jingjing Wang, and Willy Zwaenepoel. 2018. Causal Consistency and Latency Optimality: Friend or Foe? *Proc. VLDB Endow.* 11, 11 (July 2018), 1618–1632.

[12] Jiaqing Du, Sameh Elnikety, Amitabha Roy, and Willy Zwaenepoel. 2013. Orbe: Scalable Causal Consistency Using Dependency Matrices and Physical Clocks. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (Santa Clara, California) *(SOCC '13)*. ACM, New York, NY, USA, Article 11, 14 pages. https://doi.org/10.1145/2523616.2523628

[13] Jiaqing Du, Calin Iorgulescu, Amitabha Roy, and Willy Zwaenepoel. 2014. GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 03 - 05, 2014*. 4:1–4:13. https://doi.org/10.1145/2670979.2670983

[14] T.Y Hsu and A. D. Kshemkalyani. 2021. CaDRoP: Cost Optimized Convergent Causal Consistency in Social Network System. In *2021 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–10.

[15] Ta-Yuan Hsu, Ajay D. Kshemkalyani, and Min Shen. 2018. Causal consistency algorithms for partially replicated and fully replicated systems. *Future Generation Computer Systems* 86 (2018), 1118 – 1133.

[16] Ta-Yuan Hsu and Ajay D. Kshemkalyani. 2019. A Proactive, Cost-Aware, Optimized Data Replication Strategy in Geo-Distributed Cloud Datastores. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing* (Auckland, New Zealand) *(UCC'19)*. Association for Computing Machinery, New York, NY, USA, 143–153.

[17] A. Kshemkalyani and M. Singhal. 1998. Necessary and Sufficient Conditions on Information for Causal Message Ordering and Their Optimal Implementation. *Distributed Computing* 11, 2 (April 1998), 91–111.

[18] Ajay D. Kshemkalyani and Mukesh Singhal. 1996. An Optimal Algorithm for Generalized Causal Message Ordering. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing* (Philadelphia, Pennsylvania, USA) *(PODC '96)*. ACM, New York, NY, USA, 87–.

[19] Avinash Lakshman and Prashant Malik. 2010. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010), 35–40.

[20] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2011. Don'T Settle for Eventual: Scalable Causal Consistency for Wide-area Storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. ACM, New York, NY, USA, 401–416. https://doi.org/10.1145/2043556.2043593

[21] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2013. Stronger Semantics for Low-latency Geo-replicated Storage. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (Lombard, IL) *(nsdi'13)*. USENIX Association, Berkeley, CA, USA, 313–328.

[22] Tariq Mahmood, Shankaranarayanan PN, Sanjay Rao, T. Vijaykumar, and Mithuna Thottethodi. 2018. Karma: Cost-effective Geo-replicated Cloud Storage with Dynamic Enforcement of Causal Consistency. *IEEE Transactions on Cloud Computing* (06 2018), 1–1.

[23] Syed Akbar Mehdi, Cody Littley, Natacha Crooks, Lorenzo Alvisi, Nathan Bronson, and Wyatt Lloyd. 2017. I Can't Believe It's Not Causal! Scalable Causal Consistency with No Slowdown Cascades. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 453–468.

[24] M. Roohitavaf, M. Demirbas, and S. Kulkarni. 2017. CausalSpartan: Causal Consistency for Distributed Data Stores Using Hybrid Logical Clocks. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 184–193.

[25] Min Shen, Ajay D. Kshemkalyani, and Ta Yuan Hsu. 2015. Causal Consistency for Geo-Replicated Cloud Storage under Partial Replication.. In *IPDPS Workshops*. IEEE, 509–518.

[26] K. Spirovska, D. Didona, and W. Zwaenepoel. 2017. Optimistic Causal Consistency for Geo-Replicated Key-Value Stores. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2626–2629.

[27] K. Spirovska, D. Didona, and W. Zwaenepoel. 2019. PaRiS: Causally Consistent Transactions with Non-blocking Reads and Partial Replication. In *IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 304–316.

[28] Yu Tang, Hailong Sun, Xu Wang, and Xudong Liu. 2017. Achieving convergent causal consistency and high availability for cloud storage. *Future Generation Computer Systems* 74 (2017), 20 – 31.

[29] Werner Vogels. 2009. Eventually Consistent. *Commun. ACM* 52, 1 (Jan. 2009), 40–44. https://doi.org/10.1145/1435417.1435432

[30] Marek Zawirski, Nuno Preguiça, Sérgio Duarte, Annette Bieniusa, Valter Balegas, and Marc Shapiro. 2015. Write Fast, Read in the Past: Causal Consistency for Client-Side Applications. In *Proceedings of the 16th Annual Middleware Conference* (BC, Canada) *(Middleware '15)*. ACM, New York, NY, USA, 75–87.