DTC: A Dynamic Transaction Chopping Technique for Geo-Replicated Storage Services

Ning Huang, Lihui Wu, Weigang Wu, *Member, IEEE*, and Sajal K. Das, *Fellow, IEEE*

Abstract—Replicating data across geo-distributed datacenters is usually necessary for large scale cloud services to achieve high locality, durability and availability. One of the major challenges in such geo-replicated data services lies in consistency maintenance, which usually suffers from long latency due to costly coordination across datacenters. Among others, transaction chopping is an effective and efficient approach to address this challenge. However, existing chopping is conducted statically during programming, which is stubborn and complex for developers. In this article, we propose Dynamic Transaction Chopping (DTC), a novel technique that does transaction chopping and determines piecewise execution in a dynamic and automatic way. DTC mainly consists of two parts: a dynamic chopper to dynamically divide transactions into pieces according to the data partition scheme, and a conflict detection algorithm to check the safety of the dynamic chopping. Compared with existing techniques, DTC has several advantages: transparency to programmers, flexibility in conflict analysis, high degree of piecewise execution, and adaptability to data partition schemes. A prototype of DTC is implemented to verify the correctness of DTC and evaluate its performance. The experiment results show that our DTC technique can achieve much better performance than similar work.

Index Terms—Cloud service, cloud storage, data replication, transaction processing, datacenter

1 Introduction

Large-scale Internet applications, such as eBay and Gmail, usually rely on storage services provisioned by geo-replicated storage systems to support online transaction processing (OLTP) services. Such geo-replicated storage services are usually deployed across a number of datacenters located in different places, and each datacenter consists of a large number of database servers in charge of handling data access requests [1]. To achieve high availability, high locality and high reliability, data tables in a database may be horizontally partitioned into multiple shards [6], according to key ranges, and each shard may be replicated at multiple datacenters.

Since the coordination among multiple datacenters is costly, OLTP in geo-replicated systems usually suffers from long latency. On the other hand, many Internet applications are sensitive to latency due to the significant effect on user experience. It has been shown that, a response time of more than 200ms may significantly drive users away [23]. Therefore, how to reduce latency of geo-replicated systems is a critical issue.

One popular way to achieve low latency is to sacrifice consistency levels [3]. For example, Cassandra [12] and Dynamo [9] provide only eventual consistency, while Eiger [15] assumes causal consistency.

However, weakening consistency level may not be acceptable for consistency sensitive applications, like Amazon and eBay, which essentially require strong consistency. Since transaction management, especially concurrency control, is quite complex and costly, strong consistency in geo-replicated systems is really a challenging task.

Recently, quite a number of solutions have been proposed to reduce the latency caused by transaction management in geo-replicated storage systems. Among them, transaction chain in Lynx is a very effective technique [29], which adopts the approach of transaction chopping [24] to divide transaction operations into smaller pieces according to data partition scheme. Such piecewise executions can reduce latency significantly, though the total execution time of a transaction is not reduced.

Although transaction chopping is an effective approach for geo-replicated storage services, existing solutions, like transaction chain of Lynx, have several constraints. Firstly, transactions are chopped by additional annotations during programming, which increases the workload of programmers and prevents its deployment for legacy application. Secondly, Lynx adopts static and prior conflict analysis, which does not reflect the runtime situation and reduce the possibility of piecewise execution. Lastly, Lynx assumes static data partition scheme, which is not adaptive to storage changes.

To address these constraints and realize more general and dynamic transaction chopping, we propose a novel transaction chopping technique for geo-replicated storage systems, called Dynamic Transaction Chopping (DTC). DTC consists of two major parts: a dynamic chopper and a conflict analyzer. The chopper is used to divide a transaction into smaller pieces according to a dynamic data partition service. The analyzer is used to check conflict relationship among transaction pieces,

- *Ning Huang, Lihui Wu, and Weigang Wu are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, Guangdong 510275, China, and also with the Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, Guangdong Province Key Laboratory of Big Data Analysis and Processing, Guangzhou, Guangdong 510006, China. E-mail: 446818431@qq.com, xyt9103@163.com, wuweig@mail.sysu.edu.cn.*
- *Sajal K. Das is with Computer Science Department, Missouri University of Science and Technology, Rolla, MO 65409 USA. E-mail: sdas@mst.edu.*

Manuscript received 27 November 2019; revised 24 February 2021; accepted 3 June 2021. Date of publication 16 June 2021; date of current version 9 December 2022. (Corresponding author: Weigang Wu.)
Digital Object Identifier no. 10.1109/TSC.2021.3089819

1939-1374 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

and try to solve these conflicts by merging some of these pieces. Choppable transactions will be executed in piecewise, and thus data access latency can be reduced.

The key issue of DTC lies in automatically checking of chopping safety, i.e, whether consistency can be guaranteed even if a transaction is returned after the first piece. In Lynx, this checking is done by the programmer via annotations in application source code. DTC, however, conducts such safety checking dynamically in a runtime way. Since the view of running transactions changes from time to time, it is not trivial to dynamically check the conflicts among transaction pieces and determine chopping safety.

To conduct dynamic safety checking, we propose a new conflict graph that represents the relationship among transaction pieces, and design a novel algorithm to search possible conflict cycle in the graphs. The search algorithm can also conduct transaction piece merging so as to reduce conflicts. Besides, we also design associated mechanisms and modules to realize dynamic chopping, dynamic conflict detection and piecewise execution. To verify the feasibility and evaluate the performance of DTC, we implement a prototype system and carry out experimental study.

Compared with static transaction chopping based geo-replication systems like Lynx, DTC can achieve much higher piecewise execution ratio and consequently reduce latency significantly. Moreover, DTC releases programmers from manual chopping and conflict detection, and it allows dynamic data partition schemes, which makes transaction chopping much easier to be applied.

The rest of the paper is organized as follows. Existing works on geo-replicated storage systems are reviewed in Section 2. We introduce Lynx, the transaction chain system based on static transaction chopping, and discuss its problems in Section 3. Section 4 presents the design of our proposed DTC technique, including system architecture, detailed operations, correctness proof. The dynamic conflict detection algorithm and piece merging mechanism are described in Section 5, and the dynamic partition service is described in Section 6. We present an implementation of DTC together with performance evaluation in Section 7. Finally, Section 8 concludes the paper and discusses possible extensions.

2 Related Work

Most of existing works on geo-replicated storage systems/services attempt to improve system performance, e.g., low latency, by sacrificing consistency levels, since a weaker consistency level needs simpler concurrency controls and reduces transaction management latency. Eventual consistency is adopted in Cassandra [12] and Dynamo [9], which in fact does not guarantee the time that an update should be conducted at all replication nodes [3]. Per-record timeline consistency, as provided by PNUTS [7], is slightly stronger than eventual consistency. Causal consistency is between eventual consistency and strong consistency and widely used in geo-replicated environments [4], [15]. Post-execution based approaches have been also studied. For example, snapshot isolation [5] detects conflicts by exchanging write-sets before committing [14], [25].

On the other hand, strong consistency is definitely necessary for many large scale distributed applications. However, traditional transaction management protocols for strong consistency, like Two-phase Locking (2PL), are no longer suitable for geo-replicated environments. Existing works on strong consistency for geo-replicated systems adopt two different strategies: optimistic and conservative.

Optimistic solutions, like H-store [10], Percolator [21], ROCOCO [18] and Janus [19], attempt to monitor conflicts and verify the consistency among datacenters while executing transactions, rather than control the execution order. TAPIR [28] allows inconsistent operations that do not affect correctness among replicated systems. It requires that, operations at a majority of the replicas are successful, although replicas may execute them in different orders. The approach in TAPIR has been used in many transactional systems [11], [17], [18], [19], [20], [22].

Homeostasis [22] allows datacenters to operate independently without communication and the inconsistency between datacenters is governed in addition. Helios [20] actively lets datacenters exchange transaction logs so as to decide whether a transaction between datacenters can commit or not.

Conservative solutions employ specific mechanism to control the order of operations so as to guarantee consistency among datacenters. MDCC [11] orders transaction requests via the Paxos agreement protocol before executing them. The uniform order can guarantee consistency across datacenters. Spanner [8], Google's globally-distributed database, enables 2PL across datacenters by a global clock facility to enable consistency. Replicated commit [16] optimizes the cross-site communication in Spanner. Callas [27] proposes to divide transactions into groups and ensure serializability inside each group respectively. The isolation property across groups is guaranteed by a special mechanism called nexus lock.

Calvin [26] is a conservative solution. It conducts pre-execution analysis on read/write set and improves system performance by scheduling transactions with unrelated read/write sets on partitioned database system.

Lynx [29] is also a conservative solution. It adopts the technique of transaction chopping [24]. Lynx divides a transaction into small pieces according to the data shards to access. More precisely, each datacenter is assigned the transaction piece that will access the data stored inside the datacenter. All the pieces of one transaction compose a transaction chain. If no conflicts, a transaction chain can return to the client after committing its first piece, i.e, the transaction is executed in piecewise way. Then, in the view of clients, latency can be significantly reduced.

Conflict detection among transaction chains is at the core of Lynx, and it is conducted by programmers by annotating source code.

Similar to Lynx, our work is also based on transaction chopping. However, Lynx relies on application programmers to conduct conflict analysis and make annotations in source code. We propose to conduct conflict checking in a dynamic and automatic way. We also propose a piece merging mechanism to solve the conflict among transaction chain to further improve system performance.

3 Transaction Chain and Its Problems

3.1 Geo-replicated Storage System

Large Internet applications and cloud services, such as Web e-mail and social networking, usually rely on geo-distributed

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

Items (primary key=item-id)

item-id	seller	high bidder	high price
345	666	123	$200
575	123	--	--

Bids (primary key=bid-id)

bid-id	bidder	item	bid-price
1	549	345	$100
2	123	345	$200

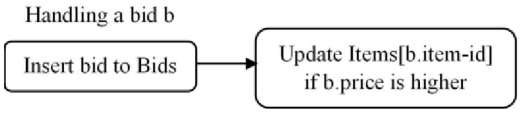


Fig. 1. Simple example of transaction chain.

datacenters to achieve high availability, locality and system throughput, and each datacenter usually consists of different types of servers, e.g, application servers and database servers.

In a distributed database system like BigTable [6], data tables are usually horizontally partitioned into shards according to key ranges, and each shard can be geo-replicated among datacenters. We assume a primary based replication management approach. One replica server is assigned to be the primary node and the others are secondary. The data access requests are sent to the primary replica, which will forward the requests to secondary replicas.

3.2 Transaction Chain in Lynx

Transaction chopping [24] is originally proposed to improve system throughput for traditional database systems. It divides a transaction into small pieces according to the data items accessed so as to realize concurrency control in small granularity. Transaction chain in Lynx [29] applies transaction chopping into geo-replicated systems. With Lynx, a programmer chops transactions into pieces by adding annotations in application, according to data partition scheme. More precisely, operations accessing shards at the same datacenter will be grouped into one piece, and all the piece of one transaction constitute a transaction chain. Each piece is executed as one hop of its transaction chain. With transaction chain, a transaction can be executed hop by hop. Fig. 1 shows example data tables and a transaction chain of an auction system, as used in [29].

Only the first piece is included in the concurrency control protocols like 2PL. A chopped transaction can return to the corresponding client immediately after the first hop commits rather than the commitment of the whole transaction. Therefore, the response time is significantly reduced.

Obviously, not all the transactions can be chopped because two transactions may have overlapped data item sets, i.e, there are conflicts between transaction chains.

In Lynx, programmers need to chop transactions into pieces and check whether a transaction can be executed in piecewise in the coding stage. The checking is done based on the SC-graph [24] and the chopping theory below:

Definition 1 (S-edge). *the edge that connects sibling pieces of the same transaction. Obviously, such an edge represents the predecessor-successor relationship between neighboring hops in a transaction chain.*

Definition 2 (C-edge). *the edge that connects two pieces from different transactions and it reflects the conflicting relationship between the two pieces. That is, the two pieces need to operate at the same data shard and at least one operation is writing.*

Definition 3 (SC-graph). *an undirected graph that represents the relationship among all transaction pieces in the system, via S-edges and C-edges.*

Definition 4 (SC-cycle). *a cycle in SC-graph, which contains both S-edges and C-edges.*

Theorem 1 (Chopping theory). *If there is no SC-cycle in the SC-graph after transactions are chopped, we say that the chopping is safe (or correct). Correspondingly, all the chopped transactions in the SC-graph are called choppable.*

A chopping is safe if a transaction return (reply) to the client (e.g, the corresponding application server that issues the request) after the first hop of the transaction is committed, i.e, this transaction can be executed hop by hop. Accordingly, we say that this transaction is choppable. Otherwise, we say that the transaction is unchoppable. An unchoppable transaction needs to be executed in the traditional way, i.e, all the operations in the transaction need to be included in the concurrency control protocol, e.g, 2PL.

3.3 Problems of Transaction Chain

Lynx relies on programmers to do conflict analysis and determine whether a transaction can be executed in a piecewise way. Such a design has several shortcomings.

Firstly, it brings additional workload to programmers. The application programmer need to master the SC-graph analysis and also know well about the database partition scheme, which are not trivial and easy for most programmers. This may prevent developers from adopting the transaction chain technique.

Secondly, it is not flexible and too conservative. Lynx assume conflict analysis is the programming stage. To guarantee safety, the SC-graph must include all possible transactions that may be executed concurrently. However, in the execution stage, instances of two conflicting transactions may appear in totally different time slots, so their conflict relationship will not affect consistency and can be executed in piecewise way. If conflict analysis is conducted during execution, only transactions with overlapping duration need to be included in conflict analysis and the percentage of piecewise execution will be increased.

Lastly, Lynx assumes a static and predefined data partition scheme. Transaction hops are determined by data shards to access, while the shards are determined by data table partition scheme. In Lynx, the chopping of transactions is determined by data partition scheme and conducted by programmers. Therefore, the partition scheme should be known by application developers. Such requirements on predefined data partition scheme will obviously constraint the deployment of transaction chain, especially for large Internet applications, which usually face fast changing users and environments.

The problems of Lynx motivated us to design a dynamic transaction chopping and analysis technique.

4 Dynamic Transaction Chopping

DTC realizes dynamic and transparent conflict detection for dynamic transaction chopping and execution. It consists of a

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

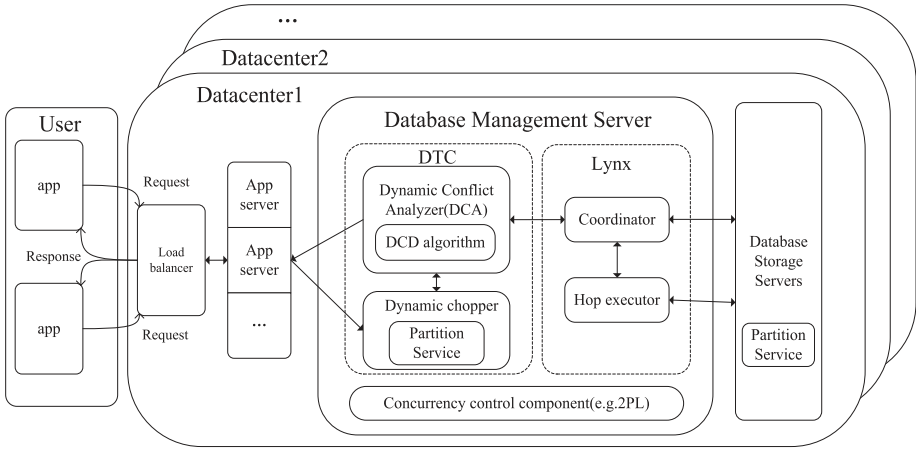


Fig. 2. Architecture of a Geo-replicated system with DTC.

dynamic chopper, dynamic conflict analyzer and associated mechanisms for dynamic piecewise transaction execution.

In the following, we describe the details of DTC, correctness proof, as well as advantages and limitations. Since the dynamic conflict detection algorithm in the analyzer is quite complex, it is described in the next section.

4.1 System Architecture and Overview

Same as in Lynx, DTC is also designed for a geo-replicated storage system, which is deployed across several datacenter in different locations. A partition scheme is used to divides data tables into shards and replicate them at different datacenters. Fig. 2 shows an illustration of the overall architecture of such a system. Each datacenter contains various server nodes, including application servers, data storage nodes and database management servers. Transaction concurrency control protocols and mechanisms are deployed at database management servers, while data tables are physically stored in data storage nodes.

Application servers execute application software developed by application programmers. The operations in an application may include different transactions of data access and processing. During execution, the requests of data processing transactions are submitted to database management servers to get access of data tables.

The modules of DTC are also deployed at data management servers as shown in Fig. 2. For concurrency control management, there is no special requirements, and traditional protocols, like 2PL, can be used.

DTC mainly contains two modules. The dynamic chopper is in charge of dividing transactions into smaller pieces, according to the data partitioning scheme provided by the partition service. The other major part is the Dynamic Conflict Analyzer (DCA for short), which is in charge of detecting conflicts among transaction chains and check the safety of piecewise execution. DTC works in an online manner during application execution and allows dynamic data portioning scheme.

The dynamic chopper chops the transaction into pieces based on the partition scheme maintained by an independent module, partition service. The chopping of transaction is driven by the partition scheme of data so the partition service is a supplemental part in our DTC technique. The partition scheme is a customized data structure to store the mapping relationship of data (shards) and the corresponding storing positions (servers). Similar to [29], DTC requires that the shards accessed at each hop are known before the chain starts executing. This will limit the application of transaction chain, i.e, DTC is not a universal technique for all application scenarios of data process transactions. The restrictions are discussed in Section 6.3.

DTC adopts an altogether different partition scheme, compared with that in transaction chain of Lynx. Each hop in Lynx consists of all data accessing operations in a datacenter and there may be several sequential hops from the same transaction in a single datacenter. Since our new system model introduces a smaller granularity of transaction hops, the analysis becomes more flexible, and the whole system can achieve higher concurrency among operations.

The procedure of piecewise execution is similar to that in Lynx. There are a coordinator and hop executors for each transaction. The coordinator invokes and monitors the execution status of transaction hops. For each hop, there is an executor to really execute the operations of a transaction piece.

4.2 DTC and Dynamic Piecewise Transaction Execution

Here, we describe the procedure of piecewise transaction execution by DTC. Fig. 3 gives an illustration of such executions.

The data partition scheme, which defines how to partition data tables into shards, is provided by a particular partition service module. Due to the increase of data volume or change of storage nodes, the data partition scheme may change accordingly, although such changes should not occur frequently. The partition service module is in charge of maintaining data partition scheme, and it is deployed as an independent module, like the configuration service in [2].

By locality, the transaction request issued by a user should be directed to the local datacenter, and then this datacenter is assigned as the first hop of the corresponding transaction and also the coordinator role of the transaction chain.

Inside a datacenter, transaction requests, directly from local application servers or directed from remote application servers, will be delivered to the dynamic chopper, which will firstly chop the transactions into small pieces

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

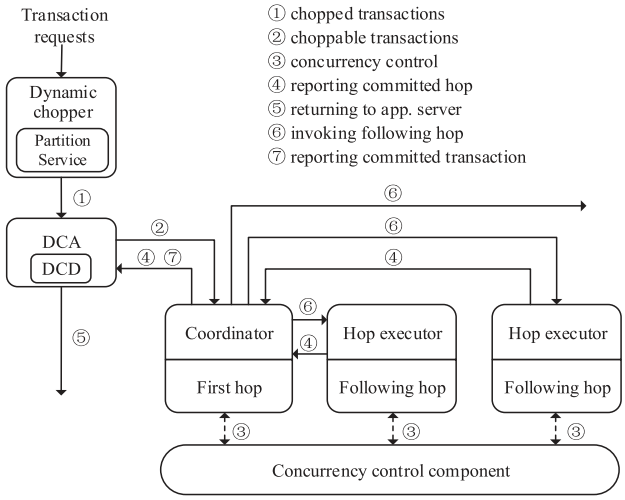


Fig. 3. Piecewise transaction execution with DTC.

simply according to the partition scheme. Each piece consists of the operations that need to access the data shards stored at one datacenter.

Then, these transaction pieces are sent to DCA, and DCA will construct SC-graph and check the conflicts among transaction pieces, so as to determine whether the chopping of the transaction is safe. Only if the chopping is safe, the transaction can be executed in piecewise. Safety checking is in fact done by the dynamic conflict detection (DCD) algorithm. DCD is at the core of DCA and it dynamically checks conflicts by searching the SC-graph. For ease of understanding, the details of DCD are described in the next section.

After the conflict detection at DCA, choppable transactions will be executed in the piecewise way. As aforementioned, the first hop will be executed at the datacenter that chops the transaction. The corresponding data management server will really execute the transaction piece and also acts as the coordinator of the transaction. After the coordinator executes and commits the first hop, it can report the state information to DTC, and consequently to the client (e.g, application server) that generated the transaction request. The other hops will be executed by the datacenters storing the corresponding data shards, which are called hop executors.

The coordinator also needs to contact the corresponding data management servers that will execute the other hops to invoke their executions. The coordinator needs to monitor the execution of all hops of a transaction. If any of them fails, the coordinator needs to re-invoke it until the execution succeeds.

When the last hop of a transaction committed, the execution of a transaction is really completed. The coordinator needs to inform the DCA module about the completion of a transaction, so that DCA can update its view of transactions being executed in the system.

During the piecewise execution, if some hop needs to modify data rather than only reads them, the corresponding server needs to spawn a sub-chain to update the derived tables of replicas so as to keep the replication consistent. Such a sub-chain is coordinated by the corresponding hop server and executed like a common transaction chain. As shown in Lynx, with the origin ordering mechanism, such sub-chains will be correctly completed and returned to the hop server. This part of the procedure is almost the same as that of Lynx and omitted here.

If the chopping done by the dynamic chopper is unsafe, the transaction needs to be executed in the traditional way. That is, all the operations need to be included in concurrency control protocol. The client can get response only after the commitment of the whole transaction.

To improve the percentage of piecewise execution, we propose to partially merge pieces chopped by dynamic chopper rather than execute the transaction as a whole piece. In such way, one hop of a transaction may contain operations executed at more than one datacenter, but the whole transaction is still divided into multiple hops. The details of such a mechanism is presented in 5.2.3.

4.3 Correctness of DTC

The correctness of DTC is examined against the classical properties required by databases, i.e, ACID (Atomicity, Consistency, Isolation and Durability). Here, we show that, ACID is guaranteed with the piecewise execution provided by DTC. Since our work focuses on transaction management crossing datacenters, we do not consider physical machines and/or system software failures.

Roughly each hop of a transaction is executed and committed as an integrated part. If there are sub-chains inside a hop, the ACID properties of the hop is guaranteed trivially by the hop coordinator mechanism.

Then, in the following, we consider only that, whether ACID properties are guaranteed crossing hops of a transaction.

Atomicity and Durability. The properties of atomicity and durability can be provided by recovery techniques. Popular recovery techniques include checkpointing and logging. In DTC, we let each hop to log its execution state in permanent storage. If some hop does not execute successfully, the coordinator in charge of the transaction will re-invoke the hop, as described in Section 4.2. With such logging and re-invoking, once the first hop is committed, each following hop will be eventually completed and committed even if there are failures[1]. Atomicity holds. In geo-replicated environments, all data are stored in stable storage of datacenters. Once a hop of transaction is completed, the data updates and modifications should have been stored by the datacenter. That is, in the view of transaction hops, durability is naturally guaranteed.

Consistency and Isolation. In traditional transaction execution without chopping, concurrency control protocols like 2PL can guarantee the properties of consistency and isolation because they can guarantee serializability, of concurrent transaction executions.

DTC also needs such concurrency control protocols to manage the execution of the first hops of choppable transactions and all operations of unchoppable transactions. Then, let consider how chopping affects consistency and isolation.

As in Lynx, we check the safety of transaction pieces via SC-graph. Static chopping based piecewise execution can provide consistency and isolation as in Lynx. Our DTC

1. Of course, if the system crashes due to serious software or hardware problems, the second or later hop may not be automatically completed by only re-invoke mechanism. Such extreme situations must be handled by administrators so that the system can be recovered correctly.

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

extends the static chopping and static checking in Lynx to dynamic chopping and dynamic checking during application execution. The consistency is guaranteed by the correctness of the DCD algorithm described in Section 5.

Please note that, to ensure correctness, we assume that only independent transactions in DTC. In data storage systems, transaction can be classified into two categories. 1) Independent transaction is composed of several, and all these hops which know their input and involved server for execution before the whole transaction begins. 2) Dependent transaction may contain some hops with unknown input or unknown execution node. DTC assumes independent transaction only. Dependent transactions can be supported by adopting some transform technique from application level, e.g., rewriting the transaction with special formula [19]. That is, all dependent transactions can be transformed to be independent ones and DTC can be used correctly.

4.4 Advantages and Overhead

As the first dynamic chopping and executing technique, DTC has the following advantages:

a) DTC is transparent to applications. Since transaction chopping and conflict checking are conducted by DTC, applications can be developed as usual and the programmers do not need to consider how to do piecewise execution. This advantage releases programmers from chopping annotations and also makes DTC applicable to common applications.
b) DTC is more accurate. Since conflict analysis is conducted during transaction execution, only really existing transactions are considered and the conflict relationship is more accurate and flexible. The percentage of piecewise execution is increased.
c) DTC is more flexible. In Lynx, data partition scheme is predefined during programming and the change of partition scheme will result in failure of transaction piecewise execution. The dynamic chopping of DTC allows changing partition schemes, which will largely facilitate the deployment of data storage systems.

About the overhead of DTC compared with Lynx, we have the following points.

a) Both DTC and Lynx detect conflicts via transaction graph traversal, so they have the same order of space and time complexity if only one run of conflict detection is considered. Using DFS against adjacent matrix, the time complexity should be $O(n^2)$, where n is the number of vertexes, i.e, transaction pieces in the graph.
b) Since DTC conducts conflict detect in runtime, it invokes more times of detections than Lynx. However, the graph of DTC should be much smaller than that of Lynx since only running transactions are included, which will save searching cost.
c) Considering the advantages of DTC, especially allowing more piecewise executions, its overhead is negligible, or at least acceptable.

5 The Dynamic Conflict Detection Algorithm

The DCD algorithm designed to dynamically detect possible conflicts among transaction pieces that produced by the dynamic chopper, so as to determine whether a transaction is choppable. Such a detection and determination is conducted according to the chopping theory [24], [29].

In static analysis, an SC-graph representing relationship among transaction chains is firstly constructed. Then, an algorithm is executed to detect whether there is an SC cycle in the graph. If an SC cycle exists, there is conflict and the transactions must be executed in traditional way. Otherwise, piecewise execution is allowed.

In our work, we consider dynamic conflict analysis, i.e, the analysis is conducted at the running time and the view of transactions changes from time to time. The conflict graph is generated according to the view of the transactions being executed and being queued to be executed (see Section 5.2.1 for the generation of conflict graph). Then, the SC-cycle based conflict detection is not enough.

In DCD, we propose two new notions: CC-graph and Simple-C-graph, which are constructed based on the SC-graph. Based on the chopping results, DCD firstly constructs an SC-graph. Then, CC-graph is constructed based on SC-graph, and Simple-C-graph is constructed based on the CC-graph. DCD detects conflicts by searching cycles in the CC-graph and Simple-C-graph.

In the following, we firstly introduce notions and definitions related and then describe the DCD algorithm. Finally, we prove the correctness of DCD.

5.1 Notions and Definitions

DCD is based on the chopping theory proposed in [24] and transaction chain defined in [29]. We firstly present the notions defined in [24] and [29], and then define new notions.

All the following definitions of edges and graphs are presented against a given set of transactions and their corresponding chopped pieces. The transactions are chopped according to predefined criterion by the partition service in Section 6.

Based on the definitions of SC-graph proposed in [24] and [29] as presented in Section 3.2, we propose new ones to realize dynamic conflict detection.

Definition 4-a (SC-cycle-A). *an SC-cycle that contains two or more C-edges connecting the same pair of transactions. This means that, there are at least two pair of conflicting operations between the same pair of transactions.*

Definition 4-b (SC-cycle-B). *an SC-cycle that is not of the type of SC-cycle A. That is, between each pair of transactions, there is at most one C-edge.*

Definition 5 (CC-graph). *a weighted and directed graph that represents the relationship among transactions in the system, with transactions as vertexes. The edges in a CC-graph, called CC-edges, are defined as below.*

Definition 6 (CC-edge). *a directed and weighted edge connecting two transactions (not transaction pieces), whose direction and weight is determined as follows:*

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

1) *For a transaction T_i in an SC-graph, if there is one or more C-edges connecting another transaction T_j, we draw a CC-edge e_{ij} with the direction from T_i to T_j.*
2) *If T_i contains only one piece connecting one or more pieces of T_j in the SC-graph, the weight of the CC-edge e_{ij} is set to be the id of that piece of T_i (a number starts from 1); otherwise the weight of e_{ij} is set to be -1.*

Based on the definition of CC-edge and CC-graph, we can get two straightforward conclusions:

i) CC-edges are pairwise. That is, if there is an edge e_{ij}, then there must exist the edge e_{ji}, since the conflict among transactions is mutual.
ii) CC-edge pairs are not symmetric. That is, the weight of e_{ij} may not be equal to that of e_{ji}.

Definition 7 (Simple/Negative connection). *if the CC-edges connecting two vertexes in a CC-graph are both with positive weight, we call that the two vertexes are simply connected and they have a simple connection relationship. Correspondingly, if two vertexes are connected with at least one negative CC-edge, we call that the two vertexes are with a negative connection relationship.*

Definition 8 (Simple-C-graph). *an undirected graph derived from a CC-graph: each edge represents a simple connection relationship in the CC-graph.*

Definition 9 (Complex-CC-cycle). *a cycle in a CC-graph that contains at least one vertex with differently weighted outgoing edges.*

Fig. 4 shows examples of graphs and edges defined above. Please note that, CC-graph is the transformation of the corresponding SC-graph. Although there is no new information generated compared to the original SC-graph, such transformation is necessary and valuable, because CC-graph facilitates our design of dynamic conflict detect algorithm.

More precisely, the underlying idea of these definitions is to represent the conflict relationship as numerical values so as to enable the conflict detection via algorithm. Here "negative connection" and "complex-cc-cycle" correspond to the two conflict scenarios to be detected and can be easily handled by detecting algorithm as shown in Section 5.2. The definitions of "SC-cycle-A" and "SC-cycle-B" are in fact intermediate notions connecting the SC cycle and CC-graph.

Proposition 1. *For an SC-graph and its derived CC-graph, if there exists a cycle of SC-cycle-A, then there must exist a corresponding negative connection relationship, and vice versa.*

Proof. By the Definition 4-A and Definition 7, the proposition obviously holds. □

Proposition 2. *For an SC-graph and its derived CC-graph, if there exists a cycle of SC-cycle-B, then there must exist a corresponding Complex-CC-cycle, and vice versa.*

Proof. By the Definition 4-B, Definition 7 and Definition 9, the proposition obviously holds. □

Theorem 2 (Extended chopping theory). *After transactions are chopped, if its CC-graph contains neither negative connections nor Complex-CC-cycles, we say that the chopping is safe (or correct). Correspondingly, all the chopped transactions in the SC-graph are called choppable.*

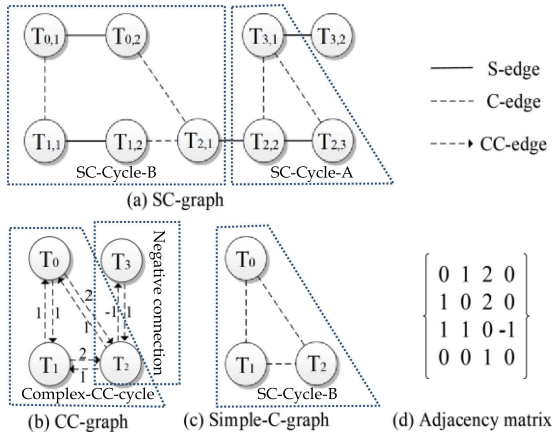


Fig. 4. Examples of graphs.

Proof. By Propositions 1 and Proposition 2, if the CC-graph of chopped transactions contains neither negative connections nor Complex-CC-cycles, there must exist no SC-cycle-A and no SC-cycle-B in the corresponding SC-graph. Then, there is no SC-cycle in the SC-graph. By Theorem 1, the chopping is safe (or correct). The theorem holds. □

For simplicity of presentation, if a hopping is unsafe due to the existence of *negative connection relationship* as described in Proposition 1, we say that the corresponding transactions are negatively unchoppable; if the chopping is unsafe due to the existence of Complex-CC-cycle as described in Proposition 2, we say that the corresponding transactions are simply unchoppable.

5.2 The Overall Framework

Algorithm 1 shows the overall framework of the DCD algorithm and also the execution procedure of transactions. The transactions are chopped by the dynamic chopper based on the partition service (described in Section 6). Based on the transaction pieces chopped, DCD firstly constructs the corresponding SC-graph, and then derives the CC-graph containing all transactions being executed and to be executed. The major operations are presented as the function update_graph() in Algorithm 1, and the details are described in Section 5.2.1. (Please note that, DCD can be executed upon transaction batches so as to reduce the cost of dynamic CC-graph update and conflict detection. Of course the size of a batch should be set upon the application requirements and system configurations.)

Then, we need to detect conflicts and unchoppable transactions. By Theorem 2, unchoppable transactions can be found by detecting negative connections and complex-CC-cycles, which correspond to negative unchoppable transactions and simple unchoppable transactions.

Negatively unchoppable transactions can be detected during the update of CC-graph, but the detection of simply unchoppable transactions is much more complex. Detection of simply unchoppable transactions is realized by searching complex-CC-cycle and the major operations are listed as the function detect_conflict() in Algorithm 2. The detailed

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

operations of simple unchoppable transactions are described in Section 5.2.2.

Algorithm 1. Overall Framework of DCD

```
    Input: existing CC_graph, new transaction T;
 1. Begin
 2.   invoke update_graph(T); //update the CC-graph;
 3.   invoke detect_conflict(-1, T, CC_graph); //Algorithm 2
 4.   invoke execute_txn(T);
 5.   return output from execute_txn();
 6. End
 7.
 8. function update_graph(T){//at DCA
 9.   for each c-edge {T, T'}:
10.     //h and h' are the adjacent hop in T and T'
11.     if CC_graph[T][T'] is assigned:
11.       CC_graph[T][T'] = -1;
13.       T.confilct_hop.append(CC_graph[T][T'], p);
14.     else:
15.       CC_graph[T][T'] = h;
16.       CC_graph[T'][T] = h';
17. }
18.   function execute_txn(T){//at Coordinator
19.   merge all hops in T.confilct_hop;
20.   output = sync_execute_piece(T.first_piece);
21.   parallel for p_i in T:
22.     async_execute_piece(p_i);
23.   return output
24. }
```

After conflict detection, the execution of the new transaction T (or the batch) will be invoked at the corresponding coordinator (the function execute_txn()), as described in Section 4.2. In this phase, DCD attempts to solve the conflict by merging transaction pieces, so as to transfer an unsafe chopping to safe one while keep piecewise execution as much as possible, as described in Section 5.2.3.

In the rest of this section, we present detailed operations of our DCD algorithm. Since the correctness of DCD can be easily proved base on Theorem 2, we omit the proof.

5.2.1 Constructing CC-graph and Detecting Negatively Unchoppable Transactions

To detect unchoppable transaction, DCD constructs the SC-graph, and construct CC-graph based the SC-graph CC-graph. The CC-graph is stored by an adjacency matrix G, with each element G[i][j] = CC_graph[i][j] representing the weight of edge e_{ij} in CC-graph. As an example, the CC-graph in Fig. 4b has the adjacency matrix in Fig. 4d.

The CC-graph is constructed in an incremental way. Upon a new transaction or a new batch, new edges connecting the new transactions are added (the function update_graph(T) in Algorithm 1). The negatively unchoppable transactions are detected during the construction of the edges of CC-graph, and the conflict relationship between transaction pieces is recorded accordingly (Lines 11-13).

Please note that, the vertexes in the graphs in our design are different from those in static chopping analysis in Lynx, because each instance (corresponding to a request) of the same transaction is denoted as a vertex.

5.2.2 Detecting Simply Unchoppable Transactions

The operations to detect simply unchoppable transactions are shown in Algorithm 2. Based on the CC-graph, DCD derives the corresponding simple-C-graph by removing negative connection edges, i.e, removing the negative values in the data structure of CC_graph. Then, DCD traverses the simple-C-graph following the DFS paradigm to detect possible cycles (Line 11, Line 15).

Algorithm 2. DFS of the DCD

```
    Input pre: = the previously visited transaction
      t: = the currently visited transaction
 1. function detect_conflict(pre, T, CC_graph){
 2.   label T as visited;
 3.   T.in-piece = CC_graph[T][pre];
 4.   T.ts = T.low = timestamp;
 5.   timestamp ++;
 6.   for each trans. T' with CC_graph[T][T'] > 0:
 7.     T.out-piece = CC_graph[T][T'];
 8.     if T' is not visited:
 9.       detect_conflict (T, T', CC_graph);
10.       T.low = min(T.low, T'.low);
11.     else if T'.ts < T.ts and T' ! = pre then //a cycle
12.       T.low = min(T.low, T'.ts);
13.       if T'.out-piece ! = CC_graph[T'][T]:
14.           T'.confilct _hop.append(T'.out-piece, CC_graph
              [T'][T])
15.     if T.low < T.ts and T.in-piece ! = T.out-piece:
16.       T.confilct _hop.append(T.in-piece, T.out-piece);
17. }
```

The cycle in Simple-C-graph can be checked based on the following well known conditions:

A vertex u in a graph is in a cycle if and only if $low[u] < ts[u]$ or u is incident with a back edge.

The $ts[u]$ represents the timestamp that u is visited and $low[u]$ refers to the least timestamp u's ancestors that u or its descendants connects via a back edge in the DFS tree.

Based on the cycle detection, unchoppable transactions can be determined. If a transaction provides one or more S-edges for a cycle, DCD will analyze the weights of CC-edges corresponding to the cycle edges in Simple-C-graph. Based on the conditions in Proposition 2, DCD can finally determine whether the involved transactions are choppable. The safety checking procedure also records the conflict hops that providing S-edge for the subsequent execution (Line 14 and Line 16).

5.2.3 Piece Merging and Conflict Solving

After the detection of negatively and simply unchoppable relation, the conflict hops that result in S-edges in an SC-cycle are all recorded (Line 13 in Algorithm 1, Line 14 and Line 16 in Algorithm 2).

To preserve the serializability of transaction executions, the conflicts in SC-cycle should be eliminated. The straightforward way is to merge all the hops of a conflicting transaction and execute it as a whole.

The execution of such a transaction should be executed under distributed concurrency control and atomic commit protocol, like 2PL + 2PC or other similar protocols. The concurrency control can preserve the serializability of merged hop

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

and original hop of other transaction. Of course, such concurrency control will introduce time and message cost. This is not caused by our design. It is in fact necessary for all transaction management systems.

However, not all the S-edges are involved in an SC-cycle, as shown in Fig. 5. Then, a transaction can be partially merged, and a conflicting transaction can still be executed in pieces. More precisely, for an SC-cycle detected, the S-edges directly constituting the SC-cycle should be merged, but others should be kept.

We can find a simple example in Fig. 5. The chains of T_1 and T'_1 constitute an SC-cycle. To solve conflicts, we claim $T_{1,1}-T_{1,2}$ and $T'_{1,1}-T'_{1,2}$ should be merged. However, for T_2 and T'_2, the chopping of $T_{2,1}-T_{2,2}$ and $T'_{2,1}-T'_{2,2}$ can be kept, and only $T_{2,2}-T_{2,3}$ and $T'_{2,2}-T'_{2,3}$ need to be merged to solve the conflicts of SC-cycle between T_2 and T'_2.

6 Partition Service

The partition service is an independent module for determining and maintaining the partition scheme of transactions. Partition service involves two key issues: partition strategy design and partition scheme update. The restrictions of the partition service are discussed in the end of this section.

6.1 Partition Strategy

The partition scheme is determined by its partition strategy, i.e, the criterion used to calculate a new scheme. Partition strategy should be defined by the upper layer applications or users according to specific requirements. With the modular design, different strategies can be easily installed to DTC.

1) Using existing data shard strategies in distributed storage systems, e.g., user locality based strategy and hotspot based strategy.

With the user locality based strategy, data tables are partitioned according to the location of clients. That is, a data shard should be placed at datacenters or data nodes near the clients.

Algorithm 3. Partition Scheme Synchronization

```
1.  function init_new_scheme (scm){
2.    //for leader datacenter
3.    construct the new scheme scm;
4.    scmstate = changing;
5.    block new requests from clients;
6.    send new(scm) to each other datacenter;
7.  }
8.  function update_scheme(){
9.    //for each datacenter, including the leader
10.   //upon receiving new(scm)
11.   if scmstate = changing: //in updating
12.     block execution of new hops from sender;
13.     if all other datacenters have been blocked and no
        hops using old scheme runing:
15.       scmstate = done;
16.       resume execution of hops;
17.   else: //not in updating yet
18.     scmstate = changing;
19.     block execution of new hops from sender;
20.     send new(scm) to each other datacenter;
21. }
```

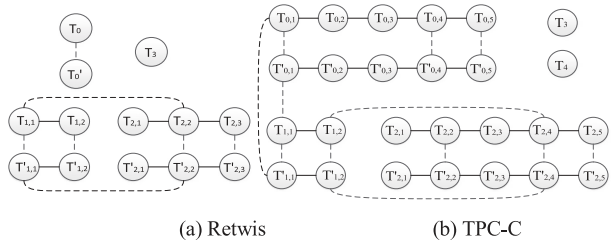


Fig. 5. SC-graphs for Retwis and TPC-C.

With hotspot based strategy, data shards are determined according to the data access frequency. Such a strategy is popularly used in data replication management.

2) Using special designed strategy for DTC. It is more interesting to design special transaction chopping aware partition strategy. That is, how to partition data is determined by conflict relationship among corresponding transaction pieces. Pieces causing conflicts should be merged and correspondingly the data shards should be merged. Please note that, here we mean to change the chopping strategy for future instances of a specific transaction, rather than the one that has started or completed.

6.2 Partition Scheme Update

All the datacenters in a geo-replicated system should use consistent partition scheme to ensure the transaction execution algorithms can work correctly.

The scheme can be constructed and maintained in two modes, i.e., static or dynamic. In static partition service, the partition scheme is predefined and will not be changed during execution, or can only be manually changed.

On the other hand, a better way is to dynamically change partition schemes in runtime, which is more flexible. However, how to synchronously update the partition scheme at multiple datacenters is a challenging issue.

A straightforward method is that, globally blocking new transaction requests at each datacenter, completing running ones, and then replicating new scheme among datacenters. Such a method is simple but obviously inefficient since the storage service is suspended during update.

To realize efficient partition scheme update, we propose a synchronization algorithm (i.e, Algorithm 3), which is inspired by the concept of distributed snapshot.

We assume that a leader datacenter is assigned to monitor the workload and generate new partition scheme. After a new scheme is generated, the leader switches to "changing" state: it will stop processing new transaction requests from clients and send scheme update messages to other datacenters (Line 4-Line 6).

Upon receiving an update message, the receiver datacenter will enter "changing" state and propagate the message to other datacenters. If it is already in "changing" state, it just blocks new hop execution requests from the sender. When update message has been received from each datacenter, the update can be terminated and transaction execution can be resumed using the new scheme.

Due to page limit, we do not include algorithm details, such as election of leader, loss of message. we also omit the proof of the correctness of Algorithm 3.

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

6.3 Restrictions and Typical Usage

The partition service relies on the knowledge of data shards to be accessed by transactions. Similar to [29], there are some restrictions on the usage of piecewise execution.

First, a transaction must be chopped into a chain such that (a) only its first hop contains a user-initiated abort, and (b) the shards it accesses at each hop are known before the chain starts executing. In other words, the transactions must have known read and write sets. As suggested in [29], a general transaction can be transformed into one with known read/write sets by applying the ideas of [35].

Second, to achieve low latency, the programmers must design the transactions so that, most of the time, the application can proceed after the chains complete their first hop (or first few hops). This is because, returning after the first hop may result in the loss of external consistency and, if misused, can generate user-perceived anomalies.

Due to the above restrictions, DTC is not a universal technique. It is suitable for large scale Web applications rather than traditional enterprise systems. Such usage scenario has been discussed in [29].

7 Performance Evaluation

To show the feasibility of our design and evaluate its performance, we implement DTC and conduct experiments in a testbed.

7.1 Experimental Setup

A virtualized cluster system using VMware is constructed to deploy a prototype of DTC and run our experiments. Each node has a virtualized single-core 3.1 GHz Intel Core and 1 GB of RAM.

7.1.1 Implementation

DTC is implemented by extending the source code of Lynx [29]. The software stack of the prototype is shown in Fig. 6. Corresponding to the architecture in Fig. 2, the DTC module implemented mainly consists of the DCD algorithm and the dynamic chopper, and the server module is copied from Lynx. The client module is responsible for generating transaction requests. We deploy the implemented system at a cluster of VMs, and each VM simulates a datacenter.

There are two types of communications across different VM nodes. The transaction invocation between DTC and Server is realized via a customized RPC library [32] as in typical datacenter systems. We use ZeroMQ [34], a popular message queue library, to direct transaction requests between different DTCs for a local first hop and monitor the state of transaction executions. To monitor the execution statement, a notification message will be sent to the DTC module after the last piece of a transaction is completed and committed. The average execution time of a piece in a single node is set to be less than 10ms, while the communication delay among different nodes is set to be larger than 100 ms.

User data to be processed by transactions are stored in an in-memory database, and the database integrally offers read-write operations in rational tables and uses 2PL for concurrency control. The storage scheme is implemented according to the description in Sections 3.1 and 4.1.

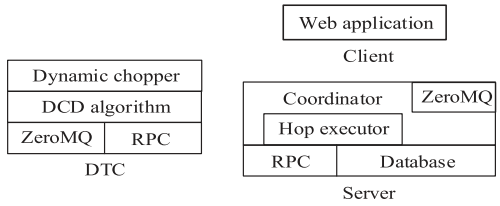


Fig. 6. Modules of implementation.

Moreover, in the prototype system, a global and centralized partition service is implemented to maintain the consistent partition scheme. For simplicity, the partition service offers a static partition scheme determined by the SC-graph.

For comparison purpose, we use Lynx as the baseline since our design is based Lynx. For fairness of comparison, we set the same partition scheme for both Lynx and DTC. The difference is that, conflicts in Lynx are detected manually against the SC-graph of all transactions of the application, while DTC detects conflicts using our dynamic detection algorithm and determines piecewise execution during running.

7.1.2 Applications

We evaluate the performance of DTC using several applications, including a social network service Retwis [30], a typical OLTP benchmark TPC-C [33], and two customized and complex applications.

Retwis is an open-source clone of Twitter. It mainly provides functions for Put and Get operations on Redis [31]. As in [28], each function implemented as a transaction. The workload distribution among different transaction are shown in Table 1.

The static relationship among the transactions in Retwis is represented by the SC-graph in Fig. 5a. The transactions are numbered according to the Table 1. Please note that, in our experiments, Lynx chops transactions statically according to the static SC-graphs in Fig. 5. For Retwis, two instances for each non-read-only transaction are included. There are SC-cycles constituted by T_1 and T_2, i.e, the Follow/Unfollow transaction and the Post Tweet transaction. Therefore, static conflict analysis by Lynx will determine that such transactions are unchoppable.

The application TPC-C [18] is a typical OLTP benchmark. TPC-C consists of three highly-contending read-write transactions and two read-only transactions. The SC-graph of TPC-C are illustrated in Fig. 5b. The workload distribution of TPC-C transactions is shown in Table 2.

Besides these two applications, we also consider two complex scenarios as shown in Fig. 7. The corresponding SC-graphs are randomly generated and quite complex. These two customized applications can show the advantage of DTC in complex and large applications.

Furthermore, we conduct additional evaluation about partition service and improved DCD algorithm based on these two customized applications.

7.1.3 Workload and Measurement

The transaction requests are handled in batch. To simulate different workload levels, we set different time intervals between batches, varying from 100 ms to 800 ms. The batch interval

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Transaction Workload in Retwis

No.	Transaction	Type	Workload
0	Add User	Read-write	5%
1	Follow/Unfollow	Read-write	15%
2	Post Tweet	Read-write	30%
3	Load Timeline	Read-only	50%

TABLE 2
Transaction Workload in TPC-C

No.	Transaction	Type	Workload
0	New Order	Read-write	44.97%
1	Delivery	Read-write	4.00%
2	Payment	Read-write	43.00%
3	Order Status	Read-only	4.03%
4	Stock Level	Read-only	4.00%

determines the degree of concurrency since a larger interval results in less concurrent transactions and vice versa.

The performance of all the applications are measured using two metrics. The piecewise ratio is defined to be the percentage of transactions (instances) that are executed in piecewise over all the transactions (instances) that are executed in the system. Piecewise ratio directly indicates the effectiveness of chopping techniques.

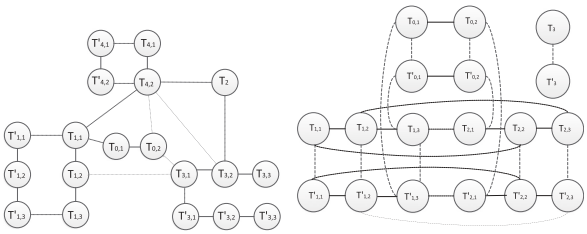
The other metric is the average execution time of a batch. This metric shows the benefit of chopping in the point of the view of users.

For the implementation of DTC, we include two versions, i.e, the original DTC without conflict solving mechanism and the improved version with conflict solving mechanism. For comparison purpose, we also tested the transaction chain technique in Lynx system [29].

In the following, we present and discuss experiment results according to the applications. In the figures plotting results, the X axis represents the batch interval.

7.2 The Results of Retwis

The results of Retwis are shown in Fig. 8. Under different workload levels (i.e, batch intervals), the piecewise ratio of Lynx keeps to be about 55 percent, while the piecewise ratio of DTC changes from 60 to 90 percent. Lynx detects conflicts using the static SC-graph of all possible transactions, and whether a transaction can be executed in piecewise is determined during programming and not affected by batch interval. On the contrary, DTC detects conflicts in runtime and allows larger percentage of piecewise executions under larger batch



(a) E-commerce (b) Social network

Fig. 7. SC-graphs for two complex applications.

(a) Piecewise ratio

(b) Average execution time (milliseconds)

Fig. 8. Performance of Retwis.

intervals (fewer conflicts may occur). Such difference shows effectiveness of the dynamic conflict detection of DTC.

Moreover, the improved DTC can achieve a larger piecewise ratio than original DTC under different workload levels, and the difference is about 5 percent, which is a quite obvious improvement.

Fig. 8b shows the average execution time of one Retwis batch. The execution time of a batch is definitely affected by piecewise ratio. When the piecewise ratio is higher, more transactions that can execute piecewise and the average execution time of batches will be lower accordingly. Since the piecewise ratio of Lynx is not affected much by batch interval, its execution time also keep stable.

7.3 The Results of TPC-C

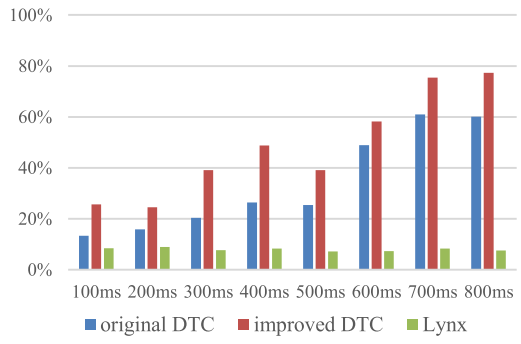
The performance results of TPC-C are plotted in Fig. 9. Roughly, the piecewise ratio of TPC-C is obviously lower than that of Retwis. This is because the read-write transactions, which may lead to SC-cycle, account as high as 92 percent of the workload in TPC-C. Moreover, transactions in TPC-C are larger than those in Retwis in terms of number of operations, as shown in Fig. 6, so more conflicts may occur and less percentage of piecewise executions are allowed.

Similar to that in Fig. 8, Lynx is almost not affected by workload level, while DTC can achieve larger piecewise ratio and lower execution time with the increase of batch level. DTC, with or without the conflict solving mechanism, can always achieve a higher piecewise ratio than Lynx.

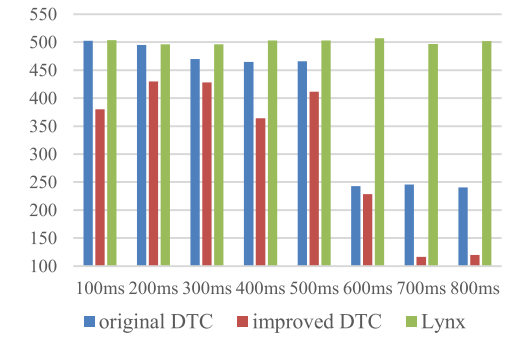
7.4 The Results of Two Complex Applications

In this subsection, we simulate two more applications as shown in Fig. 7. The two complex applications are from

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.



(a) Piecewise ratio



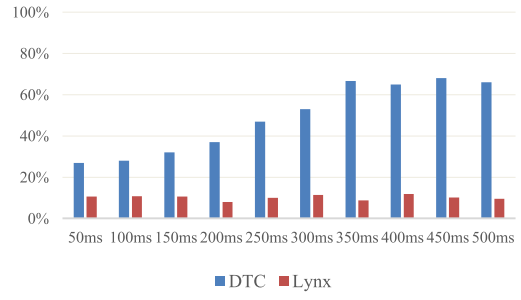
(b) Average execution time (milliseconds)

Fig. 9. Performance of TPC-C.

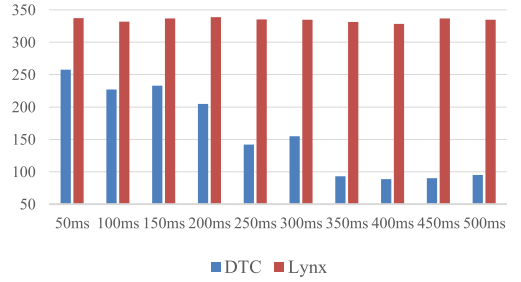
E-commerce and social networking scenarios and they are much more complex than Retwis and TPC-C.

The evaluation results are shown in Figs. 10 and 11, for piecewise ratio and the average execution time, respectively. From the results, we can observe that, under each workload level, DTC can achieve higher piecewise ratio and lower execution time.

Comparing the results in Figs. 10 and 11 with those in Figs. 9 and 8, we can find that, the advantage of DTC in complex transactions is larger than that in Retwis/TPC-C.

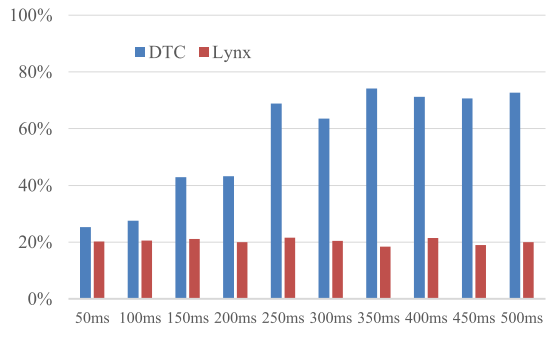


(a) Piecewise ratio

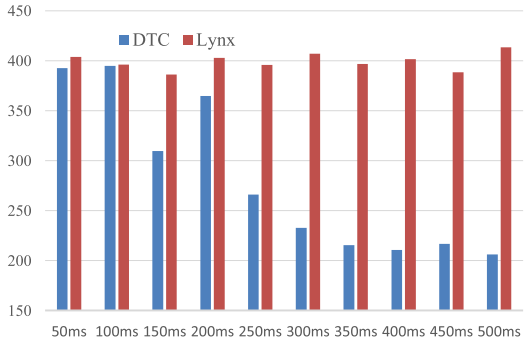


(b) Average execution time (milliseconds)

Fig. 10. Performance of E-commerce.



(a) Piecewise ratio



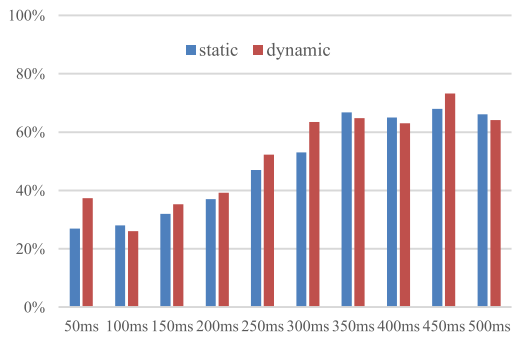
(b) Average execution time (milliseconds)

Fig. 11. Performance of social network.

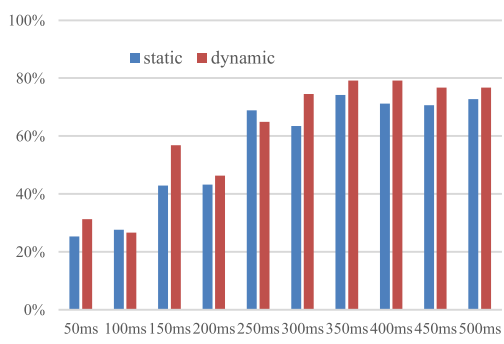
This means that, dynamic conflict detection is especially suitable for large and complex transactions.

7.5 The Results of Dynamic Partition Scheme

In this part, we conduct experiments with dynamic partition scheme described in Section 6 to evaluate its effectiveness



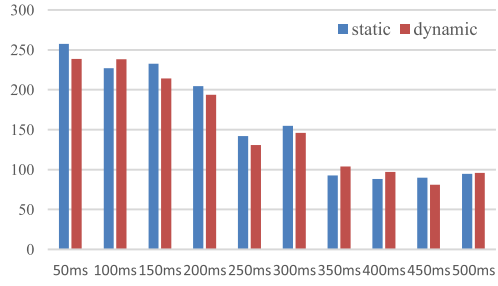
(a) E-commerce



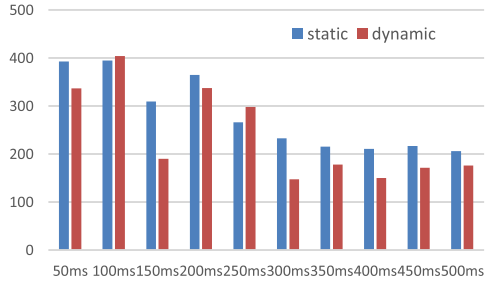
(b) Social network

Fig. 12. Piecewise ratio.

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.



(a) E-commerce



(b) Social network

Fig. 13. Average execution time (milliseconds).

and efficiency, and compare it with static partition scheme. More precisely, we initiate the partition using small shards of data item level, and then dynamically update the partition by merging shards according to the piecewise ratio obtained via historical log.

Therefore, we have two versions of DTC. The "static" refers to the DTC version with static partition scheme while "dynamic" refers to the DTC version with dynamic partition scheme. We still use the two complex applications in Section 7.4. The results are shown in Figs. 12 and 13.

The results show that, the implemented dynamic partition service can further improve piecewise ratio and reduces execution time, although the dynamic partition strategy is simple. This indicates that, partitioning data according to transaction execution status is effective and should be considered in real deployment.

8 Conclusion

In this paper, we propose DTC, a technique to achieve low latency and strong consistency in geo-replicated storage systems. The key novelty of our work lies in dynamic transaction chopping and runtime conflict analysis to realized safe and efficient piecewise transaction execution. DTC mainly consists of chopping mechanism to divides a transaction into pieces dynamically according to data partition scheme, and a conflict detection algorithm for determining the safety of piecewise execution. DTC is more effective and efficient than existing transaction chopping based geo-replication techniques, because DTC needs not change applications and allows more piecewise executions.

Transaction chopping is an interesting approach for geo-replicated datacenters and should be further explored. Many possible extensions and improvements can be considered, including distributed conflict detection and analysis mechanisms, better design of data partitioning and transaction chopping, and so on.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB0203803, in part by the National Natural Science Foundation of China under Grants U1711263, U1801266, and U1811461, in part by the U.S. National Science Foundation under Grants OAC-1725755 and OAC-2104078, and in part by the Guangdong Provincial Natural Science Foundation of China under Grant 2018B030312002.

References

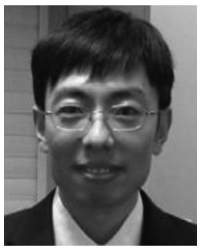
[1] D. J. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *IEEE Comput*, vol. 45, no. 2, pp. 37–42, Feb. 2012.
[2] M. S. Ardekani and D. B. Terry, "A self-configurable geo-replicated cloud storage system," in *Proc. USENIX Symp. Oper. Syst. Des. Implementation*, 2014, pp. 367–381.
[3] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Commun. ACM*, vol. 56, no. 5, pp. 55–63, 2013.
[4] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Bolt-on causal consistency, " in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 761–772.
[5] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," in *Proc. ACM SIGMOD Rec. Int. Conf. Manage. Data*, 1995, pp. 1–10.
[6] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, 2008.
[7] B. F. Cooper *et al.*, "PNUTS: Yahoo!'s hosted data serving platform," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.
[8] J. C. Corbett *et al.*, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 1–22, 2013.
[9] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," in *Proc. ACM SIGOPS Oper. Syst. Rev.*, 2007, pp. 205–220.
[10] E. P. C. Jones, D. J. Abadi, and S. Madden, "Low overhead concurrency control for partitioned main memory databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 603–614.
[11] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete, "MDCC: Multi-data center consistency," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2013, pp. 113–126.
[12] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.
[13] L. Lamport, "Fast paxos," *Distrib. Comput.*, vol. 19, no. 2, pp. 79–103, 2006.
[14] Y. Lin, B. Kemme, M. P. Martínez, and R. Jimenez-Peris, "Enhancing edge computing with database replication," in *Proc. IEEE Int. Symp. Reliable Distrib. Syst.*, 2007, pp. 45–54.
[15] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in *Proc. Presented Part USENIX Symp. Networked Syst. Des. Implementation*, 2013, pp. 313–328.
[16] H. Mahmoud, F. Nawab, A. Pucher, D. Agrawal, and A. E. Abbadi, "Low-latency multi-datacenter databases using replicated commit," *Proc. VLDB Endowment*, vol. 6, no. 9, pp. 661–672, 2013.
[17] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *Proc. ACM Symp. Operating Syst. Princ.*, 2013, pp. 358–372.
[18] S. Mu, Y. Cui, Y. Zhang, W. Lloyd, and J. Li, "Extracting more concurrency from distributed transactions," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 479–494.
[19] S. Mu, L. Nelson, W. Lloyd, and J. Li, "Consolidating concurrency control and consensus for commits under conflicts," in *Proc. USENIX Symp. Oper. Syst. Des. Implementation*, 2016, pp. 517–532.
[20] F. Nawab, V. Arora, D. Agrawal, and A. E. Abbadi, "Minimizing commit latency of transactions in geo-replicated data stores, " in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1279–1294.
[21] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *Proc. USENIX Symp. Oper. Syst. Des. Implementation*, 2010, pp. 1–15.

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.

[22] S. Roy *et al.*, "The homeostasis protocol: Avoiding transaction coordination through program analysis," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1311–1326.
[23] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Comput.*, vol. 40, no. 9, pp. 103–105, Sep. 2007.
[24] D. Shasha, F. Llirbat, E. Simon, and P. Valduriez, "Transaction chopping: Algorithms and performance studies," *ACM Trans. Database Syst.*, vol. 20, no. 3, pp. 325–363, 1995.
[25] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for Geo-replicated systems," in *Proc. ACM Symp. Oper. Syst. Princ.*, 2011, pp. 385–400.
[26] A. Thomson, T. Diamond, S. C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: Fast distributed transactions for partitioned database systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 1–12.
[27] C. Xie, C. Su, C. Littley, L. Alvisi, M. Kapritsos, and Y. Wang, "High-performance ACID via modular concurrency control," in *Proc. Symp. Oper. Syst. Princ.*, 2015, pp. 279–294.
[28] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. Ports, "Building consistent transactions with inconsistent replication," in *Proc. Symp. Oper. Syst. Princ.*, 2015, pp. 263–278.
[29] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li, "Transaction chains: Achieving serializability with low latency in Geo-distributed storage systems," in *Proc. ACM Symp. Oper. Syst. Princ.*, 2013, pp. 276–291.
[30] C. Leau, "Spring data redis – retwis-J," 2013. [Online]. Available: http://docs.spring.io/spring-data/data-keyvalue/examples/retwisj/current/
[31] Redis, "Open source data structure server," 2013. [Online]. Available: http://redis.io/
[32] Simple RPC in C++, Accesed: Jun. 2021. [Online]. Available: https://github.com/santazhang/simple-rpc
[33] TPC Benchmark C, Accesed: Jun. 2021. [Online]. Available: http://www.tpc.org/tpcc/
[34] ZeroMQ, Accesed: Jun. 2021. [Online]. Available: http://www.zeromq.org/
[35] A. Thomson and D. J. Abadi, "The case for determinism in database systems," *Proc. VLDB Endowment*, vol. 4, pp. 70–80, 2010.

Ning Huang received the BSc and MSc degrees from the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, in 2015 and 2018, respectively. He is currently with Tencent Inc. His research interests include distributed systems, data replication, and transactional mechanism.

Lihui Wu received the MSc degree from the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, in 2017. She is currently with GF Fund Management Company. Her research interests include distributed systems, data replication, and parallel state machine optimization.



Weigang Wu (Member, IEEE) received the BSc and MSc degrees from Xi'an Jiaotong University, China, in 1998 and 2003, respectively, and the PhD degree in computer science from Hong Kong Polytechnic University in 2007. He is currently a full professor with the School of Computer Science and Engineering, Sun Yat-sen University, China. He has authored or coauthored more than 100 papers in major conferences and journals. His research interests include distributed systems, cloud computing, and big data processing. He was an organizing/program committee member for many international conferences.

Sajal K. Das (Fellow, IEEE) is currently a professor of computer science and Daniel St. Clair Endowed chair with the Missouri University of Science and Technology, Rolla. From 2008 to 2011, he was with the NSF as a program director with the Division of Computer Networks and Systems. He has authored more than 650 research articles, five U.S. patents, 52 book chapters, and four books. His current research interests include parallel and cloud computing, wireless sensor networks, mobile and pervasive computing, cyber-physical systems, smart environments, big data and IoT, and security. He is one of the most prolific authors in computer science according to DBLP. His h-index is 75 with more than 23 500 citations according to Google Scholar. He was the founding editor-in-chief of Elsevier's *Pervasive and Mobile Computing* journal and as an associate editor for the *Journal of Parallel and Distributed Computing*, *IEEE Transactions on Mobile Computing* and *ACM Transactions on Sensor Networks*. He was the general chair, technical program chair, and program committee member of numerous ACM and IEEE conferences. He was the recipient of ten best paper awards and numerous awards for research, teaching, mentoring, and professional service, including the IEEE Computer Society's Technical Achievement Award for pioneering contributions to sensor networks and mobile computing.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Missouri University of Science and Technology. Downloaded on September 04,2025 at 21:08:26 UTC from IEEE Xplore. Restrictions apply.