Improving Transaction Processing Performance By Consensus Reduction

Yuqing ZHU, Yilei WANG
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
Email: {zhuyuqing,wangyl}@ict.ac.cn

***Abstract*—Transaction and high availability are both important to applications. While data partition, distribution and replication are the three key mechanisms to guarantee high availability, a coordination to reach consensuses on *replica state transitions*, *transaction operation orders* and *commit decisions* is required for transaction processing. This coordination impairs transaction processing performance. In classic transactional approaches, lock granularity is exploited to trade off consistency for performance under weaker isolation levels. We question whether transaction processing performance can be similarly improved under weaker isolation levels by consensus reduction.**

To carry out consensus reduction, we categorize transactions based on their *scopes of consensus*, as well as their *requirements on consensus*. A transaction in one category can be reduced to multiple transactions in other categories with smaller consensus scope and weaker consensus requirement. We theoretically analyze what anomalies the reduction can lead to. We thus find and define eight isolation levels by anomaly sets. We experiment to find out how these weaker isolation levels can improve transaction processing performance. Interesting results show that three weak isolation levels improve performance, while the weakest isolation level has the worst performance. Results in this paper enable users to actively choose the appropriate isolation levels for their applications.

***Keywords*-transaction processing; concurrency control; coordination; consensus reduction; isolation level; consistency**

I. INTRODUCTION

Transactional support can turn hours of big data processing into milliseconds of simple queries by enabling fast access in query processing, data warehousing, data visualization, etc., through general system functions like incremental processing [1] and materialized view maintenance [2], [3]. It continues to be recognized as a valuable functionality to applications [4]. At the same time, high availability is taken as fundamental to online services [5], [6]. Several approaches to supporting strong transactional semantics in highly available datastores are proposed in recent years, e.g., MDCC [7], Helios [8], and Hyder [9].

The three fundamental mechanisms for high availability, i.e. data partition, distribution and replication, complicate transaction processing [10]. So does the fault tolerance requirement implied by high availability. A transaction has to coordinate all data partitions and their replicas it attempts to access, so as to reach *consensuses on replica state transitions*, *transaction operation orders* and *commit decisions* [11], [8]. Without consensuses on replica state transitions, the states of multiple replicas can diverge increasingly [12]. Consensuses on transaction operation orders and commit decisions among data partitions are also key to correct and consistent database states [13]. The coordination for reaching multiple consensuses takes multiple communications across servers and limits transaction concurrency, thus impairing performance. Though coordinator-less transaction processing is possible, it requires a close and extensive study of every application [14], [15].

Conventionally, database systems offer weaker isolation levels to applications for better transaction processing performance [16], [17], [18]. For example, database systems adopting locking-based protocols implement weak isolation levels by locking with smaller granularity and shorter duration [16]. As the new transactional proposals [7], [11], [8] for distributed highly available datastores have a focus on consensus, we question *whether it is possible to improve transaction processing performance similarly in highly available datastores under weaker isolation levels by consensus reduction*.

In this paper, we propose to improve transaction processing performance by consensus reduction, which relaxes *the consensus scope* and *the consensus requirement* of transaction processing. This reduction can cause anomalies [17], which can result in different transaction execution results and database states satisfying various applications' requirements. We analyze what anomalies the reduction can lead to and define eight transaction isolation levels accordingly, exploiting the theoretical tool of serializability analysis [19], [17]. To find out how the new isolation levels can improve performance, we experiment with various deployment scenarios, varying the replica number, the replica distribution, and the replica locations. The experimental results are contrary to conventional impression that weaker isolation levels always improve performance. Only three weak isolation level can improve performance. The weakest isolation level leads to the worst performance.

In sum, we make the following contributions in this paper.

- We propose a consensus-based classification of transactions, and the reduction of a transaction type into types of transactions with smaller consensus scope and weaker consensus requirement.
- We theoretically analyze anomalies caused by consensus reduction, to help understand effects when a reduced transaction is decomposed into multiple transactions for execution. We find new anomalies that never occurred in conventional databases.

- We find and define eight weak transaction isolation levels for highly available datastores, through a consensus reduction approach different from classic approaches.
- We demonstrate that three weak isolation levels can improve transaction processing performance in highly available datastores.

The analyses and the definitions in this paper enable users to check which set of anomalies their applications can tolerate. As each anomaly set corresponds to an isolation level, they can thus find out which isolation levels are feasible for their applications. Among these isolation levels, they can then choose the isolation level with the best performance for their implementations.

The rest of the paper is organized as follows. Section II presents our classification of transactions along the two-dimensions of consensus scope and consensus requirement. Section III illustrates consensus reduction and transaction decomposition, along with an analysis of anomalies. Section IV summarizes anomaly sets for different decompositions and defines the eight isolation levels accordingly. Section V demonstrates our experimental results with the new isolation levels. Section VI reviews related work. Section VII draws the conclusion.

II. Transaction Consensuses and Categories

Conventionally, isolation levels are implemented by locking in different granularity and duration [16]. For consensus reduction, we similarly identify two aspects to be adjusted, i.e. *the scope of consensus and the requirement of consensus*.

The scope of consensus relates to the data scope of a transaction. The larger the consensus scope is, the greater possibility a transaction's data set intersects with others', and the greater this intersection can possibly be. The greater transactions intersect, the more coordination is needed and the more concurrency is compromised. In fact, transactions executed on non-intersecting sets of data can be concurrently processed with serializability even without coordination. Thus, the scope of consensus indicates the degree of concurrency.

The requirement of consensus includes the agreement on replica state transitions, transaction operation orders and commit decisions. The more agreement in involved, the more coordination is needed, and the stronger the requirement of consensus is. Some read-only transactions might have no consensus requirement, while others have strong consensus requirement. The consensus requirement is fulfilled on a transaction's commitment, which is a coordination point extending the transaction execution time and thus reducing concurrency. That is, the requirement of consensus is also an indication of concurrency possibility.

Categorization. We categorize transactions according to the two aspects. For the consensus scope, we divide transactions into three categories, i.e., operation-unit, intra-shard, and cross-shard. Transactions in the operation-unit category are composed of one single operation. Transactions in the intra-shard category involve multiple operations on data of the same data shard (a.k.a., data partition), while those in the cross-shard category operate on data across multiple shards. For each of the three categories, transactions can further be divided into multiple types according to their consensus requirement. All transaction categories and types are illustrated along the two axes of consensus scope and requirement in Figure 1.

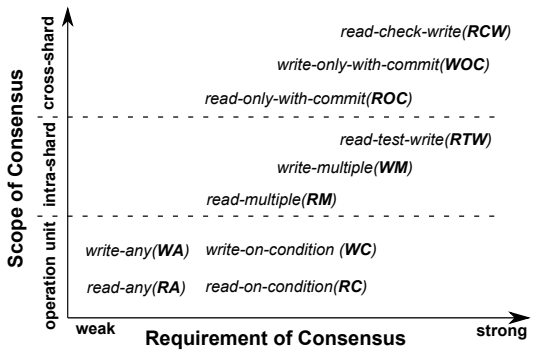

Figure 1. Consensus-oriented classification of transactions.

Operation-Unit Transactions. Four types of transactions are in this category, i.e. read-any (RA), write-any (WA), read-on-condition (RC) and write-on-condition (WC). RA and WA require no consensus in processing, while RC and WC requires only consensus local to the data shard. RC and WC transactions make stronger guarantees than RA and WA by checking the right state before execution, e.g., the value of the most recent state or the state after a certain time.

Intra-Shard Transactions. Transactions in this category include read-multiple (RM), write-multiple (WM), and read-test-write (RTW). RTW transactions first read all involving data, then test whether conditions are met, and finally apply writes (or simply abort). The execution of RM requires a database state satisfying all its reads' conditions. Then the multiple reads retrieve values and return, with no writes intervening their executions. WM is processed similarly as RM, though WM requires a consensus on replica state transitions while RM does not. The execution of all transactions in this category must base on the consensus on replica states.

Cross-Shard Transactions. Transactions in this category require consensus on both state transitions among replicas and operation sequences across multiple shards. Such transactions include read-only-with-commit (ROC), write-only-with-commit (WOC), and read-check-write (RCW). RCW is the only type that requires a commit-decision consensus in this category. For RCW, the transaction can commit or abort, but the decision must be made on the *check*, which is after the necessary reads and before the writes.

III. Reduction and Decomposition

For consensus reduction, we decompose a transaction with a larger consensus scope and a stronger consensus

requirement into transactions with smaller consensus scopes and fewer consensus requirements.

A. Anomalies in Reduction and Decomposition

The consensus reduction and the transaction decomposition can lead to anomalies of database states and transaction execution results. We take a conflict-and-anomaly-based approach [17] by identifying all conflicting relations and all possible anomalies arising due to a transaction decomposition. To simplify the discussion, we assume that only the transaction under discussion is decomposed. All other concurrent transactions are assumed to be executed with atomicity, i.e. not decomposed. We then generalize the analysis to a sequence of interdependent transactions.

Conflict-based enumeration. Anomalies arise when conflicting operations, i.e. consecutive read-write and consecutive write-write on the same data, are not compatible with the serial execution of transactions. Identifying the different conflicting relations between transactions' operations can help define isolation levels [20]. Transactions with conflicting operations are in dependence relations. *Anomalies are dependence cycles of concurrent transactions which have operations in conflicting relations*. We thus first recognize all possible decompositions of a transaction type, then enumerate conflicting operation relations between involving transactions, and finally find out the resulting anomalies.

As operation-unit transactions cannot be further decomposed, we start the decomposition analysis from the intra-shard category. Note that anomalies can lead to incorrect results of the decomposed transaction (denoted as T1), as well as transactions (denoted as T2 and processed with atomic execution) processed concurrently to the decomposed one. We identify both and denote each with different names in the analysis.

Naming Rules. We name the resulting anomalies with two or three parts. The first part of an anomaly name comes from the name of a classic anomaly name. We relate each new anomaly with the eight widely-accepted classic anomalies [17], which are represented in operation sequences as follows. If there is only one anomaly relating to a classic anomaly, the name of the anomaly has two parts (e.g. *A3-rm1*); otherwise, three parts (e.g. *A5A-1-rm1*).

In the following, *r* stands for read, *w* for write, *c* for commit and *a* for abort. The number following an operation corresponds to the number of the transaction, e.g. *r1* for a read of transaction *T1*. The *x* or *y* in the brackets is the data item used by the preceding operation.

A0 w1[x]...w2[x]...(a1/a2)
A1 w1[x]...r2[x]...(a1/c2 in any order)
A2 r1[x]...w2[x]...((c1/a1) and (c2/a2) in any order)
A3 r1[P]...w2[y in P]...((c1/a1) and (c2/a2) in any order)
A4 r1[x]...w2[x]...w1[x]...c1
A5A r1[x]...w2[x]...w2[y]...c2...r1[y]...(c1/a1)
A5B r1[x]...r2[y]...w1[y]...w2[x]...(c1 and c2 occur)

The second part of a three-part name is the sequence number that differentiates the multiple anomalies related to the same classic anomaly. The second part of a two-part name and the third part of a three-part name consist of two components, i.e. the decomposed transaction type name and a number. For this number, *1* represents the anomaly causing the incorrect result of T1, while *2* for T2 and *0* for both.

B. Intra-Shard Transactions

Read-Multiple Transaction. RM can be decomposed into multiple RCs and/or RAs. The RA decomposition of RM leads to conflicting scenarios of (r1→w2, w2→r1) and (w2→r1, r1→w2). Enumerating all incompatible operation sequences, we get the following anomalies:

RM→RA(WM, RTW, WOC, RCW)

A3-rm1 r1[P]...w2[x,y in P]...c2...r1[y in P]... (**Phantom**)
A5A-1-rm1 r1[x]...w2[x]...w2[y]...c2...r1[y]... (**Read Skew**)
A5A-2-rm1 w2[x]...r1[x]...r1[y]...w2[y]...c2... (**Read Skew**)

RM can also be decomposed in to RC. Different from RA, RC is a read transaction that cannot fall between other transactions' operations. Therefore, removing anomalies with transaction T1's operations falling between other transactions, we get the following anomalies for the RC decomposition of RM.

RM→RC(WM, RTW, WOC, RCW)

A3-rm1 r1[P]...w2[x,y in P]...c2...r1[y in P]... (**Phantom**)
A5A-1-rm1 r1[x]...w2[x]...w2[y]...c2...r1[y]... (**Read Skew**)

Recall our assumption that all transactions are executed serially except for the decomposed T1. Hence, the above anomalies can cause not only by a single transaction, but also by multiple transactions linearly aligned with dependence relations, e.g. ...r1[x]...w2[x]w2[y] c2...w3[y]w3[z]c3...r1[z]... We then generalize the transaction T2 into a serially ordered sequence of transactions, of which two have operations conflicting with transaction T1. Thus, the A3-rm1 and the A5A-1-rm1 anomalies can be described in generalized forms as follows.

RM→RA/RC(WM, RTW, WOC, RCW)

A3-rm1 r1[P=P1∨ P2]...w2[x in P1]...c2...w3[y in P2] ...c3... r1[y in P]...(**Phantom**)
A5A-1-rm1 r1[x]...w2[x]...c2...w3[y]...c3...r1[y]...(**Read Skew**)

Anomaly A5A-2-rm1 cannot be generalized similarly. The cause of this anomaly is that transaction T1 falls between another transaction's operations. Generalization can invalidate this cause. Note that, decomposing RM into RA and RC leads to a result similar to that for the RA decomposition of RM. So does decomposing WM into WA and WC.

Write-Multiple Transaction. WM can be decomposed into multiple WAs and/or WCs. The WA decomposition of WM can cause conflicting scenarios including (w1→r2, r2→w1), (r2→w1, w1→r2), (w1→w2, w2→w1), (w2→w1, w1→w2), (w1→r2, w2→w1), and (r2→w1, w1→w2). We do not include the conflicting scenarios of (w2→w1,

w1→r2) and (w1→w2, r2→w1) because we assume all writes are applied after all reads in a transaction.

Six conflicting relations and anomalies exist. Further grouping anomalies by the type of transaction T2, we get the following anomaly set. Anomalies A6- are introduced. A6- series of anomalies cannot happen in classic databases because of the locking implementation and the two assumptions of locking, i.e., well-formed and two-phase [16].

WM→WA(RM,RTW,ROC,RCW)

A5A-1-wm2 r2[x]...w1[x]...w1[y]...r2[y]...c2...(**Read Skew**)
A5A-2-wm2 w1[x]...r2[x]...r2[y]...c2...w1[y]...(**Read Skew**)

WM→WA(WM,RTW,WOC,RCW)

A6-1-wm0 w1[x]...w2[x]...w2[y]...c2...w1[y]...(**Inconsistent Write**)
A6-2-wm0 w2[x]...w1[x]...w1[y]...w2[y]...c2...(**Inconsistent Write**)

WM→WA(RTW,RCW)

A4-wm2 w1[x]...r2[x]...w2[y]...c2...w1[y]...(**Lost Update**)
A6-3-wm0 r2[x]...w1[x]...w1[y]...w2[y]...c2...(**Inconsistent Write**)

The WC decomposition analysis of WM is similar to the RC decomposition analysis of RM. The following is the resulting anomaly set.

WM→WC(RM,RTW,ROC,RCW)

A5A-2-wm2 w1[x]...r2[x]...r2[y]...c2...w1[y]...(**Read Skew**)

WM→WC(WM,RTW,WOC,RCW)

A6-1-wm0 w1[x]...w2[x]...w2[y]...c2...w1[y]...(**Inconsistent Write**)

WM→WC(RTW,RCW)

A4-wm2 w1[x]...r2[x]...w2[y]...c2...w1[y]...(**Lost Update**)

For the conflicting relations between T1 and more transactions, anomalies leading to incorrect results of transaction T2 cannot be generalized for multiple transactions linearly aligned with dependence relations. We thus remove anomalies `A5Ax-2-wm2` and `A4x-wm2`. The conflicting scenario of (w1→w2, r2→w1) can happen between T1 and an ordered series of transactions, though not between T1 and T2. Thus, we get the following anomalies.

WM→WA/WC(WM,RTW,WOC,RCW)

A6-1-wm0 w1[x]...w2[x]...c2...w3[y]...c3...w1[y]...(**Inconsistent Write**)

WM→WA/WC(WM,RTW,WOC,RCW; RM,RTW,ROC,RCW)

A6-4-wm0 w1[x]...w2[x]...c2...r3[y]...c3...w1[y]...(**Inconsistent Write**)

Read-Test-Write Transaction. RTW can be decomposed in four ways, i.e., RC+WC, RC+WM, RM+WC, and RM+WM. RA and WA are not used because RTW must *test* the read value such that certain conditions must be met, i.e. RC and WC are required.

We first study the decomposed form closest to the original transaction, i.e., RM+WM. Since RM and WM are atomic transactions, $18 = 2 \times 3 \times 3$ conflicting scenarios exist. The *3* is for the three conflicting relations of r1 vs. w2 combined with w1 vs. r2 and w1 vs. w2. The *2* is because we can flip the two conflicting operations. Assuming writes executed after all reads in a transaction (or *two-phase execution*) and the execution atomicity of transaction T2/RM/WM, 17 conflicting scenarios cannot occur due to the requirement of double-writes/reads for T1, or a transaction's read coming after its write. Thus, there remains one conflicting scenario, i.e., (r1→w2, w2→w1). We group anomalies by the transaction type of T2, apply generalizations and thus get the following.

RTW→RMWM(WC,WM,RTW,WOC,RCW)

A4-rmwm2 r1[x]...w2[x]...c2...w1[x]...(**Lost Update**)

RTW→RMWM(RTW,RCW)

A5B-rmwm0 r1[x]...r2[y]...w2[x]...c2...w1[y]...(**Write Skew**)

RTW→RMWM(WM,RTW,WOC,RCW)

A6-rmwm0 r1[x]...w2[x]...w2[y]...c2...w1[y]...(**Inconsistent Write**)

To generalize the analysis to a series of dependent transactions, we first remove anomalies leading to only incorrect results of transaction T2, i.e., `A4x-rmwm2`. Then, among the 17 conflicting relations not occurring between T1 and T2, only (r1→w2, r3→w1) can occur to T1 and an ordered series of transactions. Thus, the following anomalies can occur between T1 and multiple transactions.

RTW→RMWM(WC,WM,RTW,WOC,RCW)

A4-rmwm2 r1[x]...w2[x]...c2...r3[x]...c3...w1[x]...(**Lost Update**)

RTW→RMWM(RTW,RCW)

A5B-rmwm0 r1[x]...r2[y]...c2...w3[x]...c3...w1[y]...(**Write Skew**)

RTW→RMWM(WM,RTW,WOC,RCW)

A6-rmwm0 r1[x]...w2[x]...c2...w3[y]...c3...w1[y]...(**Inconsistent Write**)

By further decomposing the RM or/and WM parts of RTW's RM+WM decomposition, we can obtain the other three decomposition schemes for RTW. It is easy to notice that by decomposing the RM part of the RM+WM decomposition, all anomalies caused by the decomposition of RM into RCs remain for the RC+WM decomposition. The same observation can be applied when decomposing the WM part of the RM+WM decomposition. We thus have the following proposition.

Proposition 1: Assume the decomposition of RTW into RM+WM causing the set $\mathbb{S}_0$ of anomalies, and that of RM into RCs causing $\mathbb{S}_x$, then the decomposition of RTW into RC+WM causes anomalies $\mathbb{S} = \mathbb{S}_0 \cup \mathbb{S}_x$.

Similar analyses can also be made for the RM+WC decomposition and the RC+WC decomposition of RTW. As RTW has the testing phase, *all RCs must finish before any WC starts for the RC+WC decomposition.* Hence, we can get the following propositions for the RM+WC decomposition and the RC+WC decomposition of RTW.

Proposition 2: Assume the decomposition of RTW into RM+WM causing the set $\mathbb{S}_0$ of anomalies, and that of WM into WCs causing $\mathbb{S}_x$, then the decomposition of RTW into RM+WC causes anomalies $\mathbb{S} = \mathbb{S}_0 \cup \mathbb{S}_x$.

Proposition 3: Assume the decomposition of RTW into RM+WM causing the set $\mathbb{S}_0$ of anomalies, that of RM into RCs causing $\mathbb{S}_x$, and that of WM into WCs causing $\mathbb{S}_y$, then

Table I
ANOMALIES AND ANOMALY SETS

Anomaly Sets	Anomalies
S_{RA}	{A3-rm1, A5A-1-rm1, A5A-2-rm1}
S_{RC}	{A3-rm1, A5A-1-rm1}
S_{WA}	{A4-wm2,A5A-1-wm2,A5A-2-wm2,A6-1-wm0 A6-2-wm0,A6-3-wm0,A6-4-wm0}
S_{WC}	{A4-wm2, A5A-2-wm2, A6-1-wm0,A6-4-wm0}
S_{RCWC}	$S_{RMWM} \cup S_{RC} \cup S_{WC}$
S_{RCWM}	$S_{RMWM} \cup S_{RC}$
S_{RMWC}	$S_{RMWM} \cup S_{WC}$
S_{RMWM}	{A4-rmwm2, A5B-rmwm0, A6-rmwm0}

the decomposition of RTW into RM+WC causes anomalies $\mathbb{S} = \mathbb{S}_0 \cup \mathbb{S}_x \cup \mathbb{S}_y$.

Following from Proposition 1, 2, and 3, we can deduce that the decomposition of RTW into RC+WC results in $S_{RCWC} = S_{RMWM} \cup S_{RC} \cup S_{WC}$, that of RTW into RC+WM results in $S_{RCWM} = S_{RMWM} \cup S_{RC}$, and that of RTW into RM+WC results in $S_{RMWC} = S_{RMWM} \cup S_{WC}$.

C. Cross-Shard Transactions

Read-Only-with-Commit & Write-Only-with-Commit Transactions. ROC can be decomposed in three ways. The possible anomalies for ROC decompositions are the same as decomposing RM into RCs, i.e. A3-rm1 and A5A-1-rm1, though the scopes of inconsistent data differ. For the RM decomposition of ROC, these anomalies can only be caused by concurrent cross-shard transactions, i.e. WOC and RCW, while operations on the RC-decomposed partitions can also conflict with intra-shard transactions within the same partitions. Anyhow, the anomaly set and the conflicting transaction types are the same as the RC decomposition of RM. The above analysis can be similarly carried out for WOC.

Read-Check-Write Transaction. The decomposition of ROC+WOC is the closest to the original serial execution of RCW due to the assumption of the two-phase execution. This decomposition leads to a similar conflicting scenario with that of decomposing RTW into RM+WM. Thus, the possible anomaly set is the same as the RM+WM decomposition of RTW. Conflicts can not only occur between the decomposed RCW transactions and cross-shard transactions, but also between the decomposed RCW transactions and intra-shard transactions due to the existence of anomaly A4-rmwm2.

Furthermore, we can decompose ROC and WOC respectively. With the assumption of the two-phase execution, an anomaly analysis similar to that of RTW's RM+WM decomposition can be made. We call it *sub-decomposition analysis*. Each sub-decomposition of ROC or WOC can lead to an anomaly set being the union of anomaly sets for the ROC+WOC decomposition and the sub-decomposition.

RCW has $16 = 4 \times 4$ decomposition plans, in which 4 ROC decompositions combines with 4 WOC decompositions. We cannot decompose RCW into multiple RTWs, due to the two-phase assumption of transaction processing. However, we can replace WM/RM with RTW in decompositions to get new decomposition schemes. In this situation, RTW is in fact degraded into WM/RM transactions. Such decompositions do not cause new anomalies. Thus, we do not consider decomposing RCW into any forms of RTW transactions. Table I and II summarize all decompositions, the resulting anomalies and the anomaly sets.

Table II
WEAK ISOLATION LEVELS

Seq	Anomaly Set/ Isolation Level	Decomposition
I	S_{RA}	RM→RA
II	S_{WA}	WM→WA
III	S_{RC}	RM→RC; ROC→RC
		ROC→RC+RM
		ROC→RM
IV	S_{WC}	WM→WC ; WOC→WC
		WOC→WC+WM
		WOC→WM
V	S_{RCWC}	RTW→RC+WC; RCW→RC+WC; RCW→RC+WM; RCW→RM+WM RCW→RC+WC+WM; RCW→RM+WC; RCW→RC+RM+WC;RCW→RM+WC+WM; RCW→RC+RM+WM;RCW→RC+RM+WC+WM
VI	S_{RCWM}	RTW→RC+WM; RCW→RC+WOC;
		RCW→RC+RM+WOC; RCW→RM+WOC
VII	S_{RMWC}	RTW→RM+WC; RCW→ROC+WC
		RCW→ROC+WM+WC;RCW→ROC+WM
VIII	S_{RMWM}	RTW→RM+WM
		RCW→ROC+WOC

Table III
ISOLATION LEVELS AND ANOMALY EFFECTS

Isolation Levels	Effects: phantom	read skew	lost update	write skew	inconsistent write	Anomaly #
S_{RC}	√	√				2
S_{RA}	√	√				3
S_{RMWM}			√	√	√	3
S_{WC}		√	√		√	4
S_{WA}		√	√		√	7
S_{RMWC}		√	√	√	√	7
S_{RCWM}	√	√	√	√	√	5
S_{RCWC}	√	√	√	√	√	9

IV. DEFINING ISOLATION LEVELS

Summarizing the analysis in the previous section, only 13 anomalies can happen, including A3-rm1, A4-wm2, A4-rmwm2, A5A-1-rm1, A5A-2-rm1, A5A-1-wm2, A5A-2-wm2, A5B-rmwm0, A6-1-wm0, A6-2-wm0, A6-3-wm0, A6-4-wm0, A6-rmwm0. The 13 anomalies can be grouped into eight sets. Exploiting the definition methodology of the classic work [17], we can accordingly **define eight isolation levels** by these eight anomaly sets. The summarized isolation levels are listed in Table II. While the four ANSI isolation levels can be totally ordered by the degree of consistency [21], these eight isolation levels do not have a total order.

Anomalies in the eight sets can be categorized into five effects, i.e. *phantom, read skew, lost update, write skew*, and *inconsistent write*. Multiple execution sequences with conflicting relations can contribute to the same effect. For example, A4-wm2 and A4-rmwm2 can both cause the lost

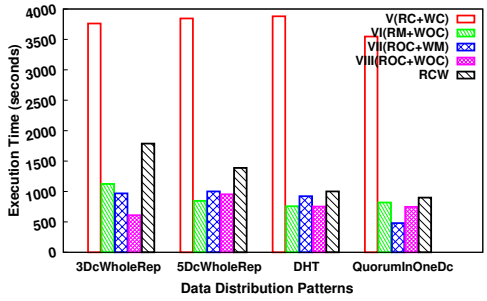

Figure 2. Actual execution time to get all transactions processed.

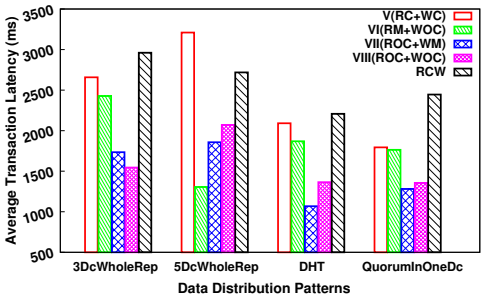

Figure 3. Average transaction latency.

update effect. In Table III, we illustrate the effects caused by each isolation level and list the number of possible anomalies under each isolation level. All anomaly sets in Table III are sorted first on the number of effects and then on the number of member anomalies.

The indication of anomaly numbers is that, assuming all execution sequences of transactions generated with the same possibility, **the more anomalies for an isolation level, the higher possibility of inconsistencies**. More anomalies lead to more abnormal execution sequences, taking up a larger portion of the whole execution sequence space and leading to a higher possibility of conflicts.

V. Experiments

In this section, we seek to find out how the weak isolation levels can improve transaction processing performance.

A. Experimental Setup

We use an eight-node cluster in the experiments. Each node is equipped with 8 cores, 60GB memory and 1Gbps network. Using software configurations, we emulate a multi-datacenter scenario, in which three or five datacenters may exist. Datacenter RTTs are randomly chosen from 20 to 200 times of the intra-datacenter RTT, which takes about 0.5 milliseconds. In each datacenter, 20 nodes are initiated. A server and a client run on each node. Data shards are distributed to nodes. Each shard has three indistinguishable replicas. We exploit the implementation of MDCC [7] for highly available datastores in the experiment.

To make thorough evaluation, we vary datastore deployment scenarios, changing the replica number, the replica distribution, and the replica location. We consider the following cases: (1) placing a quorum of replicas in one of five datacenters (QuorumInOneDc); (2) uniformly distributing data to nodes accross five datacenters (DHT); (3) placing a complete copy of data in each of three datacenters (3DcWholeRep); (4) placing a complete copy of data in each of five datacenters (5DcWholeRep); (5) accessing data following Zipfian distribution under DHT condition (ZipfianAccess), in comparison to the uniform accessing pattern of the previous four cases.

In each experiment, we execute 10 million operations, of which reads and writes take up 50% respectively. An experiment ends when all operations are executed. We test isolation levels V-VIII by grouping operations into transactions of RM+WM, RM+WOC, ROC+WM, and ROC+WOC. We executes one grouping in each experiment. RCW transactions with the serializability guarantee is executed for comparison. We test isolation levels III and IV, by comparing the executions of transactions RM vs. RC, WM vs. WC, and RM+WM vs. RC+WC respectively. Isolation levels I and II are not tested because RA and WA are seldom used.

B. Results and Discussion

Figure 2 demonstrates the execution time for four cases of 3DcWholeRep, 5DcWholeRep, DHT and QuorumInOneDc; and Figure 3 shows the average transaction processing times. The experimental results disagree with our previous understandings of weak isolation levels. **The weakest isolation level V that has the most anomalies demonstrates the worst throughput, as well as a high average transaction processing time.** The reason behind this phenomenon is that the RC+WC experiment issues much more transactions than others; and even though the granularity of a transaction is small, the costly coordination among multiple distributed data copies is inevitable.

Unexpectedly, **the isolation level VIII with the ROC+WOC decomposition has almost the best performance in all deployment scenarios. The isolation levels VI and VII also demonstrate better performance than the original serializable execution of transactions.** The reason is as follows. On the one hand, these decompositions do not lead to a great increase of transaction numbers as RC+WC does. On the other hand, they relax the strictness of coordination by narrowing the consensus scope and weakening the consensus requirement.

We do not plot the results for the ZipfianAccess case. The ZipfianAccess case requires an extremely long time for execution, e.g. 252178, 98853, and 124402 seconds for isolation levels V, VIII and serializability respectively. The durations are thousands of times of those in other cases. Therefore, *when there are lots of contentions as in the ZipfianAccess case, even the weak isolation levels cannot help*. This is due to the distribution of the multiple data replicas, the consistency of which requires many coordinations and retries, thus impairing performance. Similar results are also demonstrated in the average transaction

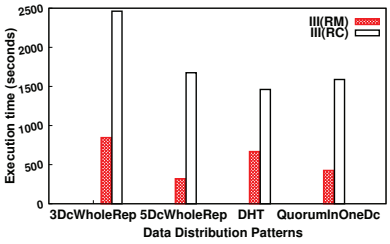

Figure 4. Execution time under isolation level III: RM vs. RC.

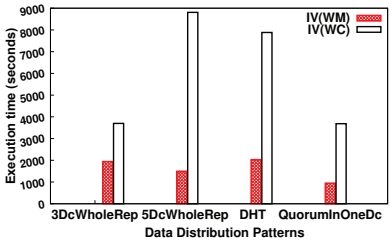

Figure 5. Execution time under isolation level IV: WM vs. WC .

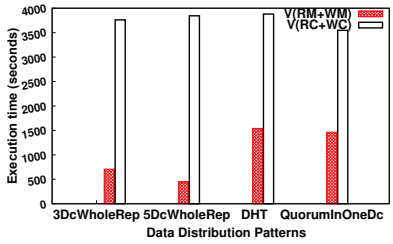

Figure 6. Execution time under isolation level V: RM+WM vs. RC+WC.

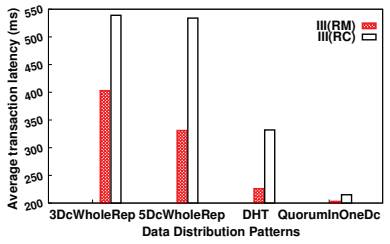

Figure 7. Average transaction latency under isolation level III: RM vs. RC.

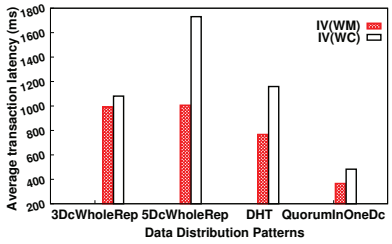

Figure 8. Average transaction latency under isolation level IV: WM vs. WC.

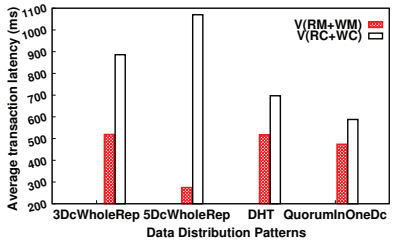

Figure 9. Average transaction latency under isolation level V: RM+WM vs. RC+WC.

processing time. With ZipfianAccess, only read-any and write-any transactions can help by guaranteeing only the eventual consistency, although *isolation VIII still has a better performance than the others even under the ZipfianAccess deployment*.

For isolation levels III and IV, we compare the performances of decompositions RM vs. RC, WM vs. WC, and RM+WM vs. RC+WC. The experimental results are shown from Figure 4 to Figure 9. Even under the same isolation level, different implementations lead to different performances. As shown in the figures, *executing multiple operations on the same shard in one transaction leads to a much better performance than executing each operation in a separate transaction*. This fact indicates that, besides decomposing a transaction, a client of highly available datastores can also improve performance by issuing multiple operations in one transaction.

Finally, we must point out that the benefit of consensus reduction can outweigh the overhead in issuing and coordinating multiple transactions, because the original transactions can have long commit durations or high abort rates due to conflicting transaction operations, while the multiple decomposed transactions can be executed concurrently and independently.

VI. Related Work

To meet applications' requirements on high availability, eventual consistency was once widely adopted [12]. Later proposals for guaranteeing causal consistency [22], [23] and strong consistency [24], [25] emerge. As transaction continues to be recognized as valuable [26], transactional support was then implemented in highly available databases [11], [9], [7], [8]. Transaction processing performance can be improved by using weaker isolation levels [16], [21]. A later work generalizes classic isolation levels by using conflicting operation relations and transaction dependency relations [18].

A recent work proposes the definition of highly available transactions [27], which support weaker consistency. The definition does not consider that a single accessible replica of a data item might not reflect all operations on the data item due to failures, and that the system cannot establish correct database states when server nodes recover after failures [28]. The two aspects are key to high availability under failures. A recent work [29] introduces a method to analyze whether coordination is needed for applications of scalable databases. In comparison, our work considers how the strictness of coordination can be relaxed for performance improvement when coordination is demanded.

Our way of decomposing a transaction into multiple transactions for execution leads to a similar execution structure to that of nested transactions [30]. However, analyses on nested transactions are mainly based on locking implementations, proposing no new isolation models. Besides, nested transactions can have hierarchical structure, while, in our proposal, transactions after consensus reduction consist of flat transactions. The work on transaction chopping [31] also breaks down a transaction into multiple smaller transactions for execution. But it mainly considers conditions for the serializability guarantee. It does not consider the high availability requirement, thus not directly applicable to transactions in highly available databases.

VII. Conclusion

In this paper, we propose to improve transaction processing performance in highly available datastores through consensus reduction. We answer three related questions: (1) whether consensus reduction can help improve transaction processing performance; (2) what isolation levels are formed in consensus reduction; (3) how weaker isolation levels

improve performance. We draw three conclusions: (1) consensus reduction can help improve transaction processing performance; (2) eight new isolation levels are formed in consensus reduction; (3) only isolation levels VI, VII and VI-II among the eight isolation levels can improve performance. While isolation level VIII has the best performance under varied deployment scenarios, the weakest isolation level V has the worst performance.

Users can use an anomaly-set table like Table 1 in the paper to check which anomaly sets their applications can tolerate. As each anomaly set corresponds to an isolation level, they can thus find out which isolation levels are feasible for their applications. Among these isolation levels, they can then choose the isolation level with the best performance for their implementations.

Acknowledgment. This work is supported in part by the State Key Development Program for Basic Research of China (Grant No. 2014CB340402) and the National Natural Science Foundation of China (Grant No. 61303054 and No. 61432006).

REFERENCES

[1] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications." in *OSDI'10*, pp. 1–15.

[2] 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. of 24th SOSP'13*, pp. 276–291.

[3] H. V. Jagadish, I. S. Mumick, and A. Silberschatz, "View maintenance issues for the chronicle data model (extended abstract)," in *Proc. of PODS'95*, pp. 113–124.

[4] M. Stonebraker, "In search of database consistency," *Commun. ACM*, vol. 53, no. 10, pp. 8–9, Oct. 2010.

[5] "Reddit, quora, foursquare, hootsuite go down due to amazon ec2 cloud service troubles," September 2015, http://www.huffingtonpost.com/2011/04/21/amazon-ec2-takes-down-reddit-quora-foursquare-hootsuite_n_851964.html.

[6] "Lessons netflix learned from the aws outage," http://techblog.netflix.com/2011/04/lessons-netflix-learned-from-aws-outage.html.

[7] T. Kraska, G. Pang, M. J. Franklin, and S. Madden, "Mdcc: Multi-data center consistency," in *Eurosys'13*, pp. 113–126.

[8] F. Nawab, V. Arora, D. Agrawal, and A. El Abbadi, "Minimizing commit latency of transactions in geo-replicated data stores," in *Proc. of SIGMOD'15*, pp. 1279–1294.

[9] P. A. Bernstein, S. Das, B. Ding, and M. Pilman, "Optimizing optimistic concurrency control for tree-structured, log-structured databases," in *Proc. of SIGMOD'15*, pp. 1295–1309.

[10] B. Kemme, "Database replication for clusters of workstations," Ph.D., EPFL, 2000.

[11] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Googles globally-distributed database," in *Proc. of OSDI*, 2012, pp. 251–264.

[12] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[13] M. T. Özsu and P. Valduriez, *Principles of distributed database systems*. Springer Science+ Business Media, 2011.

[14] S. Roy, L. Kot, G. Bender, B. Ding, H. Hojjat, C. Koch, N. Foster, and J. Gehrke, "The homeostasis protocol: Avoiding transaction coordination through program analysis," in *Proc. of SIGMOD'15*, pp. 1311–1326.

[15] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Coordination avoidance in database systems," in *Proc. of VLDB'14*, pp. 185–196.

[16] J. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger, "Granularity of locks and degrees of consistency in a shared data base," *Readings in Database System*, pp. 244–260, 1998.

[17] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ansi sql isolation levels," *ACM SIGMOD Record*, vol. 24, no. 2, pp. 1–10, 1995.

[18] A. Adya, "Weak consistency: A generalized theory and optimistic implementations for distributed transactions," Ph.D., MIT, 1999.

[19] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-wesley New York, 1987, vol. 370.

[20] A. Adya, B. Liskov, and P. O'Neil, "Generalized isolation level definitions," in *Proc. of ICDE'00*. IEEE, pp. 67–78.

[21] American National Standards Institute, *American national standard for information systems: database language — SQL: ANSI X3.135-1992*. pub-ANSI, 1992.

[22] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don't settle for eventual: scalable causal consistency for wide-area storage with cops," in *Proc. of SOSP'11*, pp. 401–416.

[23] P. Bailis, A. Ghodsi, J. Hellerstein, and I. Stoica, "Bolt-on causal consistency," in *Proc. of SIGMOD'13*, pp. 761–772.

[24] J. Rao, E. J. Shekita, and S. Tata, "Using paxos to build a scalable, consistent, and highly available datastore," in *Proc. of VLDB'11*, pp. 243–254.

[25] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Scalable consistency in scatter," in *Proc. of SOSP'11*. ACM, pp. 15–28.

[26] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the trade-offs in commercial cloud storages: the consumers perspective," in *Proc. of CIDR'11*, 2011.

[27] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Highly available transactions: Virtues and limitations," in *Proc. of VLDB'14*, pp. 181–192.

[28] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *Proc. of SOSP'07*, pp. 205–220.

[29] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Coordination-avoiding database systems," in *Proc. of VLDB'14*, pp. 185–196.

[30] T. Härder and K. Rothermel, "Concurrency control issues in nested transactions." Springer-Verlag New York, Inc., pp. 39–74.

[31] 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, Sep. 1995.