Optimizing Inter-data-center Large-Scale Database Parallel Replication with Workload-Driven Partitioning

Zhen Gao[3(✉)], Hong Min[1(✉)], Xiao Li[2], Jie Huang[3], Yi Jin[4], An Lei[3], Serge Bourbonnais[2], Miao Zheng[5], and Gene Fuh[5]

[1] IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
hongmin@us.ibm.com
[2] IBM Silicon Valley Lab, San Jose, CA, USA
{lixi,bourbon}@us.ibm.com
[3] School of Software Engineering, Tongji University, Shanghai, China
{gaozhen,huangjie,1434318}@tongji.edu.cn
[4] Pivotal Inc., Beijing, China
jinyi.smilodon@gmail.com
[5] IBM System and Technology Group, New York, USA
zhengm@cn.ibm.com, fuh@us.ibm.com

Abstract. Geographically distributed data centers are deployed for non-stop business operations by many enterprises. In case of disastrous events, ongoing workloads must be failed over from the current data center to another active one within just a few seconds to achieve continuous service availability. Software-based parallel database replication techniques are designed to meet very high throughput with near-real-time latency. Understanding workload characteristics is one of the key factors for improving replication performance. In this paper, we propose a workload-driven method to optimize database replication latency and minimize transaction splits with a minimum of parallel replication consistency groups. Our two-phased approach includes (1) a log-based mechanism for workload pattern discovery; (2) a history-based algorithm on pattern analysis, database partitioning and partition adjustment. The experimental results from a real banking batch workload and a benchmark OLTP workload demonstrate the effectiveness of the solution even for partitioning 1000 s of database tables in very large workloads. Finally, the algorithm to automate the cyclic flow of workload profile capturing and partitioning readjustment is developed and verified.

1 Introduction

Many enterprises employ multiple geographically distributed data centers running the same applications and having the same data to provide zero-downtime upgrades, cross-site workload balancing, continuous availability and disaster recovery. Across these centers, data replication is used to maintain multiple data copies in near real-time.

Y. Jin—Work done while employed by IBM.

A. Hameurlain et al. (Eds.): TLDKS XXIV, LNCS 9510, pp. 169–192, 2016.
DOI: 10.1007/978-3-662-49214-7_6

Various database replication techniques are proposed to serve different purposes. High availability (HA) within a single data center employs data replication to maintain global transaction consistency [3] or to improve fault tolerance and system performance via transaction processing localization [1, 16, 18]. Replication over unlimited distances leads to enormous challenges of scalability, efficiency and reliability, especially in an active-active deployment with heterogeneous database architectures.

Although DBMS built-in replication function has the potential for better performance via tighter software stack integration, middleware-based replication is more suitable for multi-vendor heterogeneous database environments [12]. Industrial examples of such technology include IBM Infosphere Data Replication [22], Oracle GoldenGate [23], etc. One widely used approach is to capture committed data changes from DBMS recovery log and to replicate the changes to target DBMS. Replicating data after changes committed at the source does not impact the response time of source-side applications. This paper addresses the performance problem of large-scale asynchronous database replication optimization for minimizing the data staleness and data loss in case of unrecoverable disasters.

Parallel replication is a desirable solution to increase the throughput by concurrently replicating changed data through multiple logical end-to-end replication channels. Such concurrent replication can potentially split a transaction's writeset among channels. Similar to DBMS snapshot consistency, point-in-time (PIT) snapshot consistency is provided via time-based coordination among replication channels [22]. PIT consistency is a guarantee of replicated data having a consistent view with the source view at an instance of past time. Such a time delay in PIT consistency is called PIT consistency latency. PIT consistency latency at the target DBMS is determined by both replication channel throughput and the duration between when the first element of a transaction's writeset is replicated and when the last element is replicated. It is not difficult to envision that higher replication throughput delivers lower PIT consistency latency. In addition, normally the more replication channels a transaction's writeset is split into, the longer it takes to reach PIT consistency. Over-provisioning with underutilized replication channels also introduces extra complexities and wastes resources.

This paper addresses the partitioning automation in parallel database replication cross data centers. Partitioning a database is a challenging task in PIT consistency latency reduction. By following design principles of DBMS data independence [2], databases and applications are often designed separately. Database access patterns usually differ by applications. One specific database partitioning scheme hardly suits the needs of other applications. Furthermore, new applications are continually deployed on existing databases and access patterns change as business requirements evolve. Taking into account of varying workload characteristics, scale of database objects and resource constraints, it is impractical for database administrators to have comprehensive understandings of all the database activities and to manually perform and adjust database partitioning for parallel data replications.

Our design aims at minimizing replication channels and achieving desired point-in-time consistency. With the observation that similar workload patterns re-occur in most business applications, we propose a two-stepped approach. In our approach, a log-based mechanism is employed for workload pattern discovery, and a history-based

algorithm is used for pattern analysis and database partitioning. The partition granularity is at DBMS object level such as tables and table partitions, which can reach up to thousands or tens of thousands in a large enterprise IT environment. Finer grained partitioning, such as at the row level, is less practical due to higher overhead in runtime replication coordination and DBMS contention resolution. Our approach discovers and analyzes the data access patterns from the DBMS recovery log, and makes partitioning recommendations using a proposed two-phased algorithm called Replication Partition Advisor (RPA)-algorithm. In the first phase, the algorithm finds a partitioning solution with the least replication channels such that the PIT consistency latency is below a threshold tied to a service-level agreement (SLA). The second phase refines the partitioning solution to minimize the number of transaction splits. Our approach is applicable to share-nothing, share-memory and share-disk databases [20]. The real-world workload evaluation and analysis demonstrate the effectiveness of our solution.

The rest of the paper is organized as follows. Section 2 introduces more background about inter-data-center parallel data replication. Section 3 describes the workload profile tool (WPT). Section 4 presents the RPA-algorithm and discusses how a real-time re-partitioning can be achieved. The experiment evaluations are presented in Sect. 5. We discuss related work in Sect. 6 and end the paper with the conclusion in Sect. 7.

2 Background on Parallel Data Replication

Based on the data change propagation, mainstream replication technologies can be classified into two major types: synchronous replication (also called eager replication) and asynchronous replication (also called lazy replication). In eager replication, the changes to all the copies are in a single transaction (unit of work). If a failure happens on any copy, the entire transaction will roll back. Compared with eager replication, lazy replication eliminates the impact on transaction response time by relaxing the strong consistency among copies. Instead of using two-phase commit protocols, lazy replication chooses an optimistic protocol: after the transactions are committed on the source database, the data changes are asynchronously captured, propagated and then applied to the target databases.

The fundamental difference between eager replication and lazy replication is the way to optimize the tradeoff between data consistency and system performance. Whenever increased transaction time is not tolerable (often the case for financial transactions), eager replication is not an option because of the propagation delay incurred over geographic distances. Each 100 km of fiber typically adds about 1 ms of delay [5]. Lazy replication does not impact transaction response time, but introduces two major issues:

(1) Data staleness: After a transaction has committed at the source database, the subsequent data access to the target databases might not return the updated values immediately and consistently. To measure the window of inconsistency between source and target copies, a point-in-time (PIT)-consistency latency is introduced

to describe how much time the target database is behind the source database. We further define consistency group (CG), a set of tables for which transaction consistency is always preserved. The PIT consistency is per CG.

(2) Data loss: An unrecoverable disaster (e.g., earthquake) on the source copy can cause a loss of the data changes that have not been applied to the target databases. PIT consistency latency also largely affects the data loss window in lazy replication. RPO (Recovery Point Objective) in Lazy replication is a non-zero unless there are other means to compensate. Data loss is a function of the replication delay.

To alleviate the impacts of data staleness and data loss, it is highly desirable to reduce the PIT consistency latency especially with the ever-growing data volumes. In the wide area network replication, the PIT consistency latency can reach a non-tolerable value with respect to SLA, during a heavy workload period, particularly batch processes that might update each row of an entire database. To reduce the PIT consistency latency, replication protocols might divide the database objects among several replication channels; potentially have to relax the ACID compliant transaction integrity. At the same time, through synchronization across parallel replication transmission or replay, the eventual data consistency can be ensured at target databases even after disaster recovery (also called 100 % recovery consistency objective). Once all channels have caught up to the same point, consistency is guaranteed. The major benefit is to increase the overall replication throughput through concurrency. Although the negative effects are anomalies (e.g., dirty read) during the replication, most read-only applications can use the data as long as it is not stale beyond a certain threshold, and/or can retry if data has not yet arrived. In these applications, data staleness is more significant than temporary data anomalies.

Our work is applicable to an active-active WAN configuration (where transactions can be executed at either site) presuming that proper transaction routing provides conflict prevention. For discussion simplicity, we present uni-directional replication in an active-query configuration (a.k.a. master-slave [9]) where update transactions are restricted to a designated master copy in one data center and read-only transactions are executed in other data center copies. Upon a failure on the active copy caused by disasters, one of the query copies assumes the master role and takes over the updates.

Figure 1 illustrates a logical architecture of typical parallel lazy data replication between two database systems that potentially reside in two data centers. A parallel replication system can be modeled as a network $G(C \cup A, E)$ with a set of capture $C = \{c_1, c_2, ..., c_s\}$ and a set of apply $A = \{a_1, a_2, ..., a_r\}$. To replicate data changes, a capture agent, such as Capture1 in Fig. 1, captures the committed data changes from the database recovery logs at the source site, packs and sends them over a transport channel. The transport channels manage reliable data transfer between the two sites. An apply agent, such as Apply1 in Fig. 1, applies the changes to the target database. Each capture and apply agent can be attached to a different database node in a cluster. Within the capture and apply agent, whenever possible, multiple threads are used to handle the work with the protection of causal ordering.

The link $(a, c) \in E$ represents a *logical replication channel*, which is an end-to-end replication data path from a log change capture at the source site to a change apply at

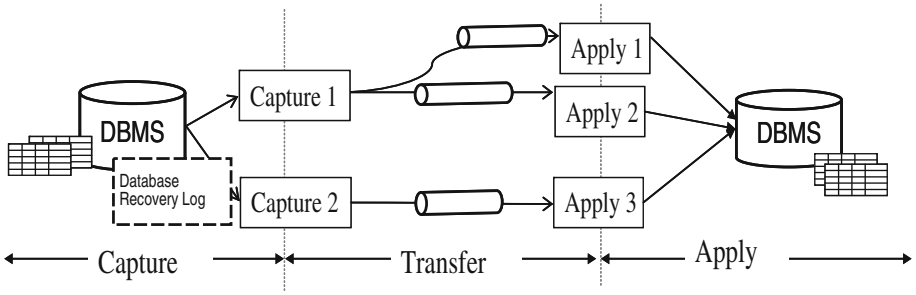

Fig. 1. Logical architecture of parallel lazy replication

the target site. Three channels are shown in Fig. 1. A throughput capacity or bandwidth *BW(a, c)* measures the maximum data throughput, in bytes/second, of a channel. The value is affected by all the involved components, e.g., source log reader, capture, network, apply, target database, etc. For simplicity, this paper assumes that the effective "bandwidth" is static. All changes within each database object (tables or partitions) are replicated by one channel and this is designated by a preconfigured subscription policy. Each capture agent only captures the changes from its subscribed objects. Transported data changes at target site are subscribed by one or more apply agents on mutually disjoint sets of objects. The entire set of database objects within each replication channel is guaranteed to preserve serial transaction consistency. Hence, the set of database objects $TB = \{tb_1, tb_2, ..., tb_k\}$ that are replicated within the same replication channel *(a, c)* is called a *consistency group* denoted as *cg(a, c, TB)*.

When a transaction's writeset is split into different consistency groups, the transaction is split into multiple partial transactions with the same source-side commit time. Each partial transaction is replayed at the target as an independent transaction. Replication then operates with eventual consistency: i.e., transaction consistency is guaranteed only when all table changes are replicated up to a common point-in-time. Eventual transaction consistency is suitable for a large number of read-only applications that can use the data as long as it is not stale beyond a certain threshold. Eventual consistency must be restored before write applications can be switched in case of planned site switch or unplanned disaster. Like IBM IIDR Q-Replication [22], replication across consistency groups can be synchronized so that data is not applied to the target DBMS unless it has been received into persistent storage at the target for all consistency groups. Thus, in case of disaster, consistency can be restored by draining the queues for all consistency groups up to a point that is consistent across all queues. When eventual consistency is not acceptable, the target DBMS still can restore the point-in-time consistency using the source-side commit time. Normally, such a consistency recovery mechanism is tightly integrated with the multi-version concurrency control.

In the next two sections, we discuss our solution for optimizing the partitioning of database objects into minimum number of consistency groups to achieve a PIT consistency latency goal based on our workload profiling approach.

3 Workload Abstraction

3.1 Transaction Pattern and Workload Profile

This section details workload abstraction in our proposed partitioning solution for parallel data replication. Considering the entire workload cannot be recorded, we employ a landmark window model for incrementally summarizing the data activity statistics of each table in the workload and the coupling relationships among tables implied by transaction commit scopes.

The term "transaction pattern" is introduced to define a profiled entity that contains statistical and transactional information in a workload. A number of transactions belong to the same transaction pattern if and only if they update exactly the same set of tables, regardless of the specifics such as update sequence, data volume or operation types (insert, update or delete). For instance, given a table set T = {A, B, C, D}, examples of possible transaction patterns are P{A, B, C}, P{A}, P{B, C, D}, P{A, B}, etc. P{A, B, C} and P{A, B} are not considered as the same transaction pattern even though one has a subset of tables of the other. These patterns reflect the hidden table relations recorded in transaction commit scopes. Separate collection of pattern statistics of P{A, B, C} and P{A, B} facilitates the partitioning algorithm in the evaluation of transaction splits. For horizontally partitioned tables, each partition is regarded as an individual physical object. In WPT, the definition of transaction pattern can be easily extended to treat different partitions as different table objects, especially when most of industrial DBMS logs contain the partition identifier. For simplicity, the rest of the paper only discusses tables.

To measure pattern-specific workload size, we record the table-specific statistics, including the numbers of insert, update and delete operations, and data change volumes measured by bytes. For an insert operation, all column values of the new row need to be replicated to the target site. For an update operation, in addition to the values of all the updated columns, the old key values should also be replicated to target sides for row lookup and collision detection. For a delete operation, only the key values need to be replicated. Thus, the workload size depends on the operation types, the actual log contents with the column values, the table definition and the actual column value size. Captured from DBMS catalogs, table schema definitions are used to decode the column values of each row for computing accurate data changes of each operation.

The collected workload patterns and their statistics are modeled by landmark window model. The entire workload is chunked to disjoint pattern snapshots by user-specified time intervals. The collection of all the snapshots constitutes a workload profile.

Figure 2 shows an example transaction pattern entity in a JSON format. It includes information such as snapshot time, transaction pattern identifier (ID), transaction count, the number of tables included, identifiers of the tables, as well as insert, update and delete volumes in bytes. Table-level statistics in a particular transaction pattern include the following content showed in Fig. 2: database ID and table ID; total count of the IUD (Insert, Update, Delete) operations; total bytes of the data that are replicated of the IUD operations; and total bytes of the raw log data in the IUD operations etc.

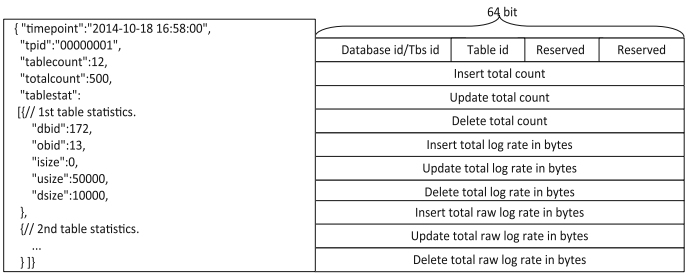

Fig. 2. Example WPT data in JSON format

3.2 Workload Profiling Tool

We implemented a workload profiling tool (WPT) to execute alongside with a commercial database system. The log transaction reader uses the database recovery log APIs to scan through the transaction history log. A transaction pattern control block (TPCB), which stores information of a transaction pattern, is stored in a hash table in an in-memory buffer, as shown in Fig. 3. For every transaction record gathered in log scanning, the statistical information is accumulated if the matching transaction pattern exists in the hash table. If no matching transaction pattern is found, a new transaction pattern entry is added to the hash table. In Fig. 3, TPCB manager maintains a linked list of TPCB buffer. Currently, each buffer has a 64 MB space. When one buffer is full, a new buffer is created and inserted into the list. Using a hash table to store TPCBs ensures that a TPCB entity can be quickly found from these buffers and updated. The address of each TPCB entity is saved in the hash table as a hash value, while the hash key is the hash code of TPCB ID of one TPCB entity. After completing all log records within a snapshot interval, WPT dumps all the gathered information from memory to disk files and then feeds the files to RPA.

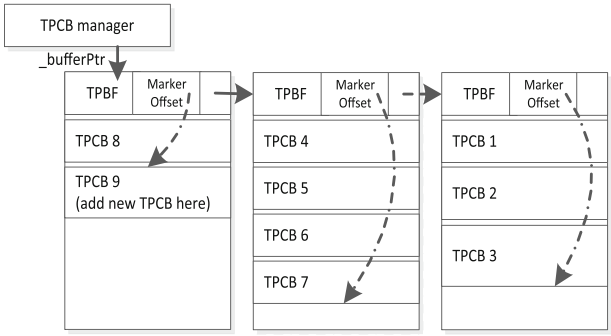

Fig. 3. In-memory TPCB hash table

Since WPT executes in a separate address space from the DBMS, it does not disturb normal database workloads. From a system resource sharing prospective, one can run

WPT at a lower job priority than regular application workloads to avoid or reduce resource contentions. An alternative is to run WPT offline. For example, if the production system disk that contains the database log is mirror-copied to a different system at a local or remote site, WPT can process the log files from the mirrored disk. Multiple WPT instances can also execute concurrently to process log records from different time periods.

WPT and RPA are used in the initial configuration of parallel replication as well as can be applied in the subsequent tuning and optimization process. Figure 4 shows the process of how users can iterate through the capturing, profiling, analysis and database partitioning steps, along with workload growths or new application deployments, to adjust replication groups. RPA supports iterative tuning to help users continuously optimize their partitioning solution. It is up to each replication software whether the redeployment of replication partitioning can be done online or offline.

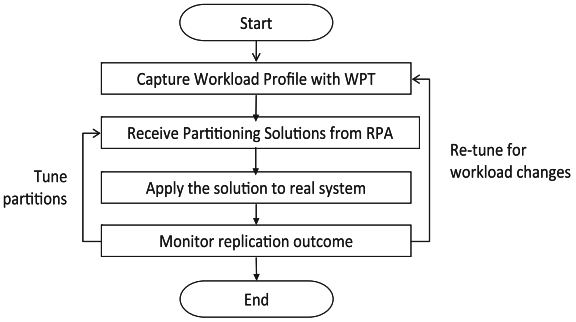

Fig. 4. Iterative parallel replication tuning

4 Replication Partition Advisor Algorithm

4.1 Problem Formulation

Let *WK(TB, TX, T, IUD)* denote a replication-specific workload collected during a time window $T = \{t_0, t_0 + dt, t_0 + 2 \cdot dt, \ldots t_0 + v \cdot dt\}$, where dt is the sample collection interval; $TB = \{tb_1, tb_2, \ldots, tb_n\}$ is a set of n replication objects (e.g., tables) whose changes are to be replicated and $TX = \{tx_1, tx_2, \ldots, tx_k\}$ represents their transaction activities; and *IUD(TB, T)* is the time series statistics of inserts, updates and deletes on the tables in a time window T. Given a parallel replication system $G(C \cup A, E)$, RPA-algorithm partitions all the replicated database objects TB to form a set of m mutually disjoint non-empty partitions $CG = \{cg_1, cg_2, \ldots, cg_m\}$, where cg_i is a consistency group replicated by a particular channel $E(a, c)$. The objective is to find a solution such that m is minimal and the worst replication latency in CG is below a user-supplied threshold H.

For a particular replication channel, the PIT consistency latency at a specific time point t_p is the difference between t_p and the source commit time for which all transactions to that point have been applied to the target at time t_p. The latency of each

channel is directly related to the logical replication throughput capacity *BW(a, c)* as well as the size of workload assigned to this channel. The workload size is defined as the number of replicated data bytes. For a specific channel, it can process at most $dt \cdot BW$ bytes within dt seconds. The residual workload will be delayed to the next intervals. Residual workload $RES_{cg,i}$ for a consistency group cg at time $t_0 + i \cdot dt$ is the remaining work accumulated at $t_0 + i \cdot dt$ that has not been consumed by cg_i. Thus, $RES_{cg,i}$ can be computed iteratively by:

$$RES_{cg,i} = max\{(RES_{cg,i-1} + \sum IUD(TB_{cg}, t_0 + (i-1) \cdot dt) - dt \cdot BW), 0\} \quad (1)$$

Assuming data is consumed on a first-in-first-out basis, the PIT consistency latency for cg at time $t_0 + i \cdot dt$ is the time to process the accumulated residue and new activities at $t_0 + i \cdot dt$:

$$PIT_{cg,i} = \left(RES_{cg,i} + \sum IUD\left(TB_{cg}, t = t_0 + i\right)\right)/(dt \cdot BW) \quad (2)$$

The maximum PIT consistency latency PIT_{cg} of group cg during the time period is computed as:

$$PIT_{cg} = max\left\{PIT_{cg,i} | i = 0, 1, 2, \ldots, v\right\} \quad (3)$$

The maximum PIT consistency latency $PIT_{CG\text{-}max}$ of a set of consistency groups CG is the highest value of PIT_{cg} among all consistency groups in CG.

The objectives of the partitioning optimization can be formulized as follows. Given a workload W, a parallel replication system G and its replication channel bandwidth *BW(a, c),* and an SLA-driven PIT consistency latency threshold H, the first objective function is defined as:

$$L = \min\{|CG||\forall CG : PIT_{CG-\max} \leq H\}, \quad (O1)$$

where $|CG|$ is the size of a consistency group set CG, i.e., the number of groups in the set. O1 is to find the partitioning solutions with the lowest number L of consistency groups such that the highest PIT consistency latency of all the replication channels $PIT_{CG\text{-}max}$ is less than or equal to H. Let P_L represent all the partition solutions of group size L and satisfy O1. The second objective is to find a partitioning solution with the minimized number of transaction splits.

$$T\_split = \arg\min_{CG \in P_L} \left\{ \sum_{tx \in TX} \sum_{i=1}^{L} tr^T(cg_i, tx) | tr^T(cg_i, tx) \in \{0, 1\} \right\}, \quad (O2)$$

where $tr^T(cg_i, tx)$ is either 1 or 0 representing whether transaction tx has tables assigned to group cg_i or not. When all the tables in transaction tx are assigned to a single group, $tr^T(cg_i, tx)$ equals 0 for all groups except one. O2 seeks to find the partition solution in P_L such that the aggregated count is minimized. When no transaction split is required, *T_split* equals the total number of transaction instances in the workload.

4.2 RPA-Algorithm Phase-1: Satisfying PIT Consistency Latency with the Least Groups

Our RPA algorithm consists of two phases: phase-1 is to find a solution that satisfies the first objective O1, and then phase-2 applies a transaction graph refinement approach to achieve the objective O2. The algorithm flow of phase-1 is listed below followed by a description.

RPA-algorithm Phase-1 Steps

1_1. Aggregate the total amount of work $W_{sum} = \sum IUD(TB,T)$ *for all tables in* $TB=\{tb_1, tb_2, ..., tb_n\}$ *and in time period* $T=\{t_0, t_0+dt, t_0+2\cdot dt, ... t_0+v\cdot dt\}$.

1_2. Compute the lower bound L_{lower} *of the number of consistency groups* $L_{lower} = W_{sum}/(BW\cdot v\cdot dt + BW\cdot H)$. *Set initial CG number* $L = L_{lower}$.

1_3. Sort all tables in TB in descending order by each table's peak activity max(IUD) and total activity $\sum$ *(IUD), represented by* TB_P *and* TB_T *respectively.*

1_4. Select a subset TB_{top} *consisting of top tables from both list* TB_P *and list* TB_T

1_5. Exhaust all the combinations of placing TB_{top} *tables into L groups. Select the placement with the lowest maximum PIT consistency latency and continue to next step.*

1_6. Iterate through the rest tables in their descending order in TB_p. *Test each table against each consistency group and compute potential maximum PIT latencies* PIT_{pmax_i}, $i = 1,2...L$, *for each group. Place the table in the group with the lowest potential* PIT_{pmax}. *If* $PIT_{pmax} > H$ *for the selected group, stop and go to 1_8.*

1_7. Compute the maximum PIT_{max_i}, $i = 1,2...L$ *for all consistency groups.*

1_8. For all consistency groups. If $max(PIT_{max\_i}) > H$ *for* $i = 1,2...L$, *increment* $L=L+1$ *and repeat steps 1_5 to 1_8 until* $max(PIT_{max}) \leq H$. *The last L is the minimum number* L_{min} *of consistency groups.*

Given bandwidth *BW(a, c)* and a user-specified PIT consistency latency threshold H, the first two steps in phase-1 obtain the lower bound L_{lower}, for the number of consistency groups. The lower bound describes the best case scenario: the workload volume distributes uniformly in both table and time dimensions, while the PIT consistency latency reaches the highest at the end of the time window $t_0 + v \cdot dt$ and the residual workload evenly spreads among all channels, i.e. $W_{sum} = L_{lower} \cdot BW \cdot (v \cdot dt + H)$. Starting with this lower bound L_{lower}, the process in steps *1_3* to *1_6* partitions the tables into L_{lower} groups. We then re-examine the actual maximum PIT consistency latency of all groups in steps *1_7* and *1_8*. If the latency is higher than the threshold H, another round of partitioning is performed with the number of groups incremented by 1.

For each fixed group number, the problem becomes to partition n tables into L consistency groups for PIT consistency latency minimization, which is an NP-hard problem [7]. Given that the number of tables in a workload can reach thousands or even more, it is not realistic to exhaust all the partitioning combinations for finding the best among them. Instead, the greedy algorithm is introduced to resolve such a problem [8].

When applying the greedy algorithm, we use a two-step approach for improving the possibility of finding a *global optimal* solution instead of a local optimum. First, using the most active tables TB_{top} (selected in step *1_4*), step *1_5* enumerates all the possibilities of partitioning them into *L non*-empty groups. The number of combinations for such a placement grows rapidly with the numbers of tables and groups. For avoiding an impractically high cost of step *1_5*, the size of TB_{top} is determined based on a reasonable computation time on the system where RPA runs. The best choice from the exhaustive list of placements is the one with the lowest maximum PIT consistency latency. Step *1_5* is then followed by a greedy procedure in step *1_6* that tests each of the rest tables against each consistency group and computes the group's potential new maximum PIT consistency latency contributed by the table. The group with the lowest new maximum PIT consistency latency is the target group for the table placement. The greedy iteration in step *1_6* uses a stronger heuristics for reaching the minimum number of consistency groups, even though it is possible that other partitioning schemes that satisfy objective *O1* (Sect. 4.1) also exist. An added benefit is that this heuristics tends to generate consistency groups with less PIT consistency latency skews among them.

Our approach is particularly effective when there are activity skews among the tables. In fact, such skews are common in real-world applications. Figure 5 shows a customer workload analysis on how tables weight within the workload with respect to total and peak throughputs. A table with a higher *x-axis* value weights more in terms of total throughput than those with lower *x-axis* values. Such a table contributes more to the overall workload volume accumulation and channel saturation. A table with a higher *y-axis* value is more likely to contribute to higher PIT consistency latency at its own peak time. As shown in Fig. 5, tables with higher peak or total throughputs constitute a small fraction in the entire workload. Based on this observation, *step 1_4* selects the top tables with higher total and peak throughputs for enumerative placement tests.

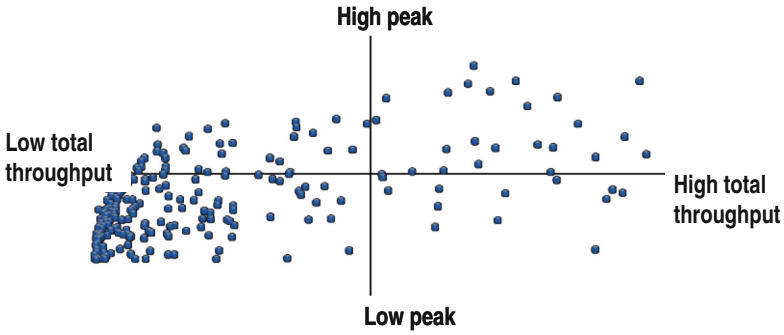

Fig. 5. Table activity distribution in a real-world banking application workload

Throughput-Balancing: An Alternative to PIT Consistency Latency Minimization. Calculation of PIT consistency latencies is impossible when quantified replication bandwidth is unavailable. In this case, the optimization goal of the RPA-algorithm

is adjusted to balance the peak volume and total volume given a targeted number of consistency groups. Instead of computing PIT consistency latency, steps *1_5* and *1_6* choose the candidate group based on the accumulated peak volume and total volume after adding a new table. Both factors are positive correlated with the PIT consistency latency. Total throughput-based placement tries to balance utilizations of physical replication channels. Peak throughput-based placement is for capping the highest workload volume among all channels. Understanding workload peaks also facilitates capacity planning and system configuration. This alternative is referred to as the throughput-balancing algorithm (RPA-T-algorithm).

4.3 Transaction Split Reduction

RPA-algorithm phase-1 focuses on reducing PIT consistency latency. This section describes phase-2, which attempts to reduce transaction splits for statistically increasing serial consistency in data replication.

Transaction Graphs. In RPA, we use an undirected weighted graph *TG(TB, TX, T, IUD)* to model tables and their transaction relationships within a workload *WK(TB, TX, T, IUD)*. Each node in the graph represents a table in *TB*. For simplicity, the same notation $TB = \{tb_1, tb_2, ..., tb_n\}$ is also used to represents the graph nodes. The weight of a node is the time series IUD statistics for the table in the workload profile. An edge $e(tb_i, tb_j)$ connecting two nodes tb_i and tb_j denotes that there exists one or more transaction patterns that correlate both tables. The weight of the edge $|e(tb_i, tb_j)|$ is the total transaction instance counts from all the transaction patterns that involve both tables. Figure 6 illustrates an example of a transaction graph with 19 tables. Table *T1*'s weight is associated with a time series statistics *{234, 21, 654, 2556, ..}, which* indicates data activities to be replicated at each time point for *T1*. The weight 731 of edge *e (T1, T2)* means that there are 731 committed transactions involving both *T1* and *T2*.

Because of relational constraints or other reasons, there are cases when transaction consistency must be preserved among certain tables. That means, these correlated tables need to be assigned to the same consistency group. When the RPA-algorithm

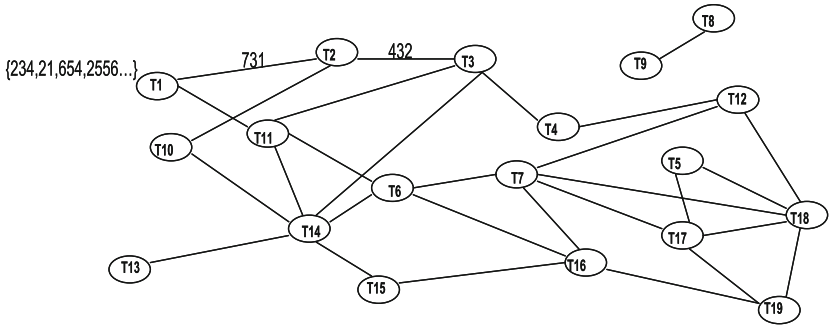

Fig. 6. Transaction graph

builds a transaction graph for a workload, each set of such correlated tables is first merged into a single node with aggregated node statistics and edge weights.

When partitioning a set of table nodes, RPA-algorithm groups the tables to form multiple clusters, which are possibly connected by edges. For a transaction instance that is split into q clusters, the number of edges (of weight 1) connecting these q clusters equals $q \cdot (q-1)/2$. This number monotonically increases with q when $q > 1$. Hence, minimizing the number of split transactions, as formulated by O2 in Sect. 4.1, is equivalent to minimizing the number of edges, or aggregated edge weight. Equivalently, the problem of minimizing transaction splits is a graph-partitioning problem, which is to divide a graph into two or more disconnected new graphs by removing a set of edges. As a classic partitioning problem, *minimum cut graph-partitioning* is to remove a set of edges whose aggregated weight is minimal. A constraint for typical graph-partitioning applications is to balance the total node weight of each partition. Differently, the target of our problem is to minimize the maximum PIT consistency latency among all the groups, each corresponds to an individual consistency group.

General Graph-Partitioning Algorithms. Agraph-partitioning problem, as an NP-complete problem in general, is typically solved by heuristics in practice. One widely used algorithm for two-way partitioning (bi-partitioning) is the Kernighan-Lin algorithm (KL algorithm) [13]. It is an iterative improvement algorithm over two existing partitions. It seeks to reduce the total edge cut weight by iteratively swapping nodes in pairs between the two partitions. Fiduccia-Mattheyses algorithm [6] (FM algorithm) further enhances the KL algorithm. By moving a node to a new group, it reduces its edge cut to the other partition while increasing its edge connection to its home partition. It also removes KL algorithm's restriction of moving nodes in pairs. The improved algorithm is referred to as KL-FM algorithm. For large graphs, multi-level bi-partitioning is often applied through graph coarsening and expansion [10]. The quality of their final solutions, which could be a local optimum, is affected by the initial partitioning. Spectral solution [17] can find the global optimum by deriving partitions from the spectrum of the graph's adjacency matrix, but it does not fit our transaction graph model with time series statistics as node weights. Partitioning a graph into more than two partitions can be achieved via a sequence of recursive bi-partitioning. Refinement heuristics for k-way partitioned graph have also been developed [11].

Transaction Split Reduction by Consistency Group Refinement. Before introducing our RPA-algorithm phase-2, we first discuss how to reduce transaction splits between two already partitioned consistency groups by FM algorithms. This process is referred to as an algorithm for 2-CG refinement (CG-RF-2). The process refines the partition via node/table movement. Each move needs to ensure that the PIT consistency latencies for both refined groups remain below PIT_{max} or within a specified margin around PIT_{max}.

Algorithm for 2-way CG refinement(CG-RF-2)

C_1 Create graph representations for each input consistency groups cg_1 and cg_2
C_2 Compute PIT latencies PIT_{cg1} and PIT_{cg2} for cg_1 and cg_2, respectively. Define $PIT_{max}=max(PIT_{cg1}, PIT_{cg2})\cdot(1+\alpha)$ as the upper bound for margin α.
C_3 Compute the gain of each node. The gain for a node table tb_i , as defined in FM algorithm, is computed as the total edge weight between tb_i and all the nodes in the group that tb_i does not belong, subtracted by total edge weight between tb_i and all the nodes in the same group as tb_i, i.e. $g(tb_i)= \sum(|e(tb_i, tb_j)|)- \sum(|e(tb_i, tb_k)|)$ where tb_j belongs in the different group than tb_i , and tb_k belongs in the same group as tb_i. The intuition is that if $g(tb_i)$ is positive, moving tb_i from its current group to the other group reduces the edge cut between the two groups.
C_4 Find the node n_1 with the maximum gain g_1 and whose move from its current group to the other allows each group's PIT consistency latency remains below the PIT_{max} value from C_2. Lock node n_1, mark its movement from its current group to the other as an element mv_1 and store in the moving list mv_list. In some cases, the gain of node n_1 is non-positive. However, it is still moved with the expectation that the move will allow the algorithm to "escape out of a local minimum".
C_5 Update the gains of all the nodes that are connected to n_1 due to its movement.
C_6 Repeat C_4 and C_5 for the rest of the nodes until all the nodes are locked. All movements are stored in mv_list{ $mv_1,...,mv_n$}in the order that they are found . The gains corresponding to these node moving steps is $\{g_1,g_2 ,...,g_n\}$.
C_7 Find the best sequence of $mv_1, mv_2,...,mv_k$ $(1\le k\le n)$ such that $\sum(\{g_1,g_2 ,...,g_k\})$ is maximum and positive.
C_8 Mark the move of these k tables permanent. The refined groups are cg_1' & cg_2'.
C_9 Free all the locked nodes.
C_10 Repeat steps C_3 to C_9 until no move can be found in C_7.

The PIT consistency latency upper bound in *C_2* is set to preserve the optimization objective and speed up the algorithm convergence. When the two input groups are produced by RPA-algorithm phase-1 and α is set to 0, CG-RF-2 algorithm preserves the same maximum PIT consistency latency value from phase-1 while refining the groups for transaction split minimization. When $\alpha > 0$, the PIT consistency latency constraint is relaxed and potentially more nodes are moved to reduce transaction split. Alternatively, a user-supplied PIT threshold H can be used as the constraint.

In some cases, the two-step procedure of bi-partitioning and refinement can be used recursively to create a higher number of partitions, given that the refinement constraint can be distributed along the recursion paths. Such an approach works for throughput-balancing partitioning optimization, i.e. the alternative algorithm RPA-T. However, PIT consistency latency is not a constraint measure that can be easily distributed while still guaranteeing convergence during recursive bi-partitioning. Therefore a non-recursive approach is needed.

RPA-Algorithm Phase-2: K-Way Consistency Group Refinement for Transaction Split Reduction. This section presents the phase-2 of our RPA-algorithm for

transaction split reduction. The algorithm (called CG-RF-k) is derived from the k-way refinement algorithm proposed by Karypis et al. [11].

RPA-algorithm Phase-2 (CG-RF-k)

Ck_1 For the k consistency groups $cg_1,\ldots,cg_k$ created by RPA-algorithm phase-1, create the graph representation for the workload and these k partitions.

Ck_2 Iteration through all the nodes, find the set N_e of all the nodes that each has edge connections to other groups that it does not belong to. Compute the gain for each element in N_e, denote a gain as $g(tb_i, cg_m)$ in which cg_m is a group that node (table) tb_i does not belong but has edge connections to one or more of its nodes. The gain is computed the same as algorithm CG-RF-2 step C_3.

Ck_3 Compose subset N_e' of N_e with nodes that only have positive gains.

Ck_4 For each node tb_i in N_e', test it with its connected groups for potential new PIT latencies. Among those groups whose potential new PIT latencies are below the user specified threshold H, select the group with the largest positive gain for tb_i to move into. If none of the group qualifies the PIT threshold requirement, do not move tb_i.

Ck_5 Update the gains of all the affected nodes due to the move of tb_i, including tb_i. Updates N_e' following the same criteria as in Ck_3.

Ck_6 Repeat steps Ck_4 and Ck_5 until there is no node in N_e'.

RPA-algorithm phase-2 starts its refinement process from the partitioning result of phase-1, which finds the minimum number of groups while satisfying maximum PIT consistency latency threshold. Every node move seeks to reduce the positive gains, i.e. trading higher inter-group edge cut weight with lower intra-group edge cut weight. This process keeps reducing the transaction split count until reaching the lowest.

4.4 Real-Time Partitioning Evolution

In a production environment, partitioning can evolve to adapt to the changing workload patterns and system environment. Both workloads and underlying resources are self-governing agents that are autonomous from replication software. The maintenance issues are even more important than the initial construction, especially in such a dynamic environment. When the detected changes lead to the real time PIT consistency latency increase above a specific threshold H, re-partitioning is executed with the following three-step tasks:

(a) Re-computing the partitioning of database objects with the latest data.
(b) Draining the on-going replication. Since the queued work can be congested at anywhere in between the capture and apply, in reality it is very complicated to re-direct those to different replication channels. During draining, capturing of new workload from current channels is suspended to allow the existing channels to finish replicating queued workload. Newly added channels can start as soon as possible since there is no queued workload in those channels.
(c) Deploying the new replication partitioning configuration into use.

The first two steps can be applied in parallel. Since step (a) usually consumes a considerably shorter time than step (b), we only concern the PIT impact by step (b) in the following discussion. Denote the period using the earlier consistency group configuration P_1, and the period with the modified consistency group configuration P_2. Computing residue throughput and PIT in the situation of re-partitioning is a bit different from what is described in Sect. 4.1. Denoting the PIT consistency latency of the old-period consistency group in the last period as PIT_{p1}, the time for draining the old-period workload td is computed as:

$$td = PIT_{p1} \cdot dt \tag{4}$$

The residual workload REScg,i in the draining period and later should be computed iteratively according to the value of time interval index i:

$$RES_{cg,i} = \begin{cases} RES_{cg,i-1} + \sum IUD(TB_{cg}, t_0 + (i-1) \cdot dt), & i \le PIT_{p1} \\ \max\{(RES_{cg,i-1} + \sum IUD(TB_{cg}, t_0 + (i-1) \cdot dt) - dt \cdot BW), 0\}, & i > PIT_{p1} \end{cases} \tag{5}$$

Accordingly, *PITcg-i* computation should be adjusted as follows:

$$PIT_{cg,i} = \begin{cases} PIT_{cg-old} + (RES_{cg,i-1} + \sum IUD(TB_{cg}, t = t_0 + i))/(dt \cdot BW), & i \le PIT_{p1} \\ (RES_{cg,i-1} + \sum IUD(TB_{cg}, t = t_0 + i))/(dt \cdot BW), & i > PIT_{p1} \end{cases} \tag{6}$$

5 Experiments and Analysis

We applied our work to a batch workload and an OLTP workload. The batch workload is from a banking business and we collected the WPT data from an offloaded production DBMS recovery log. For the OLTP workload, we expanded the schema of TPC-E benchmark [24] and simulated workload profile data for analysis. In both experiments, the analysis processes complete within minutes.

5.1 Transaction Split Avoidance Algorithm

For the purpose of comparison and establishing experimental baselines, we devised an algorithm named Transaction Split Avoidance (TSA). This algorithm assists studying the trade-offs between transaction split and either replication latency or throughput-balancing in all our experiments. We implemented TSA algorithm which seeks reducing PIT consistency latency under the constraints that no transaction split is allowed. Using a transaction graph, the TSA algorithm groups nodes (database objects) connecting to each other directly or indirectly into one *virtual table* by breadth-first search, and then applies the PIT minimization algorithm to these virtual tables.

T_1 Create a graph representation including all tables as nodes and their transaction-related connections as edges.

T_2 Traverse all nodes (tables) in the graph, and partition them into a set of connected sub-graphs with no connection between any two sub-graphs.

T_2-1 Starting from any table node tb_i in the graph, *perform depth-first-traversal to find all table nodes that are reachable directly or indirectly from node tb_i via connecting transaction edges and other connected nodes.*

T_3 Merge tables in each sub-graph into one virtual table by accumulating their workloads. VT (see formula 7) is a collection of such virtual tables. The time series statistics on a virtual table in the time window T is an accumulation (see formula 8) from all real tables that constructs the virtual table.

$$VT = \{vt_1, vt_2, ..., vt_l\} \tag{7}$$

$$\begin{cases} IUD(vt_j, T) = \sum IUD(tb_k, T), \\ tb_k \in vt_j, \\ T = \{t_0, t_0 + dt, t_0 + 2 \cdot dt, ..., t_0 + v \cdot dt\} \end{cases} \tag{8}$$

T_4 Sort all virtual tables in descending order by each virtual table's peak PIT consistency latency max(PIT), represented by VT_{pit}.

T_5 Select a subset VT_{top} consisting of top virtual tables from list VT_{pit}.

T_6 For a specified number of consistency groups L, exhaust all the possible placement of grouping VT_{top} into L CGs. Select the placement with the lowest maximum PIT consistency latency and assign real tables associated with those virtual tables into corresponding consistency groups.

T_7 Iterate through the remaining virtual tables in their descending order in VT_{pit}. Test each virtual table against each consistency group and compute potential maximum PIT latencies PIT_{max_i}, i=1,2,....,L, for each group. If there are some groups whose PIT latencies are not impacted by adding this virtual table, place tables belonged to this virtual table into the group with lowest PIT consistency latency. Otherwise, place these tables into the group whose corresponding PIT_{max} is the lowest.

5.2 Experiment with a Large Bank Batch Workload

This workload profile was collected from a database log representing a four-hour batch processing window with 1 min sample interval. There are 824 tables with active statistics among a total of 2414 tables, and 5529 transaction patterns are discovered from 12.7 million transaction instances. The number of tables correlated by transaction patterns varies between 1 and 27 within the histogram shown in Fig. 7.

We apply the RPA-algorithm with a replication bandwidth *BW = 5* MB/s. To put in prospective, this bandwidth is equivalent to insert 50 K 100-byte records per second into a database. Starting from the lower bound of 3 consistency groups following step 1_2 of RPA-algorithm phase-1, Fig. 8 shows the maximum PIT consistency latency of

each group, in the unit of a sample interval, when the workload is partitioned into 4, 6, 8, 10 or 12 groups. As the number of consistency groups increases, the PIT latencies are reduced for each configuration. The reason that the three highest PIT consistency latency values remain unchanged in 8-, 10- and 12-group cases is because these three groups are assigned with only one volume-heavy table to each group. To further reduce point-in-time latency, single channel replication bandwidth has to be increased by improving the underline replication technologies in network, database, and replication software.

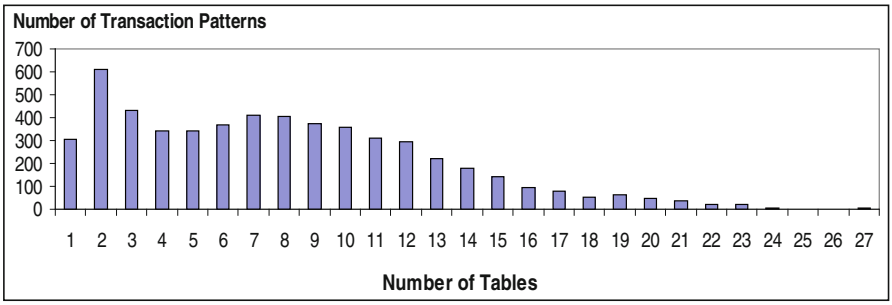

Fig. 7. Distribution of transaction patterns over the number of tables in a batch workload

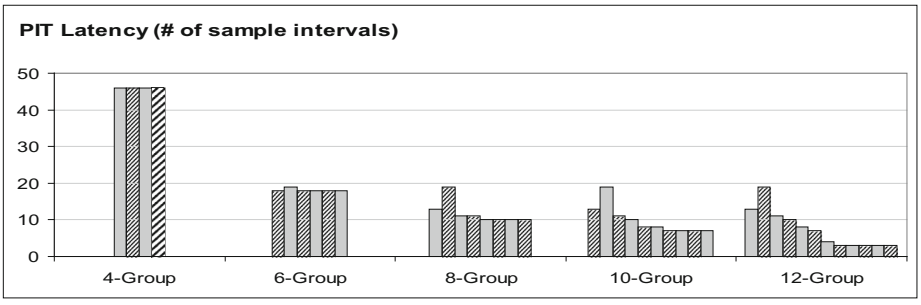

Fig. 8. Partitioning result of batch workload with RPA-algorithm phase-1

Next we apply both phase-1 and phase-2 of RPA-algorithm to reduce transaction splits for a given PIT consistency latency threshold $H = 60$ (1 h). The lowest number of consistency group for this threshold is four from phase-1. Figure 9 shows the result of phase-2. The first chart in Fig. 9 shows the maximum PIT consistency latency of each consistency group using different variations of RPA-algorithm such as phase-1 only, phase-1 plus phase-2 with allowed increase in PIT consistency latency within 0 %, 10 % and 20 % margin, as labeled accordingly in the chart. The second chart in Fig. 9 shows the transaction split distribution in terms of number of groups. Note that splitting into one group means no splitting. TSA algorithm's results are also provided for comparison.

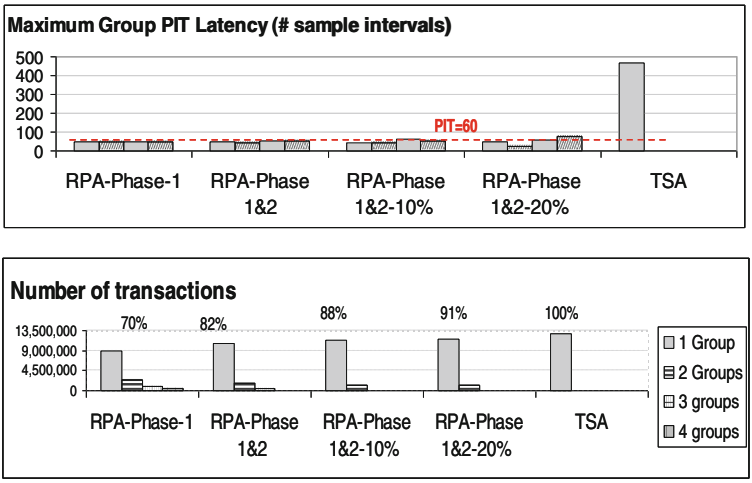

Fig. 9. Partition and transaction split results with RPA-algorithm phase-1 & phase-2 (4 CGs)

The charts show that when phase-2 is used after phase-1, the percentage of non-splitting transactions increases from 70 % with "RPA_Phase1" to 82 %, 88 % and 91 % respectively for RPA_Phase1&2, RPA_Phase1&2-10 % and RPA_Phase1&2-20 %. With the TSA algorithm, all the transactions are non-splitting; however the maximum PIT consistency latency reaches unacceptably high of over 450 1-min sample intervals. In addition to demonstrating that RPA-algorithm can effectively reduce transaction split, the result provides trade-offs study between transaction split and PIT consistency latency.

5.3 Experiment with an OLTP Workload

TPC-E is a newer OLTP data centric benchmark. Its processing is composed of both READ-ONLY and READ-WRITE transactions. Only the READ-WRITE transactions with data changes are used in our study. The TPC-E table schema consists of 33 tables, and 23 of which are actively updated during the transaction execution flows.

To simulate more complex real-world workloads, we expanded the schema by increasing the number of tables by 30× as well as increasing transaction correlations among the tables. Based on the augmented schema and workloads, as well as TPC-E specification on how the tables are updated, we generated a simulated workload profile data with 155 transaction patterns and over 6 million transactions.

OLTP workloads usually update the smaller amount of data within the scope of a committed transaction. Since the volume is lower than the batch, we experiment with our alternative throughput-balancing algorithm (RPA-T-algorithm) and to partition the tables and balance total throughput among 8 consistency groups.

The analyses of the partitioning results using RPA-T phase-1 and RPA-T phase-1&2 are shown in Table 1 and Fig. 10. To be more intuitive, relative standard deviation (RSTDEV = standard deviation/mean) is used to evaluate the effectiveness of

throughput-balancing among consistency groups, as listed in Table 1 for each algorithm. With no surprise, the RSTDEV value is near 0 (0.03 %) for RPA-T phase-1 since it is optimized for balancing throughput; the RSTDEV value for TSA is very high (282 %) since it does not address balancing. Figure 10 offers a different view than Fig. 9 for analyzing how the transaction split is distributed. In Fig. 10, y-axis indicates the percentage of the total transactions that are contained within *x* number or less consistency groups, *x* being the label on x-axis. The percentage values on y-axis increase and reach 100 % for eight consistency groups, i.e. all transactions are replicated within eight groups or less. An algorithm whose curve progresses to 100 % slower than another means that a higher percentage of the transactions are split into more consistency groups when using this algorithm than using the other one. With TSA algorithm, none of the transactions are replicated with more than one consistency group. For RPA-T phase-1 algorithm, only a small number of transactions (0.0015 %) are replicated in one group and 15 % are replicated in one or two groups, etc.

Table 1. Throughput RSTDEV for different algorithm

	RPA-T	TSA	RPA-T phase-1&2 (throughput trade-off %)		
	phase-1		0 %	1 %	5 %
RSTDEV	0.03 %	282 %	0.03 %	1.15 %	7.84 %

Like RPA-algorithm, RPA-T phase-2 seeks to reduce transaction split count among consistency groups generated by RPA-T phase-1. Table 1 and Fig. 10 show that the RPA-T phase-1&2 (0 %) curve progresses only marginally faster than RPA phase-1. Because the activities in this workload are uniformly distributed among different tables and along the time dimension, by not allowing throughput trade-offs (0 %), it limits the number of tables that can be moved during refinement. For further transaction split reduction, more trade-offs are needed on throughput-balancing constraint. As observed from Fig. 10, with 1 % and 5 % allowed adjustment on throughputs constraint during each refinement step, there are significant increases in the number of transactions that

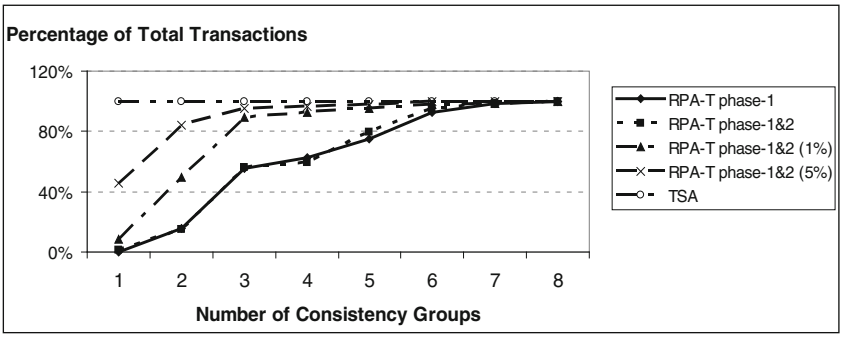

Fig. 10. Transaction split result for OLTP workload

are replicated using less consistency groups. For example, 49.2 % and 84.0 % of transactions are replicated with two consistency groups or less, respectively using RPA-T phase-1&2 (1 %) and RPA-T phase-1&2 (5 %). The trade-offs increase the throughput deviations among groups, e.g. to RSTDEV = 1.15 % for RPA-T phase-1&2 (1 %) and RSTDEV = 7.84 % for RPA-T phase-1&2 (5 %). Such deviation is less significant compared to the reduction in transaction splits.

5.4 Simulation of Partitioning Evolution

We conducted simulation to demonstrate how our RPA tool is applied when real-time workload fluctuates and deviates from the previous profile. In the experiment, we devise a monitor to check the PIT consistency latency of all consistency groups periodically using timestamp information associated with the workloads. When a consistency group's PIT consistency latency is identified higher than the threshold due to the change of run-time environment, re-partitioning is triggered. In this experiment shown in Fig. 11, four consistency groups were used initially in capture and apply pairs (C_1, A_1), (C_2, A_2), (C_3, A_3), and (C_4, A_4), and each replicated a set of database objects not overlapping with the other groups. At the very beginning, the bandwidth of each channel was 300 KB/s. As we can see from Fig. 11, the maximum PIT consistency latency among consistency groups was under the threshold 9 during the first 60 time units (In this experiment, 1 time unit = 1 min). However, at time 60, the system replication bandwidth decreased to 180 KB/s and caused the PIT latencies of all consistency groups to increase significantly. As a result, the PIT consistency latency of one consistency group exceeded the threshold at time 62 and triggered the re-partitioning. We re-applied RPA with the changed bandwidth value and adjusted the partitioning scheme to use as many as eight consistency groups. At the same time, draining started for the queued workload that had been captured but not applied from (C_1, A_1), (C_2, A_2), (C_3, A_3), and (C_4, A_4). At time 69, the draining of all database objects

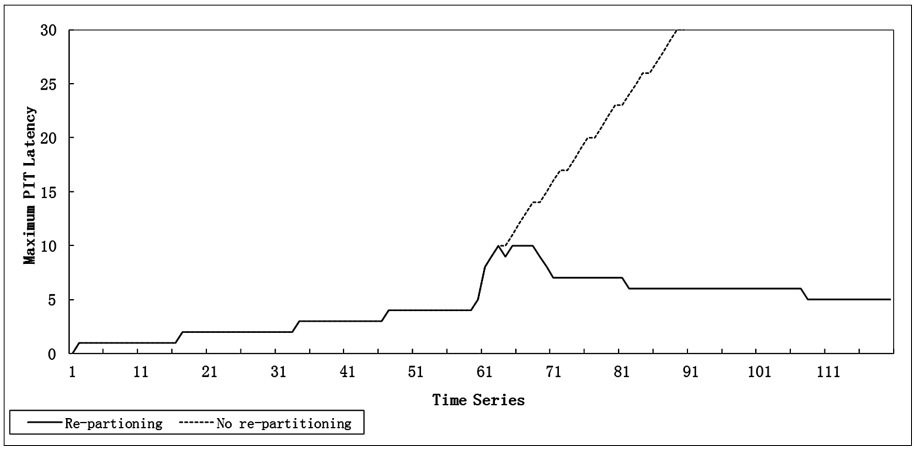

Fig. 11. Simulation of re-partitioning

was finished. All eight replication channels started working to capture and apply any changes since timestamp 62 in RDBMS transaction log. As mentioned in Sect. 4.4, the newly added four channel could have started earlier but for simplicity we started all after the last draining was finished. With the appliance of new configuration, the *PIT consistency latency* had dropped below the threshold after time 69. On the other hand, Fig. 11 also shows that the maximum PIT consistency latency would potentially be beyond 30 if no adjustment was made.

6 Related Work

Database replication is a key technology and a challenging problem for achieving data serving high availability and disaster tolerance [9, 12]. Prior works attempt to address various aspects of replication such as transaction consistency protocols, scalability and performance, etc. (e.g. [14, 15, 19]). In "share nothing" architecture, data replication is used to move data elements among processing nodes to mitigate system failure or to localize transaction processing for better performance [1]. In Spanner [3], synchronous replication is used to achieve transaction consistency in globally distributed data stores. The work in Schism [4] proposes an approach of workload-driven, graph-based replication and partitioning combined with explanation and validation. The work in SWORD [18], targeting data-as-a-service in a cloud environment, achieves higher scalability over prior work with a set of new techniques and introduces incremental re-partitioning. Both works build replication components within the data-serving software.

As reported by Cecchet et al. [1], various challenges still exist when applying database replication in commercial business environments. Motivated by a real-world problem, this paper aims at optimizing middleware-based parallel data replication, especially in a long-distance multi-data-center setting. By filling a gap in understanding database objects affinities with transaction workloads, our work investigates how to group a large number of database objects to improve the performance with a constraint of user-specified PIT consistency latency threshold. To the best of our knowledge, we are the first to propose an automatic design solution to this optimization problem.

We developed heuristics for using a greedy process [8] to achieve the first objective of minimizing the number of consistency groups with a PIT consistency latency constraint. Based on practical analyses, an optimization technique is also proposed to improve the probability of finding a global optimal result. For reducing the transaction splits, which is the second optimization objective, we model the workload as a transaction graph and transform the problem to a graph-partitioning problem. Finally, it is solved by our proposed heuristics based on the existing graph-partitioning algorithms [6, 13, 11]. Both Schism work [4] and SWORD work [18] apply graph algorithms for fine-grain partitioning of tables horizontally in a distributed environment. They model tuples and transactions as graphs and use it to determine the placement of work or data within a cluster of nodes. For the partitioning problem in large-scope data replication across databases and data centers, our workload-pattern-driven approach focuses on modeling and analysis at the database object level. Common graph model and partitioning algorithms provided by existing software such as METIS [21] are not sufficient

for our problem. This is because, in order to address workload fluctuation and address PIT objective, we need to model a workload transaction graph using time series statistics from the tables and the transactions and the computation of PIT consistency latency is iterative with respect to workload volume and time. Our algorithm also needs to introduce problem-related heuristics during the partitioning phase to handle multiple optimization objectives and trade-offs under PIT consistency constraint.

7 Conclusion and Future Work

Large-scale database replication is essential for achieving IT continuous availability. This paper presents a workload discovery and database replication partitioning approach to facilitate parallel inter-data-center data replication that is applicable to both share-nothing and share-disk databases. Our design and algorithms are demonstrated with a real customer batch workload and a simulated OLTP workload. In practice, the work has been applied to a real-world business applications environment. For future work, we plan to further fine-tune the optimization model for the replication stack.

Acknowledgements. We would like to thank Austin D'Costa and James Z. Teng for their insights.

References

1. Cecchet, E., Candea, G., Ailamaki, A.: Middleware-based database replication: the gaps between theory and practice. In: SIGMOD (2008)
2. Codd, E.F.: The Relational Model for Database Management, Version 2. Addison-Wesley, New York (1990). ISBN: 9780201141924
3. Corbett, J.C., et al.: Spanner: Google's globally-distributed database. In: OSDI (2012)
4. Curino, C., Jones, E., Zhang, Y., Madden, S.: Schism: a workload-driven approach to database replication and partitioning. Proc. VLDB **3**, 48–57 (2010)
5. DeCusatis, C.: Handbook of Fiber Optic Data Communication: A Practical Guide to Optical Networking, 4th edn. Academic Press, London (2013). ISBN: 10 0124016731
6. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: Proceedings of the 19th Design Automation Conference, pp. 175–181, January 1982
7. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1990)
8. Graham, R.L.: Bounds on multiprocessing anomalies and related packing algorithms. In: AFIPS Spring Joint Computing Conference, pp. 205–217 (1972)
9. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: SIGMOD (1996)
10. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998)
11. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (1998)
12. Kemme, B., Jiménez-Peris, R., Patiño-Martínez, M.: Database replication. Synth. Lect. Data Manag. **5**, 1–153 (2010). Morgan & Claypool Publishers

13. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Techn. J. **49**, 291–307 (1970)
14. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In: SIGMOD (2005)
15. Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: MIDDLE-R: consistent database replication at the middleware level. ACM TOCS **23**(4), 375–423 (2005)
16. Pavlo, A., Curino, C., Zdonik, S.B.: Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In: SIGMOD (2012)
17. Pothen, A., Simon, H.D., Liou, K.: Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. **11**(3), 430–452 (1990)
18. Quamar, A., Kumar, K.A., Deshpande, A.: SWORD: scalable workload-aware data placement for transactional workloads. In: EDBT (2013)
19. Serrano, D., Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B.: Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In: Proceedings of the 13th PRDC (2007)
20. Stonebraker, M.: The Case for Shared Nothing. IEEE Database Eng. Bull. **9**(1), 4–9 (1986)
21. http://glaros.dtc.umn.edu/gkhome/views/metis
22. IBM Infosphere Data Replication. http://www-03.ibm.com/software/
23. Oracle GoldenGate. http://www.oracle.com/technetwork/middleware/goldengate/
24. http://www.tpc.org/tpce/