Inter-Data-Center Large-Scale Database Replication Optimization – A Workload Driven Partitioning Approach

Hong Min[1,*], Zhen Gao[3,*], Xiao Li[2], Jie Huang[3], Yi Jin[4,**], 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}@tongji.edu.cn
[4] Pivotal Inc. Beijing, China
jinyi.smilodon@gmail.com
[5] IBM System and Technology Group
fuh@us.ibm.com, zhengm@cn.ibm.com

Abstract. Inter-data-center asynchronous middleware replication between active-active databases has become essential for achieving continuous business availability. Near real-time replication latency is expected despite intermittent peaks in transaction volumes. Database tables are divided for replication across multiple parallel replication consistency groups; each having a maximum throughput capacity, but doing so can break transaction integrity. It is often not known which tables can be updated by a common transaction. Independent replication also requires balancing resource utilization and latency objectives. Our work provides a method to optimize replication latencies, while minimizing transaction splits among a minimum of parallel replication consistency groups. We present a two-staged approach: a log-based workload discovery and analysis and a history-based database partitioning. The experimental results from a real banking batch workload and a benchmark OLTP workload demonstrate the effectiveness of our solution even for partitioning 1000s of database tables for very large workloads.

1 Introduction

Continuous availability (CA) is a critical aspect of business information technology resiliency. Enterprises normally maintain multiple active replicas for improving data availability and reducing data loss for the planned and unplanned outages. More enterprises and governments have realized that the data replicas should be geographically dispersed. Large-scale and long-distance data replication is a challenge especially under unprecedented application and data growths.

* Co-corresponding authors.
** Work done while employed by IBM.

H. Decker et al. (Eds.): DEXA 2014, Part II, LNCS 8645, pp. 417–432, 2014.

Various database replication technologies have been proposed for 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] [15] [17]. In an enterprise IT environment that consists of distant data centers across a wide area network (WAN), heterogeneous database architectures and active-active data serving configurations, data replication for continuous availability needs to address additional challenges. Our paper addresses an optimization problem in such an enterprise data replication setting.

Although some argue that integrating replication functionalities inside DBMS provides better replication performance, middleware-based replication solutions are preferable for supporting replications in cross-vendor heterogeneous database environments [11]. Industrial examples of such technology include IBM Infosphere Data Replication [21], Oracle GoldenGate [22], etc. One widely used approach is to capture committed data changes from DBMS recovery log and replicate to target DBMS. Replicating data after changes are committed at the source does not impact the response time of source-side applications. Thus, such an asynchronous solution (a.k.a. lazy replication) is widely applied in the context of WAN data replication.

However, lazy replication introduces data staleness and potential data loss in case of unplanned outages. Higher replication throughput implies shorter data stales and less data loss. 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 [21]. 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 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. 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 waste resources.

This paper proposes an automatic solution for addressing a partitioning problem in parallel middleware-based inter-data-center data replication. Partitioning database objects becomes a critical challenge in PIT latency reduction for parallel replication. First, databases and applications are often designed separately following DBMS data independence principles [2]. It is impractical to decouple database objects from application serving and replication prospective in most cases. Furthermore, new applications are continually deployed on existing databases and access patterns change as business requirements evolve. Giving the workload complexity, database object scales and resource constraints, it is challenging, error-prone and time-consuming for database administrators to understand the comprehensive picture of all the database activities and manually partition database for implementing parallel replication.

Our solution of partitioning database objects aims at employing minimum replication channels for achieving desired point-in-time consistency. We present a workload discovery and history-based partitioning approach based on the observation that similar workload characteristics and patterns reoccur in most business applications. 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 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 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 [19]. 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 in inter-data-center parallel data replication. Section 3 describes briefly the workload profile collected for our RPA tool. Section 4 presents the RPA-algorithm. The experiment evaluations are presented in Section 5. We discuss related work in Section 6, ending with the conclusion in Section 7.

2 Background on Parallel Data Replication

Our work is applicable to an active-active 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 [8]) 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.

Fig. 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 it 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.

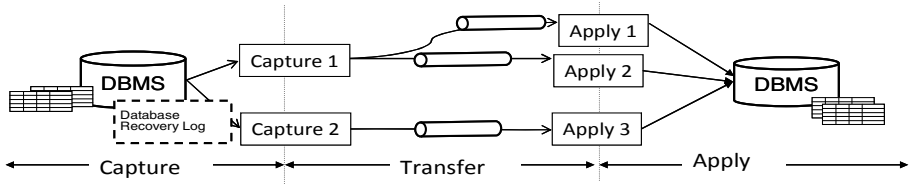

Fig. 1. Logical architecture of parallel lazy replication

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 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 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 $TB=\{tb_1, tb_2,\ldots,tb_k\}$ 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 sub-transactions that each is handled by a consistency group. The replication target eventually reaches point-in-time consistency. 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 latency goal.

3 Workload Profile and Replication Partition Advisor

Workload discovery is performed by our Workload Profiling Tool (WPT). This tool collects transaction information, called WPT workload profile, from DBMS recovery log for another tool, Replication Partition Advisor (RPA), for analysis. The log can either be a real time log or a history log. In the workload profile, all transactions are clustered based on their accessed tables. Each cluster is called a *transaction pattern*, which is characterized by a distinct set of tables or table partitions. Each transaction pattern contains all the transactions that access an identical set of tables or table partitions, but the access order and frequency can vary. For example, given a table set $S=\{A,B,C,D\}$, examples of possible transaction patterns are $P\{A,B,C\}$, $P\{A\}$, $P\{B,C,D\}$, *etc*. In our solution, a table partition is treated as a database object, just as any individual table. For simplicity, the rest of the paper only discusses tables.

Measuring the replication cost per operation depends on the operation type, the table definition and the actual logged column size and values. For example, replicating an update operation requires all the updated column values and the pre-updated key values. The total size of all these column values drives the costs of log capture at the source, network transmission, and operation replay at the target. WPT *workload profile* is defined as the workload patterns with the associated information, which consists of the snapshot time, transaction counts, the number of accessed tables, table schema, and the insert, update and delete volumes in bytes for each snapshot.

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 latency for cg at time $t_0+i \cdot dt$ is the time to process the accumulated residue and new activities at $t_0+i \cdot dt$:

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

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

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

The maximum PIT 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 partition 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:

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

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.

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

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 is 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 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 $TB=\{tb_1, tb_2,\ldots,tb_n\}$ and in time period $T=\{t_0,t_0+dt,t_0+2\cdot dt,\ldots t_0+v\cdot dt\}$.

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 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 $max(PIT_{max}) \le H$. The last L is the minimum number L_{min} of consistency groups.

Given bandwidth $BW(a,c)$ and a user specified PIT 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 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 using 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 latency minimization, which is an NP-hard problem [6]. 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 [7].

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 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 latency contributed by the table. The group with the lowest new maximum PIT 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* (Section 4.1) also exist. An added benefit is that this heuristics tends to generate consistency groups with less PIT 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. Fig. 2 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 latency at its own peak time. As shown in Fig. 2, 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.

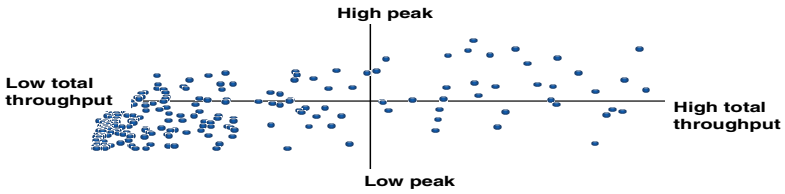

Fig. 2. Table activity distribution in a real-world banking application workload

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 RPA-algorithm is adjusted to balance the peak volume and total volume given a targeted number of consistency groups. Instead of computing PIT 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 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 $TB=\{tb_1, tb_2,\ldots,tb_n\}$ 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 the 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. Fig. 3 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*.

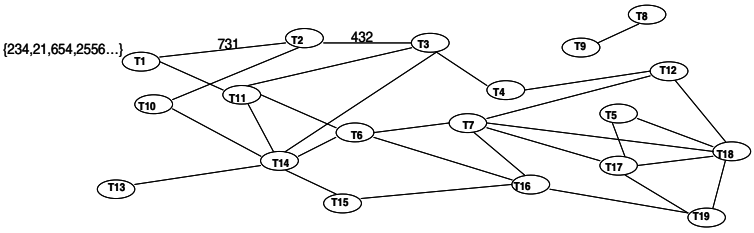

Fig. 3. Transaction graph

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 RPA-algorithm builds 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 Section 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. A graph 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 Kernighan-Lin algorithm (KL algorithm) [12]. 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 [5] (FM algorithm) further enhances 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 [9]. The quality of their final solutions, which could be local optimum, is affected by the initial partitioning. Spectral solution [16] 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 [10].

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 latency remain 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$\{ mv_1,\ldots,mv_n\}$ in the order that they are found. The gains corresponding to these node moving steps is $\{g_1,g_2,\ldots,g_n\}$.

C_7 Find the best sequence of $mv_1, mv_2, \ldots, mv_k$ ($1 \le k \le n$) such that $\sum(\{g_1, g_2, \ldots, g_k\})$ 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 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 latency value from phase-1 while refining the groups for transaction split minimization. When $\alpha > 0$, the PIT 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 [10].

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 transaction split count until reaching the lowest.

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 [23] and simulated workload profile data for analysis. In both experiments, the analysis processes complete within minutes.

5.1 Transaction Split Avoidance Algorithm

For studying trade-offs between transaction split and replication latency or throughput balancing, we devised an algorithm Transaction Split Avoidance (TSA) that partitions database objects without allowing any transaction split. Using transaction graph, TSA algorithm is a modified tree traversal algorithm. Without getting into details, this algorithm works by repeatedly selecting an unassigned table node and grouping it with all the nodes that connect to it directly or indirectly.

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 minute 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 with histogram shown in Fig. 4.

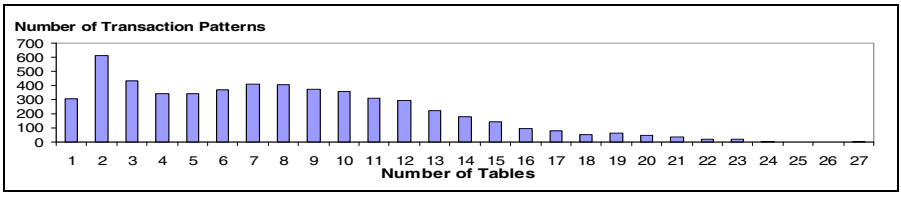

Fig. 4. Distribution of transaction patterns over the number of tables in a batch workload

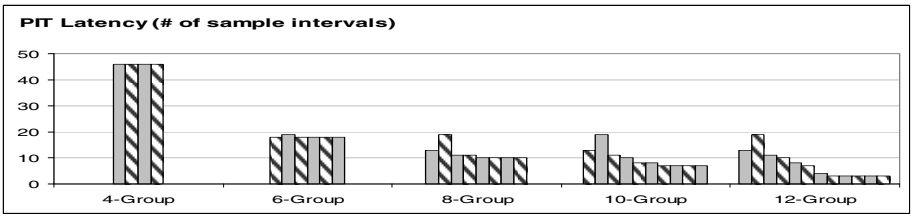

Fig. 5. Partitioning result of batch workload with RPA-algorithm phase-1

We apply RPA-algorithm with a replication bandwidth *BW= 5MB/second*. To put in prospective, this bandwidth is equivalent to insert 50K 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. 5 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 latency values remain unchanged in 8-, 10- and 12-group cases is because these three groups are assigned with only one volume heavy table each. 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.

Next we apply both phase-1 and phase-2 of RPA-algorithm to reduce transaction splits for a given PIT latency threshold *H=60* (1 hour). The lowest number of consistency group for this threshold is four from phase-1. Fig. 6 shows the result of phase-2. The first chart in Fig. 6 shows the maximum PIT 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 latency within 0%, 10% and 20% margin, as labeled accordingly in the chart. The second chart in Fig. 6 shows the transaction split distribution in terms of number of groups, note that splitting into one group means no splitting. TSA algorithm, results are also provided for comparison.

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 TSA algorithm, all the transactions are non-splitting; however the maximum PIT latency reaches unacceptably high of over 450 1-minute 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 latency.

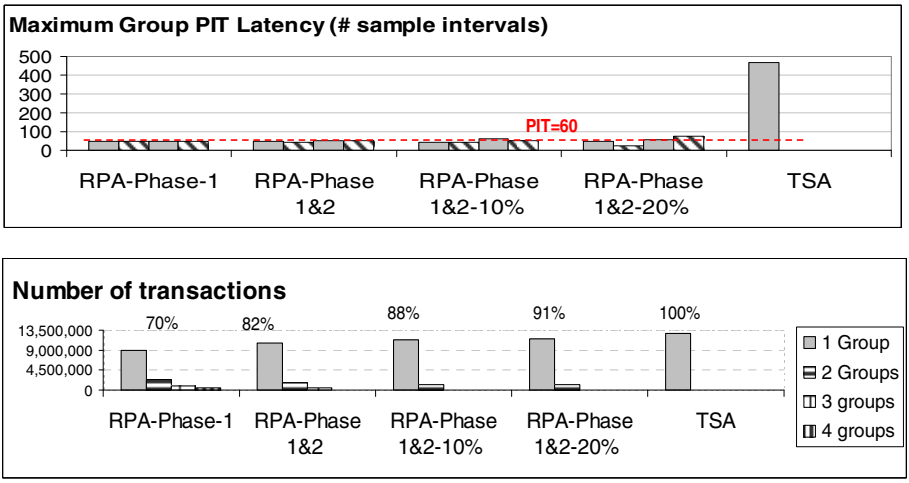

Fig. 6. Partition and transaction split results with RPA-algorithm phase-1 & phase-2 (4 CGs)

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 30x 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. 7. 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 throughout; the RSTDEV value for TSA is very high (282%) since it does not address balancing. Fig. 7 offers a different view than Fig. 6 for analyzing how the transaction split is distributed. In Fig. 7, 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 tradeoff**		
	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. 7 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 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. 7, with 1% and 5% allowed adjustment on throughputs constraint

during each refinement step, there are significant increases in the number of transactions that 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.

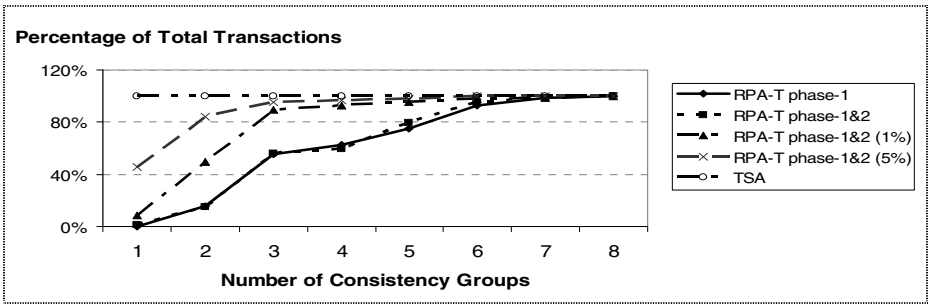

Fig. 7. Transaction split result for OLTP workload

6 Related Work

Database replication is a key technology and a challenging problem for achieving data serving high availability and disaster tolerance [8][11]. Prior works attempt to address various aspects of replication such as transaction consistency protocols, scalability, performance, etc. (e.g. [14] [13][18]) . 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 [7] to achieve the first objective of minimizing the number of consistency groups with a PIT 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 [5][12][10]. Both Schism work [4] and SWORD work [17] apply graph algorithms for finer-grain partitioning of tables horizontally across a distributed environment. They model tuples and transactions as graphs and use them, to determine the placements of

data or works within a cluster. Instead, for resolving a partitioning problem in large-scale data replication across databases and data centers, our workload pattern driven approach focuses on modeling and analysis at database object levels. Common graph models and partitioning algorithms provided by existing software such as METIS [20] are not sufficient for our problem. The major reason is that the transaction graph model needs to support time series statistics and the computation of PIT-consistency latency is an iterative process.

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 real-world business applications environment.

For future work, we plan to further fine-tune the optimization model for the replication stack. We are also interested in looking into how to further automate the cyclic flow of workload profile capturing and inter-database or inter-data-center data replication partitioning and re-adjustment.

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 (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. VLDB (2010)
5. 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 (1982)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co, New York (1990)
7. Graham, R.L.: Bounds on multiprocessing anomalies and related packing algorithms. In: AFIPS Spring Joint Computing Conference, pp. 205–217 (1972)
8. Gray, J., Helland, P., O'Neil, P.: The dangers of replication and a solution. In: SIGMOD (1996)
9. Karypis, G., Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing 20(1), 359–392 (1998)
10. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (1998)

11. Kemme, B., Jiménez-Peris, R., Patiño-Martínez, M.: Database Replication. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2010)
12. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Systems Technical Journal 49, 291–307 (1970)
13. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In: SIGMOD (2005)
14. Patiño-Martinez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: MIDDLE-R: Consistent database replication at the middleware level. ACM TOCS 23(4) (2005)
15. Pavlo, A., Curino, C., Zdonik, S.B.: Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In: SIGMOD 2012 (2012)
16. Pothen, A., Simon, H.D., Liou, K.: Partitioning sparse matrices with eigenvectors of graphs. SIAM Journal on Matrix Analysis and Applications 11(3), 430–452 (1990)
17. Quamar, A., Kumar, K.A., Deshpande, A.: SWORD: scalable workload-aware data placement for transactional workloads. In: EDBT 2013 (2013)
18. Serrano, D., Patino-Martinez, M., Jimenez-Peris, R., Kemme, B.: Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation. In: Proceedings of the 13th PRDC (2007)
19. Stonebraker, M.: The Case for Shared Nothing. IEEE Database Eng. Bull. 9(1), 4–9 (1986)
20. http://glaros.dtc.umn.edu/gkhome/views/metis
21. IBM Infosphere Data Replication, http://www-03.ibm.com/software/
22. Oracle GoldenGate, http://www.oracle.com/technetwork/middleware/goldengate/
23. http://www.tpc.org/tpce/