Replication-driven Live Reconfiguration for Fast Distributed Transaction Processing

Xingda Wei, Sijie Shen, Rong Chen, Haibo Chen
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Contacts: {rongchen, haibochen}@sjtu.edu.cn

ABSTRACT

Recent in-memory database systems leverage advanced hardware features like RDMA to provide transactional processing at millions of transactions per second. Distributed transaction processing systems can scale to even higher rates, especially for partitionable workloads. Unfortunately, these high rates are challenging to sustain during partition reconfiguration events. In this paper, we first show that state-of-the-art approaches would cause notable performance disruption under fast transaction processing. To this end, this paper presents DrTM+B, a live reconfiguration approach that seamlessly repartitions data while causing little performance disruption to running transactions. DrTM+B uses a pre-copy based mechanism, where excessive data transfer is avoided by leveraging properties commonly found in recent transactional systems. DrTM+B's reconfiguration plans reduce data movement by preferring existing data replicas, while data is asynchronously copied from multiple replicas in parallel. It further reuses the log forwarding mechanism in primary-backup replication to seamlessly track and forward dirty database tuples, avoiding iterative copying costs. To commit a reconfiguration plan in a transactionally safe way, DrTM+B designs a cooperative commit protocol to perform data and state synchronizations among replicas. Evaluation on a working system based on DrTM+R with 3-way replication using typical OLTP workloads like TPC-C and SmallBank shows that DrTM+B incurs only very small performance degradation during live reconfiguration. Both the reconfiguration time and the downtime are also minimal.

1 INTRODUCTION

Many applications like web services, stock exchange and e-commerce demand low-latency, high-throughput transactions over a large volume of data. Modern transaction processing systems scale by sharding data across a number of machines. State-of-the-art transaction processing systems like H-Store [16] and Silo [27, 30] have achieved orders of magnitude higher performance than previous systems. Recent designs like DrTM [29, 5], FaRM [11], and FaSST [15] further achieved millions of transactions per second on a small-scale cluster by exploiting novel hardware features like RDMA.

While sharding essentially distributes the client loads across multiple machines, it also faces a new challenge such that an improper data sharding scheme would cause notable workload imbalance as well as degraded overall performance. Work imbalance becomes even more severe for dynamically skewed workloads where the hotspot constantly changes over time [7, 3, 19, 17]. For example, the volume on the New York Stock Exchange (NYSE) during the first and last ten minutes of the trading day is an order of magnitude higher than at other times [24], while the access pattern is indeed skewed. This not only requires many distributed accesses in transactions but also causes frequent transaction aborts or stalls due to contended accesses on certain partitions. For example, our evaluation shows that when moving the warehouse selection from uniform to a highly skewed distribution, the transaction throughput degrades by 10X for TPC-C [26].

Prior approaches tackle this problem through *live reconfiguration* of the sharding plan [24]. Here, an optimal reconfiguration plan needs to balance among the following key requirements: 1) non-intrusiveness to running transactions; 2) minimal data movement; 3) balanced load after reconfiguration. To this end, E-Store [24] first generates an optimal plan according to current load distribution by re-assigning database tuples. It then uses Squall [12] to apply the new physical layout by migrating tuples online. Specifically, Squall fetches tuples on demand from the source node to reduce downtime. We term such an approach as a *post-copy* migration scheme, as an analogy to the live migration of virtual machines [6].

While post-copy approaches like Squall have been shown to be effective to quickly balance the load for H-Store [16], our investigation finds them ineffective for transaction processing systems with orders of magnitude higher throughput like DrTM [29, 5], FaRM [11] and FaSST [15]. This is due to prohibitive performance degradation during the lengthy post-copy phase, which is caused by enormous data transfer costs. Specifically, we have implemented the Squall-like approach on DrTM+R [5] and evaluated the effectiveness on TPC-C [26]. Even under configuration with low skew, there are near 4 seconds where millions of transactions execute with extremely low throughput and high latency, which means tens of millions of transactions were disrupted or stalled during the live reconfiguration process. This is prohibitively expensive for a transaction processing system demanding sub-millisecond latency and stable performance. Consequently, this leaves the system trapped between two bad options: it can either delay reconfiguration and thus run with somewhat degraded performance for a long period, or it can reconfigure and thus run with seriously degraded performance for a short period.

This paper describes DrTM+B, a live reconfiguration scheme targeting distributed transaction processing with high throughput and low latency. Unlike prior post-copy based approaches [13, 12], DrTM+B uses a *pre-copy* based approach to reducing disruption to running transactions. Traditionally, pre-copy mechanisms [9, 10] would iteratively migrate tuples to the destination node (pre-copy phase) and only start the live reconfiguration process (commit phase) when the difference between the source and the destination nodes drops below a threshold, or the number of iterative copies exceeds a threshold. While such a pre-copy based approach causes little service disruption time, it may cause a notable downtime during the commit phase and the number of tuples to be transferred during the iterative pre-copy phase may still be non-trivial. To this end, DrTM+B also incorporates two key optimizations to further reduce data transfer and disruption to transactions.

First, we propose a novel reuse of fault-tolerant replicas to accelerate data transfer. This is based on the observation that state-of-the-art distributed transaction systems such as FaRM [11], FaSST [15] and DrTM+R [5] use primary-backup replication to tolerate failures. DrTM+B's reconfiguration plans take this into account: the new configuration uses previous backup nodes when possible to avoid physical tuple movement. This optimization can commonly avoid data copying in the pre-copy phase. When data copying is necessary, DrTM+B leverages all existing replicas by asynchronously pulling data in parallel. This can shorten the migration time and reduce disruption to transactions since the data copying uses one-sided RDMA operations to bypass the CPU of the source node, which may be busy with the transaction processing.

Second, data migration with pre-copy usually requires tracking and copying dirty data, even when the destination has already held a data copy (has a backup replica). This results in more migration iterations and lengthy downtime since many dirty tuples may be generated during the data migration process. To this end, DrTM+B reuses its fault-tolerance logging mechanism for tracking and copying dirty tuples. Therefore, no additional data copying is needed in the commit phase. Further, DrTM+B employs a cooperative commit protocol to minimize the downtime required to migrate final states, where the concurrency control protocol is slightly modified to be aware of the configuration change.

We have implemented DrTM+B by extending DrTM+R [5]. The extensions include reusing fault tolerance mechanism for live reconfiguration and making the OCC protocol aware of the reconfiguration process.[1] To demonstrate the effectiveness of DrTM+B, we evaluated it using TPC-C and SmallBank with changing skewness. Evaluation results show that DrTM+B can complete a reconfiguration within 40 milliseconds if existing replicas suffice to balance the workload. With data copying, DrTM+B only takes 3 seconds and 1 second to reconfigure TPC-C and SmallBank workloads with only 7% and 3% throughput drop during reconfiguration respectively. There is no observable downtime and DrTM+B finishes live reconfiguration significantly faster than existing post-copy approaches.

In summary, this paper makes the below contributions:

- A pre-copy based scheme to reduce downtime of live reconfiguration. (§3)
- Two key optimizations that further minimize data transfer, service disruption and downtime. (§4)
- An intensive evaluation that confirms the effectiveness of DrTM+B. (§6)

2 Background and Motivation

2.1 Fast In-memory Transaction Systems

Recent commoditization of advanced hardware features like RDMA have stimulated the design and implementation of fast distributed in-memory transaction systems like DrTM [29, 5], FaRM [11] and FaSST [15]. These systems exploit the low-latency transmission of RDMA to boost distributed transactions, resulting in orders of magnitude higher throughput compared to previous systems and reaching millions of transactions per second even on a small-scale cluster.

To achieve high throughput and low latency, such systems follow a local execution mode such that each worker thread will run a transaction to completion. The request will be routed to the node which contains most of the tuples the transaction accesses for efficiency. To scale, these systems continue the practice of H-Store [16] and others by splitting a large volume of data into multiple partitions spreading across multiple nodes.

To achieve high availability upon failures, a common approach is to use Vertical Paxos [18] with primary-backup replication [16, 11, 5, 15]. The primary-backup replication has been shown to have a lower number of replicas to tolerate the same number of failures compared to Paxos [11]. Under primary-backup replication, each partition is commonly configured to use 3-way replication (one primary and two backups). The transaction will write the log at each node with a backup before committing on the primary. To make backups catch up with

[1] DrTM+B can coexist with the fault-tolerance mechanisms based on primary-backup replication [5, 11] since both the pre-copy phase and the commit phase are applied in a transactionally safe way.

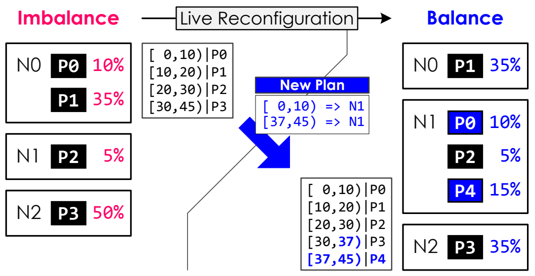

Fig. 1: *An example of reconfiguration.*

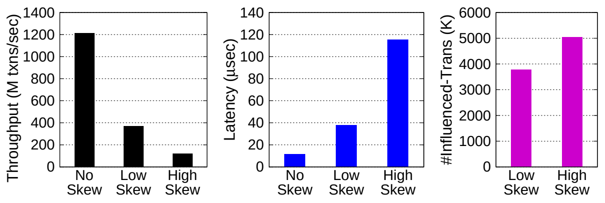

Fig. 2: *(a) The throughput and (b) the latency of DrTM+R, as well as (c) the number of influenced transactions during reconfiguration for TPC-C with different workloads.*

the primary, each node will periodically apply updates in logs to the backup replicas in background.

2.2 Skewed Workloads & Live Reconfiguration

While sharding scales out in-memory transactions, it is hard or even impossible to find a static sharding configuration that works well for all workloads, especially for those with dynamic changing hotspots and skewness [24]. Actually, prior work on production applications has shown that hot objects change rapidly over time [3, 7, 19, 17]. Some e-commerces like daily deals and flash sales can abruptly and significantly increase the visits and transactions on particular products, resulting in order spikes [1].

A change in skewness can cause severe load imbalance, leading to notably degraded overall performance. Fig. 1 shows a sample database which is partitioned into 4 partitions by the range of key and is initially assigned to 3 nodes with a balanced workload. However, the presence of dynamical changes in load may result in skewness, where some of the nodes will become overloaded while others may be idle. For example, if most accesses currently focus on the tuples in Partition 1 and 3 (P1 and P3), Node 0 and 2 (N0 and N2) will be overloaded while Node 1 (N1) will be underloaded.

To illustrate the performance impact from skewed workloads on fast distributed transaction systems, we conducted an initial experiment using DrTM+R [5] for TPC-C [26] on a 6-node cluster. For this experiment, TPC-C scales by partitioning a 25-million-tuples database into 192 warehouses (32 per node). We test different skewed settings (no skew, low skew and high skew), and report the average throughput and latency of the system. Additional details of our experimental setting are described in Sec. 6. As shown in Fig. 2(a), the throughput decreases by 3.3X and 10.0X from no skew to low skew and high skew respectively. In addition, as shown in Fig. 2(b), the increases of latency also reach 3.7X and 11.0X for low skew and high skew respectively.

Hence, it is highly desirable for a distributed in-memory transaction system to support fast and seamless live reconfiguration to quickly adapt to frequent workload changes. As shown in Fig. 1, the current hotspots on the sample database occur in Partition 1 and 3 (P1 with 35% and P3 with 50%). A proper reconfiguration plan is generated to lively migrate the Partition 0 and 4 (P0 and P4) from the overloaded nodes (N0 and N2) to the underloaded node (N1). To achieve this, some partition (i.e., P3) may need to be split to achieve fine-grained elasticity in the reconfiguration plan.

2.3 Disruptiveness of Post-copy Migration

To address the above issues, a recent state-of-the-art system called E-Store [24] has provided the live reconfiguration feature by automatically identifying whether a reconfiguration is needed and creating a new plan to balance the load across nodes. E-Store uses Squall [12] to execute the plan by lively migrating data among nodes in a transactionally safe way. Specifically, Squall follows the post-copy based approach [13], which first applies the reconfiguration plan and then pulls database tuples in an on-demand (reactive) way. It further introduces optimizations such as asynchronous migration and splitting reconfigurations.

Fig. 3 illustrates the timeline of the post-copy approach adopted by Squall. After receiving a new plan (migrating P0's primary from N0 to N1), the reconfiguration manager (RM) starts a reconfiguration by broadcasting the new plan to all nodes. All nodes then temporally suspend accesses to P0 and wait for all outstanding transactions on P0 to finish. After that, RM notifies all nodes to update the new location of P0's primary in the mapping table from partitions to nodes (PN table) and resume accesses to P0. Afterward, N0 will periodically migrate the tuples of P0 to N1 in an *asynchronous* way. Meanwhile, all transactions involving P0 will also be (re-)assigned to N1. N1 will examine whether the tuples the transaction accesses have been migrated. If not, N1 will block the transaction and send a pull request to *reactively* migrate the tuples from N0.

The post-copy scheme aims at reducing the downtime and avoiding repetitive data transfer. However, this causes significant disruption to transaction processing during data migration and a lengthy period of migration time due to two main reasons. First, transactions will be blocked due to missing database tuples. The transactions that access multiple tuples will mostly be blocked, even by multiple times. This is especially an issue for standard OLTP applications like TPC-C [26]. Second, migrat-

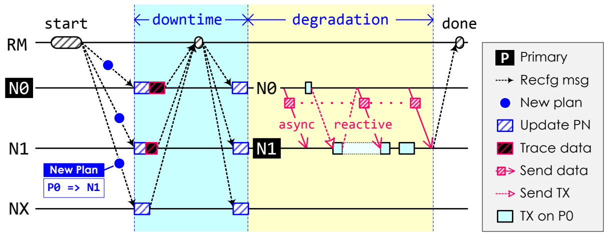

Fig. 3: *The execution flow of post-copy based reconfiguration.*

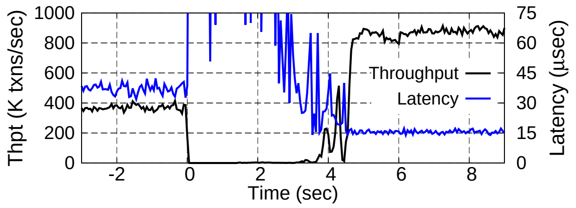

Fig. 4: *The throughput and median latency timeline of Squall live reconfiguration on DrTM+R for TPC-C with low skew.*

ing data will compete CPU and memory bandwidth with transaction processing on an already overloaded node, which may further aggravate the imbalance. The length and strength of performance degradation would be prohibitive for ultra-fast OLTP systems [5, 11, 15] that process millions of transactions per second with tens of microsecond latency.

To illustrate this, we implemented a post-copy based live reconfiguration on DrTM+R [5] with all optimizations in Squall [12], and conducted an experiment for TPC-C [26] with various workloads. Fig. 2(c) shows that the post-copy approach will disrupt several millions of transactions. Worse even, the throughput of DrTM+R degrades to nearly zero for more than 4 seconds when reconfiguring a TPC-C workload with *low skew*, as shown in Fig. 4. This is because the transactions in TPC-C require accessing multiple tuples, where fetching database tuples on demand would easily cause lengthy stalls and even aborts of transactions. Even worse, the length of performance degradation in the post-copy approach will proportionately increase with the growth of data size and the number of tuple accesses in a transaction. This becomes an even more serious issue for rapidly changing workloads that require frequent live reconfiguration [3, 7, 19, 17].

3 Overview of DrTM+B

DrTM+B is designed to support live reconfiguration for fast distributed in-memory transaction systems (§2.1). Like E-Store [24], DrTM+B contains two cooperative parts: Monitor and Planner. The monitor detects load imbalance and identifies the tuples causing it, and the planner decides whether there is a need to reorganize the tuples and generate the corresponding reconfiguration plan.

Unlike Squall [12], DrTM+B adopts a *pre-copy based* approach to migrating tuples, like those widely used in the live migration of virtual machines [6]. Fig. 5 illustrates a typical pre-copy mechanism which contains two consecutive phases: *iterative pre-copy* and *commit*. In the iterative pre-copy phase, the involved data is first copied from the source node to the destination node (i.e., Data-Copy), then the dirty data is iteratively copied to the destination (i.e., Diff-Copy). When the amount of dirty data is small enough or the number of iterations exceeds a threshold, the commit phase starts where it performs final data synchronization (i.e., Data-Sync) to transfer remaining dirty data, and state synchronization (i.e., State-Sync) to (re-)assign transactions according to the new reconfiguration plan.

Issues with pre-copy. While the pre-copy based approach can minimize service disruption during data migration, there are two main issues limit its effectiveness in fast in-memory transaction systems. First, the pre-copy phase may be lengthy with large data transfer and may even hard to converge, since the high frequency of transaction processing will produce a huge amount of dirty tuples. Second, it is hard to find an efficient way to track the dirty tuples during migration and synchronize such tuples to destination nodes.

Observation. To address the above issues, we exploit two observations in fast in-memory transaction systems with high availability [11, 5, 15], where the primary-backup replication is used to tolerate failures. First, as the backup replicas contain nearly the same content as the primaries, it is possible to reuse the *fault-tolerant replicas* to avoid most data transfer in the pre-copy phase. Second, these systems typically use *log forwarding* to synchronize data between primaries and backups. It opens an opportunity to freely track and synchronize the updates on tuples during data migration.

Our approach. In the pre-copy phase (§4.2), DrTM+B prefers to reuse the *fault-tolerant replicas* to skip the Data-Copy by a replication-aware reconfiguration plan (§5). Note that DrTM+B may split a partition when its granularity is not small enough to balance the workload (e.g., P3 in Fig. 1). For extremely rare cases where all nodes holding the replicas of a hot partition are also overloaded, DrTM+B will create a new replica for the hot partition at a spare node. Furthermore, since the source node holding the hot partition is busy with transaction processing, DrTM+B uses one-sided RDMA READ on the destination node to *asynchronously* pull tuples from all replicas in parallel. For the Diff-Copy, DrTM+B seamlessly reuses the *log forwarding mechanism* in replication systems to freely track and synchronize the new updates on migrated tuples (i.e., dirty), since the log essentially contains the changes of each

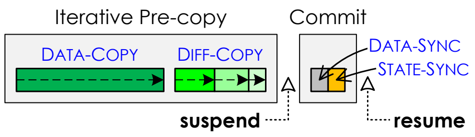

Fig. 5: *The execution flow of pre-copy live reconfiguration.*

committed transaction. Further, DrTM+B can concurrently execute DATA-COPY and DIFF-COPY.

In the commit phase (§4.3), DrTM+B supports the DATA-SYNC by *draining updates* in logs to the backup replica. When the amount of logs is too much and thus may cause a notable downtime, DrTM+B uses a *cooperative commit protocol* where a primary continues to commit transactions while the backup is applying logs. Yet, the transaction commit protocol (e.g., OCC) is modified such that the log versions of involved transactions are forwarded to the primary. Next, when the backup has applied all previous logs, it can quickly apply the remaining logs only synchronizing with the primary. In DrTM+B, the STATE-SYNC usually takes little time since it only needs to update a few state tables (e.g., PN table). Finally, one-sided RDMA READ is used to actively watch the state of the node holding the new primary, which helps timely resume the execution on migrated data.

4 REPLICATION-DRIVEN LIVE RECONFIGURATION

Our goals are to minimize service disruption and downtime while completing a reconfiguration as fast as possible. This section presents the detailed design on how DrTM+B optimizes both the pre-copy and the commit phases by a novel reuse of primary-backup replication.

4.1 Basic Data Structure

Database tuples in DrTM+B are assigned to multiple disjoint partitions, which are further distributed across multiple nodes. For brevity, this paper uses range partitioning as an example.[2] DrTM+B maintains a few data structures to support fine-grained live reconfiguration, as shown in Fig. 6. The first one is a mapping table from key to partition (KP table), which makes it possible to provide fine-grained reconfiguration at the tuple level. The second one is a mapping table from partition to node (PN table), which maintains the type (i.e., primary (`P`) or backup (`B`)) and the state (i.e., whether the transaction is executable (`E`) or not (`N`)) of each replica. Both KP and PN tables are replicated and individually changed on each node. A reconfiguration will change the two tables to reflect the new plan.[3]

[2]A two-tiered partitioning [24] can be used to fully support fine-grained planning and live reconfiguration at the tuple level.

[3]Each node updates the local tables atomically via an RCU-like mechanism during live reconfiguration and after all worker threads have executed at least one transaction.

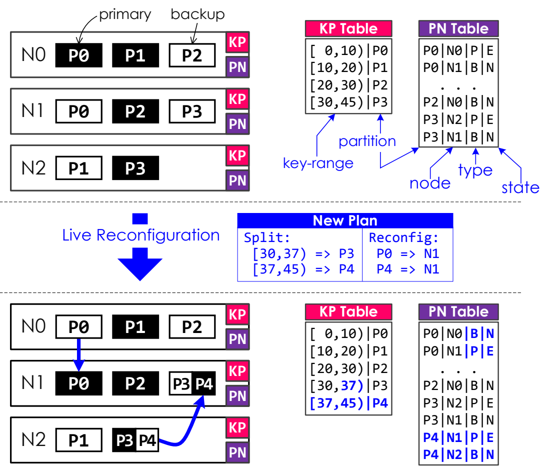

Fig. 6: *An overview of live reconfiguration on DrTM+B. In PN table,* **P** *and* **B** *stand for "Primary" and "Backup", which mean the type of the partition.* **E** *and* **N** *stand for "Executable" and "Non-executable", which mean the state of the partition.* **C** *and* **X** *stand for "Create" and "eXchange", which are used for creating partition and notifying transactions to enter exchange mode (see Fig. 7,8, 9, and 10).*

4.2 Pre-copy Phase

Generally, DrTM+B prefers to reuse existing replicas for fault tolerance to support live reconfiguration such that data transfer can be avoided. The underloaded nodes holding backup replicas are superior candidates to take over the workload from the overloaded nodes. However, for some highly skewed workloads, performing data migration at a partition granularity may not be able to achieve an optimal balance. Further, a spare node with the backup for the partition cannot always be found. DrTM+B addresses these two issues through *partition splitting* and *asynchronous replication* accordingly.

Partition splitting. To support fine-grained migration at *tuple level*, DrTM+B allows to split a single partition into multiple ones and can reconfigure these new partitions individually. For example, in Fig. 6, P3 is split into two new partitions (i.e., P3 and P4), and one of them (P4) is migrated from N2 to N1 in the commit phase.

Splitting a partition has minimal impact on outstanding transactions since there is no real data movement involved at this stage. Yet, we cannot naively split one partition. Consider the example in Fig. 6, where keys from [37,45) are being re-assigned from P3 to P4. If some transaction updates the key 40 in P3 on N2, while the key has been assigned to P4 at backup replica on N1, this replica may miss this update which causes inconsistency. It is vital to split the partition among all nodes *synchronously*. Moreover, all previous logs of committed transactions should be applied on all backups in advance. Consequently, DrTM+B defers the commit of splitting partitions to the commit phase (§4.3), which can synchronously change the configuration to the entire cluster.

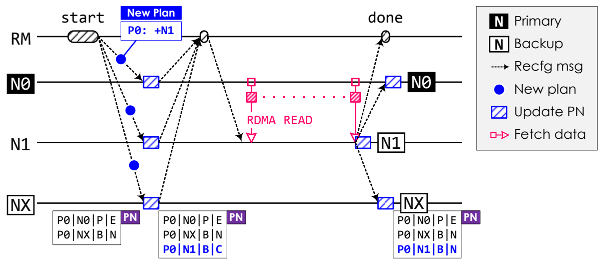

Fig. 7: *The execution flow of asynchronous replication. A new backup replica of P0 is created on N1.*

Asynchronous replication. The number of replicas for fault tolerance depends on the requirement of degrees for availability. For 3-way replication, there are two candidate nodes that can be considered when reconfiguration without data movement. However, in some rare cases, the planner cannot find an optimal reconfiguration plan under current distribution of replicas. DrTM+B also allows to asynchronously create a new backup replica for some tuples in hot partitions on a spare node and then migrate the workload to that node.

To avoid adding new workload to hot nodes, DrTM+B adopts a pull-based approach, which leverages one-sided RDMA READ on the spare node to fetch tuples of the partition. To ensure the new backup replica receives the updates from running transactions, the reconfiguration manager (RM) will first acknowledge every node to add a new backup replica entry to its PN table. Consequently, subsequent transactions will write logs to the node which will hold the newly created replica. The spare node will start to fetch tuples from the primary of the partition after the notification from RM, which avoids missing some updates on the primary. Note that the spare node may fetch the tuples updated by transactions when creating the new replica. Yet, DrTM+B can still guarantee the consistency of the new backup even though the updates are duplicated in logs. Since each update has a version, the updates with an out-of-date version in the log or from the primary will be simply skipped. Finally, all nodes will receive the notification of the completion of asynchronous replication, and the new replica can be treated as a normal backup of the partition.

An example of asynchronous replication is shown in Fig. 7 where another copy of P0 is created on N1. For simplicity, we just create the whole P0 on N1 without losing generality. The process starts when RM sends the new plan to all nodes in the cluster. A new entry for N1's P0 with the state *Create* (**C**) is added to PN table when each node receives the new plan. After that, subsequent transactions will write logs to N1 after changing some tuples in P0, while N1 will drain updates in the log to the newly created backup. Each node replies to RM when all transactions have noticed the new plan. After all acknowledgments are collected by RM, RM then notifies N1 to fetch tuples from the primary of P0. At last, N1 will notify all nodes to update the state of N1's P0 to *Non-executable* (**N**) in their PN tables.

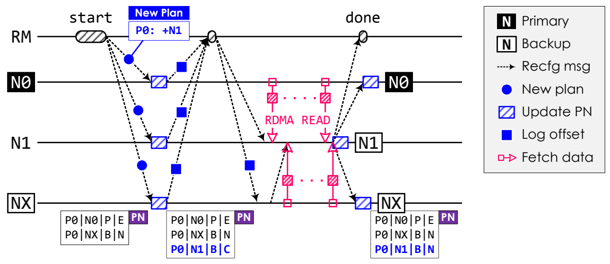

Fig. 8: *The optimized execution flow of asynchronous replication. A new backup replica of P0 is created on N1.*

Parallel data fetching. While asynchronous replication has no interference with the execution of transactions, it may still delay the reconfiguration and increase the degree of imbalance. Since data fetching dominates the execution time of asynchronous replication, DrTM+B optimizes this by splitting the migrated data into multiple disjoint ranges and fetching them from multiple replicas in parallel. However, the backup replicas may be *stale* since some logs have not been drained. Hence, the RM must collect the latest log versions (i.e., log offsets) before changing PN table on all nodes[4] first, and then informs each backup to apply its logs beyond such versions. After that, the backups can allow the spare node to fetch data from them.

As shown in Fig. 8, the log offset of P0 on each node is piggybacked to the acknowledgment message to the RM, and the RM sends the collected log offsets to every node with the backup replica of P0 (NX in this example). After NX's logs have been drained according to the collected log offsets, NX sends an explicit message to N1 to invite it to fetch P0's data from its backup replica.

4.3 Commit Phase

The pre-copy phase guarantees that *a destination node has already possessed a backup replica with the migrated tuples, and new updates to these tuples are being forwarded through the primary-backup logs*. However, to commit a new configuration, it is necessary to atomically promote the backup to the new primary and demote the previous primary to the backup. Moreover, before exchanging the roles, DrTM+B must ensure that all updates in the log of the backup have been applied so that the primary and the backup are equivalent. Unfortunately, in common concurrency control protocols [11, 5], each transaction individually writes logs to all backup replicas of modified tuples. This means that each node contains the updates from all nodes and is not fully aware of which transaction wrote the updates and when.

[4]Each transaction will individually write the logs to all backups, such that each backup contains the logs from all nodes [11, 5, 15].

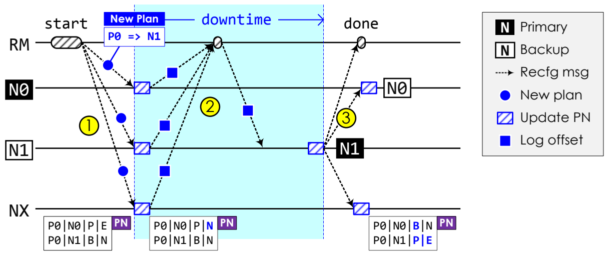

Fig. 9: *The execution flow of basic commit protocol.*

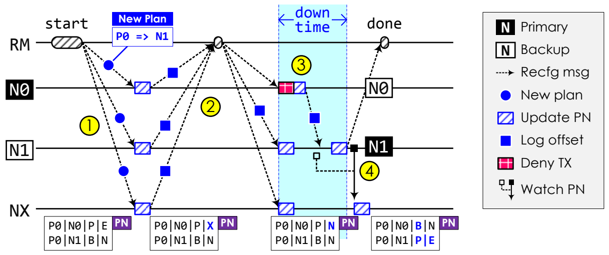

Fig. 10: *The execution flow of cooperative commit protocol.*

Basic Commit Protocol. The basic commit protocol will first suspend all involved transactions on every node, and then apply the logs to candidate backup replicas. Finally, all nodes will resume the involved transactions with new configuration. Fig. 9 shows the timeline of the basic commit protocol, where DrTM+B attempts to *exchange* the roles of P0's replicas on N0 (source) and N1 (destination) by executing the following steps:

1. ***Suspend.*** The RM will inform all nodes the new plan, and they first suspend all involved transactions on P0 by updating the state of P0's primary in PN table from *Executable* (**E**) to *Non-executable* (**N**).

2. ***Collect.*** Every node will send its log offset to the RM. The RM waits for all responses and informs the destination node to drain its logs according to the offsets.

3. ***Resume.*** After draining the logs, the destination node will inform the RM that the commit phase has done, and notify other nodes to resume the execution of involved transactions by updating their PN tables again. Each node will exchange the roles (i.e., primary and backup) of replicas and change the state of new primary from *Non-executable* (**N**) to *Executable* (**E**).

Note that the logs are continuously applied in background on each node. Hence, the logs on the destination node may have been applied beyond the log offsets before receiving the notification from the RM.

Cooperative Commit Protocol The downtime of the basic commit protocol highly depends on the number of updates in logs to be applied. Since the logs from the same node are stored together to be efficiently appended using one-sided RDMA WRITE, it is hard and not cost-effective to apply logs for a certain backup. Consequently, the destination node has to apply all logs from all nodes, which may result in a lengthy downtime of the commit phase when facing massive transactions.

Further, in DrTM+B and other systems [11, 5] which leverage RDMA primitives for log forwarding, the log versions are scattered over all nodes. Thus the log offsets must be collected from all nodes in the cluster to indicate the latest state of migrated tuples in the backup. Hence, the exchange of roles must be carried with a global coordination. Moreover, the destination node must notify all nodes to resume involved transactions after the commit phase, which may incur non-trivial downtime.

To remedy this, DrTM+B optimizes the basic exchange protocol to minimize the downtime by adding a new *exchange mode* for traditional concurrency control protocol (e.g., OCC). DrTM+B will ask all nodes running involved transactions in exchange mode by changing the state of primary in PN table from *Executable* (**E**) to *eXchange* (**X**). Under *exchange mode*, each node can still execute involved transactions but need to forward their log versions to the primary in addition. Then the primary can initiate the exchange and delivery the log offsets to the destination. The destination node will apply the logs and update the configuration in the PN table. The rest of nodes will watch the state of the new primary on demand and resume the execution of involved transactions. Fig. 10 shows the execution flow of the cooperative commit protocol, where DrTM+B attempts to exchange the roles of P0's replicas on N0 (source) and N1 (destination) by executing the following steps:

1. ***Prepare.*** The RM will inform all nodes the new plan, then every node notifies involved transactions to enter into the exchange mode by changing the state of P0's primary in its PN table from *Executable* (**E**) to *eXchange* (**X**). When the transaction executes in the exchange mode, it will explicitly inform the primary (N0) its log version during commitment with message. N0 will abort such transactions if it has already transferred its ownership.

2. ***Collect.*** Every node will send the log offset to the RM. The RM waits for all responses and informs the destination node to drain its logs according to the offsets.

3. ***Suspend.*** After receiving the message from RM, every node informs involved transactions to leave the exchange mode by setting the state of P0's primary to *Non-executable* (**N**). The primary (N0) starts to transfer P0's ownership to N1; it first denies transaction committing from other nodes and then informs N1 with the further log offsets collected from transactions running in the exchange mode.

4. ***Resume.*** To resume the execution on migrated data, N1 waits for the logs to be drained according to the offsets received from N0. Then it updates its PN table to resume involved transactions by exchanging the roles (i.e., primary and backup) of replicas and changing the state of new primary to *Executable* (**E**).

Unlike the basic commit protocol, DrTM+B leverages an RDMA-friendly watching mechanism to timely notify all nodes to resume the execution on migrated data. The PN table will be allocated in an RDMA memory region which can be read by all nodes. Each node will lazily update its PN table until it watches that the partition has become *Executable* (**E**) at the destination node. For example, when a transaction on NX touches P0 whose state in PN table is *Non-executable* (**N**), NX will suspend the transaction and continuously probe N1's PN table until the state of P0 become *Executable* (**E**). Then NX will update its PN table to avoid further watching and resume the transaction.

ALGORITHM 1: Generate a reconfiguration plan.

```
Data: $L_N$: an array of the workload for each node.
      $L_P$: an array of the workload for each partition.
      $avg\_load_N$: the average load per node.
      $P$: a list of all partitions.
      $T_{PN}$: a mapping table from partitions to host nodes.
      ($L_N$, $L_P$ and $avg\_load_N$ are provided by the monitor.)
1:  Function GENERATE_PLAN
2:      $new\_plan \leftarrow \{\}$
3:      Sort $P$ by the descending order in $L_P$
4:      for $p$ in $P$ do
5:          $src$ = $primary$ in $T_{PN}[p]$
6:          if $L_N[src] > avg\_load_N$ then
7:              $dst = src,\ load = L_N[src]$
8:              for $backup$ in $T_{PN}[p]$ do
9:                  if $(L_N[backup] < avg\_load_N)$ &&
10:                    $(L_N[backup] < load)$ then
11:                     $dst = backup,\ load = L_N[dst]$
12:             if $dst\ != src$ then
13:                 $new\_plan \leftarrow (p, dst)$
14:                 $L_N[src]$ -= $L_P[p]$, $L_N[dst]$ += $L_P[p]$
15:     if HAS_BALANCED $(L_N)$ then
16:         return $new\_plan$
17:     ... // fine-grained planning of E-Store
```

5 Replication-aware Reconfiguration Plan

DrTM+B follows E-Store [24] to use a *two-phase monitoring* mechanism to detect load imbalance and identify the tuples causing it, as well as a greedy planning algorithm to generate the reconfiguration plan. To support the replication-driven live reconfiguration, DrTM+B extends the algorithm by considering the distribution of replicas.

The extended algorithm will first try to greedily balance the workload by migrating the partitions to the nodes that hold their replicas. As shown in Algorithms 1, the planner will start from the partition p with the highest workload $L_P[p]$ tracked by the monitor at runtime (line 3–4). If the node with the primary of partition p is overloaded (line 5–6), the planner will check whether the nodes with the backup of partition p are underloaded (line 7–11). If found, the node with the least workload will be designated as the destination of partition p in the reconfiguration plan (line 13). The workload of source and destination node will be updated (line 14). After iterating all partitions, the planner will return the new plan if the workload has been balanced (line 15–16). Otherwise, DrTM+B will follow the *fine-grained* planning algorithm in E-Store to refine the plan by splitting the partitions and/or creating new replicas, which requires the knowledge from *fine-grained* monitoring [24].

6 Evaluation

6.1 Experimental Setup

The performance evaluation was conducted on a small-scale cluster with 6 machines. Each machine has two 10-core Intel Xeon E5-2650 v3 processors with 128GB of DRAM and a ConnectX-3 MCX353A 56Gbps InfiniBand NIC via PCIe 3.0 x8 connected to a Mellanox IS5025 40Gbps InfiniBand Switch. All machines run Ubuntu 14.04 with Mellanox OFED v3.0-2.0.1 stack.

We implemented DrTM+B based on DrTM+R [5] where 3-way replication is enabled. Each machine dedicates one processor to run up to 8 worker threads and 2 auxiliary cleaner threads[5], the another processor is assigned to clients. To make an apple-to-apple comparisons, the state-of-the-art post-copy approach with all optimizations in Squall [12] was also implemented on DrTM+R as the baseline. In our experiments, we run all systems with 10s for warm-up and use a monitoring time window of 1s [24]. The backup replicas are randomly assigned to all nodes during database initialization.

We use two standard OLTP benchmarks to evaluate DrTM+B: TPC-C[26] and SmallBank [25]. *TPC-C* simulates principal transactions of an order-entry environment, which scales by partitioning a database into multiple warehouses spreading across multiple machines. We use a database with 192 warehouses (32 per node). The cross-warehouse ratio is set to 1% for the new-order transactions according to the specification. Similar to prior work [24], two skewed settings are evaluated. For low skew, the Zipfian distribution is used where two-thirds of transaction requests go to one-third of warehouses. For high skew, 40% of requests follow the Zipfian distribution used in low skew, while the remaining requests target four warehouses located initially on the first server.

SmallBank models a simple banking application where each transaction performs simple reads and writes on user accounts. SmallBank scales by partitioning user accounts into multiple partitions spreading across multiple machines. We use a database with 192 partitions (100K accounts for each). The default probability of cross-machine accesses is set to 1%. Again, two differ-

[5]DrTM+B follows DrTM+R to use auxiliary cleaner threads for applying updates in the log to the backup replicas periodically.

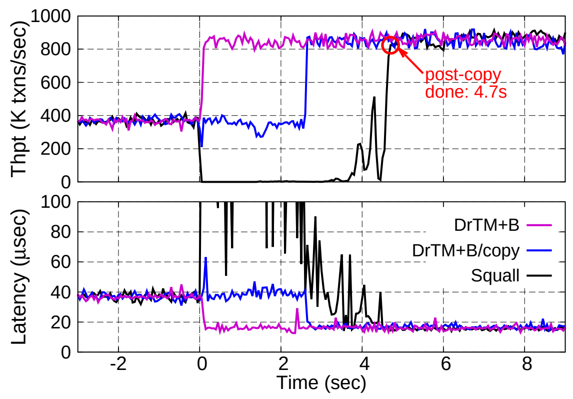

Fig. 11: *The perf. timeline for TPC-C with low skew.*

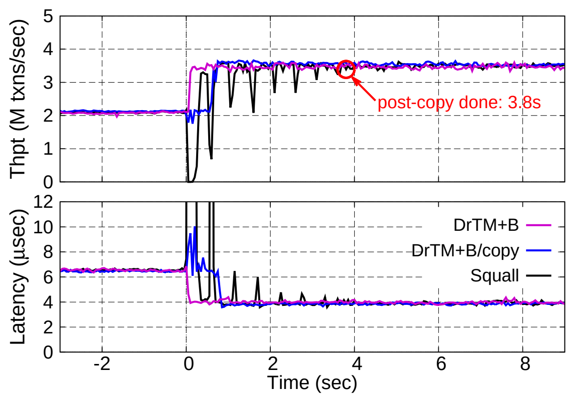

Fig. 13: *The perf. timeline for SmallBank with low skew.*

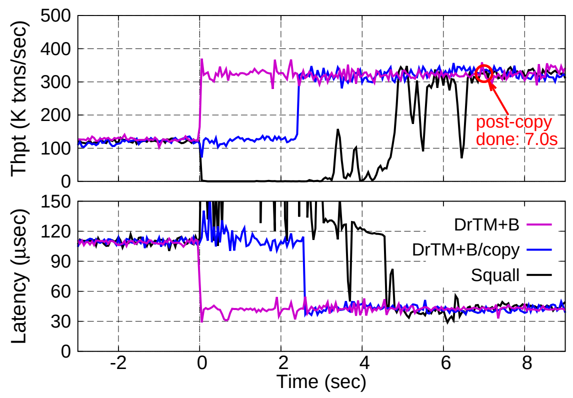

Fig. 12: *The perf. timeline for TPC-C with high skew.*

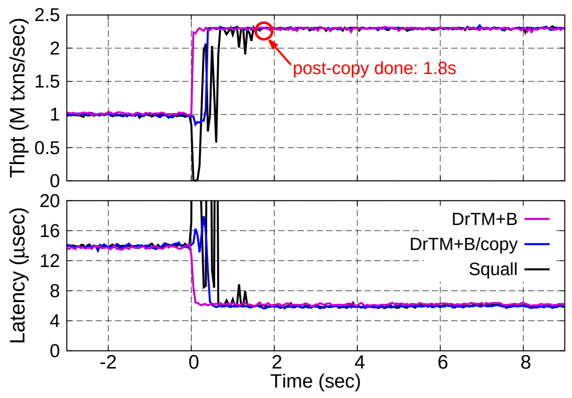

Fig. 14: *The perf. timeline for SmallBank with high skew.*

ent skewed settings are evaluated. For low skew, two-thirds of transaction requests go to 25% of records in the database. For high skew, 90% transaction requests go to 10% of records.

6.2 Performance for Skewed Workloads

We first evaluate the performance of DrTM+B and Squall for both benchmarks with various skewed workloads. The same initial configuration and reconfiguration plan are enforced to both systems, which ensures the same behaviors before and after live reconfiguration. In fact, for a 6-node cluster with 3-way replication, the existing backup is enough to provide load balance for both low and high skewed workloads. Therefore, DrTM+B can skip the data transfer. However, we still provide the performance of DrTM+B/copy as the reference, which enforces to migrate data with asynchronous replication to the destination node without regarding the existing data, while *parallel data fetching* and *cooperative commit protocol* are enabled by default.

TPC-C: Low skew. Fig. 11 shows the performance timeline of live reconfiguration with different approaches. After reconfiguration, the throughput increases by 2.1X, while the latency decreases by 33%. Since there is no data movement, DrTM+B has imperceptible downtime due to sub-millisecond commit phase. Squall needs 4.7s to finish reconfiguration due to *reactive* data transfer, while DrTM+B/copy only takes about 2.5s thanks to *parallel data fetching*. The performance degradation in DrTM+B/copy is also trivial, the average throughput drops by about 7% and the average latency increases by about 6% during live reconfiguration. This is because in DrTM+B/copy all logs are sent concurrently and data fetching mostly occupies CPU resources at *spare* nodes. For Squall, the throughput drops by around 99% and the latency increases by 7.7X in this period (the first 2.5s) due to frequent transaction aborts and increased contention on CPUs.

High skew. Fig. 12 shows how DrTM+B can improve the performance of TPC-C workload with high skew. After reconfiguration, the throughput increases by 3.0X, while the latency decreases by 64%. When there is no data movement, DrTM+B improves performance instantly. It takes about 7.0s for Squall, while DrTM+B/copy only needs 2.6s. Moreover the throughput of DrTM+B/copy decreases by about 2% during live reconfiguration and the latency increases by nearly 3%, while Squall suffers from 98% throughput degradation in this period since most workload is focused on hot spots.

SmallBank: Low skew. Fig. 13 shows the performance timeline of live reconfiguration with different approaches. After reconfiguration, the throughput in-

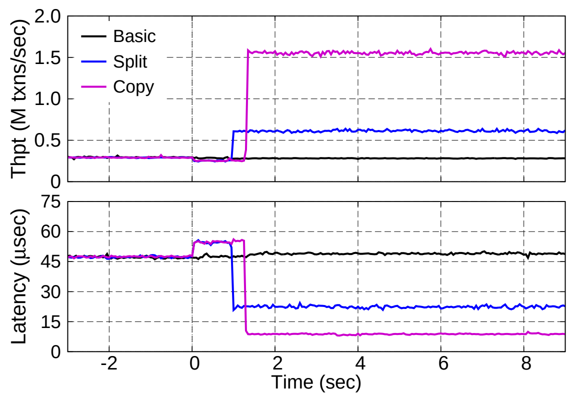

Fig. 15: *The perf. timeline for SmallBank with a load spike.*

creases by 1.7X, while the latency decreases by 42%. As existing backups are enough to balance, DrTM+B gains performance increase instantly. The reconfiguration by Squall requires 3.8s, while DrTM+B/copy only takes 0.7s to finish. Further, the throughput of DrTM+B/copy drops by only 3% and latency increases by 11% during live reconfiguration. By contrast, Squall has about 23% throughput drop and nearly 6.3X latency increase in this period. The throughput drops less for Squall in SmallBank compared with TPC-C since the transactions are much simpler and less affected by missing data.

High skew. Fig. 14 shows the performance timeline of SmallBank workload with high skew. After reconfiguration, the throughput increases by 2.4X and the latency decreases by 69%. Squall takes 1.8s to finishes while DrTM+B/copy only needs 0.3s. Moreover DrTM+B/copy has only 6% throughput drop with 8% latency increase without notable downtime, while the numbers for Squall are 22% and 3.8X respectively in this period. Although the performance is improving gradually during live reconfiguration, it still has non-negligible costs.

Load spike: We further evaluate DrTM+B with a load spike for SmallBank where 90% of workloads focus only on one partition (0.5% of records). In such workload data movement is necessary for the optimal plan. Fig. 15 shows the performance timeline of DrTM+B with different settings. Note that DrTM+B starts one second fine-grained monitoring after detecting the imbalance at time 0 to generate fine-grained plans for DrTM+B/split and DrTM+B/copy. This will tentatively increase latency by 17%. DrTM+B only exchanges primary with backup, which barely changes the performance since the hot data resides in one partition. DrTM+B/split uses *partition splitting* to balance the workload on the hot partition, which can immediately improve the throughput by 2.1X and reduce the latency by 54%. DrTM+B/copy further uses *asynchronous replication* to create new replicas on spare nodes, so that the throughput is improved by 5.4X

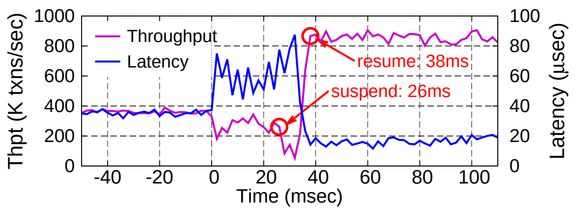

Fig. 16: *The focus timeline for TPC-C with low skew.*

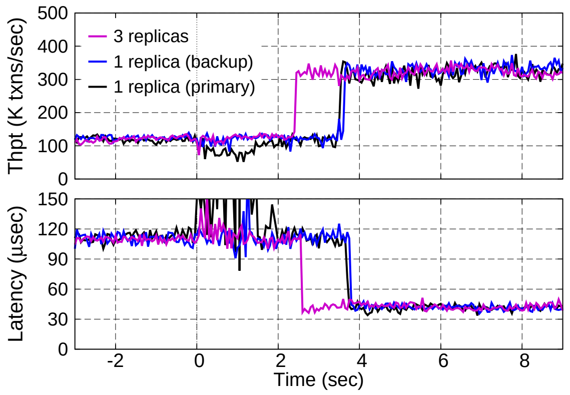

Fig. 17: *The perf. timeline using different replication copy mechanisms for TPC-C with high skew.*

while the latency is decreased by 82%. The improvement is slighted delayed (0.5s) due to data movement.

6.3 Breakdown of Commit Phase

Since the commit phase of the pre-copy approach will cause service downtime, we further study the time breakdown of DrTM+B's commit phase using TPC-C workload with low skew and illustrate the focus timeline in Fig. 16. RM issues a new plan at 0ms, suspends all nodes at 26ms, and resumes the execution at 38ms. In DrTM+B, all logs are concurrently drained during the commit phase, therefore the waiting time to activate the destination partitions is much short. Moreover, the downtime of DrTM+B is minimized to only 12ms thanks to the *cooperative commit protocol*. The throughput drops slightly by 12% during the commit phase.

6.4 Optimization for Replication Copy

To further study the impact of different optimizations on the pre-copy phase, we conduct an experiment on balancing TPC-C workload with high skew. Fig. 17 shows the performance timeline of live reconfiguration with three different settings in DrTM+B, which indicate fetching data directly from the primary, 1 backup replica, or all of 3 replicas in parallel. When fetching directly from the primary, it incurs 24% throughput drop and 2.8X latency increase due to the contention on CPU at the primary. Data fetching from 1 backup replica has nearly no impact to throughput, and the latency only increases 3%. This is because the backup node has less load than that of the primary. Moreover, data fetching from 3 replicas in parallel can notably shorten the migration time from

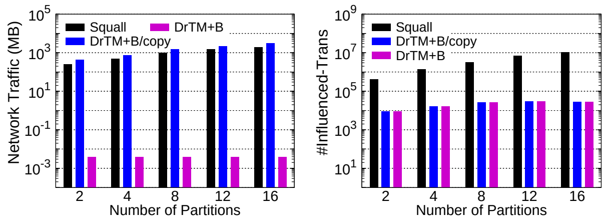

Fig. 18: *A comparison of (a) network traffic and (b) performance degradation between DrTM+B and Squall.*

3.5 to 2.6 seconds.

6.5 Micro-benchmark

We further use a micro-benchmark based on TPC-C to study the performance of DrTM+B and Squall, where 24 partitions are deployed on 3 nodes each holds one warehouse. We measure the metrics by swapping different numbers of partitions between first two nodes.

Network traffics. Fig. 18(a) shows the log scale data transferred with the increase of swapping partitions. For DrTM+B, only 4KB metadata is transferred during the commit phase since there is no data movement. For Squall, the network traffic is mainly dominated by the size of migrated partitions. For DrTM+B/copy, the data transferred is nearly doubled due to additional logs of the new backup replica, which can be avoided by removing one existing backup.

Influenced transactions. Fig. 18(b) presents the log scale influenced transactions with the increase of swapping partitions. DrTM+B and DrTM+B/copy have almost the same amount of influenced transactions, which only happens during the commit phase. The number of influenced transactions in DrTM+B is only 2% of Squall when swapping two partitions.

7 Related Work

Live reconfiguration for shared-storage database: There have been a few efforts to provide live reconfiguration features to shared-storage databases. For example, Albatross [9, 10] uses a traditional pre-copy approach to iteratively migrating the database. DrTM+B also uses a pre-copy approach but overcomes several limits through a novel reuse of fault-tolerant replicas. Zephyr [13] uses post-copy to migrate database while allowing transactions to execute during data migration. ProRea [20] extends Zephyr's approach by proactively migrating hot tuples to reduce service interruption. Elastras [8] decouples data storage nodes from transaction nodes and provides elasticity by moving or adding partitions. Slacker [4] leverages existing database backup tools to migrate data, and uses stop-and-copy or pre-copy method to create backups.

Live reconfiguration for partitioned databases: The importance of providing load balance has stimulated a few recent designs targeting partitioned databases. Squall [12] follows Zephyr's post-copy mechanism to performing live reconfiguration with the support of fine-grained tuple level migration. Compared to Zephyr, it introduces some optimizations such as pull prefetching and range splitting. In this paper, we show that using a post-copy based approach cause notable service disruption for fast in-memory transaction processing. DrTM+B makes a novel reuse and extension of the replication mechanism to achieve fast and seamless reconfiguration.

Providing elasticity on non-transactional systems: Spore [14] aims at addressing skewed access patterns on keys for in-memory caching systems (i.e., memcached). The basic approach is to create replicas of popular keys, which is similar to the asynchronous replication in DrTM+B. However, it does not consider the transactional execution on the key/value store. The reconfiguration in primary-backup systems has also been studied in distributed systems [28, 23]. For example, Shraer et al. [23] proposes a protocol to dynamically configure Apache Zookeeper without leveraging an extern reconfiguration service. Similarly, The commit protocol in DrTM+B incurs no downtime to partitions where their primaries have not changed during reconfiguration.

Generating reconfiguration plan: E-Store [24] proposes a fine-grained tracking approach and can generate a new partition plan to migrate data tuples between partitions. DrTM+B further extends it to consider the location of existing fault-tolerant replicas. Accordion [21] uses a mechanism to find a better partition plan to reduce distributed transactions. A few recent work has provided a general partitioning service for datacenter applications [2] or general DBMS schema [22].

8 Conclusion

This paper described DrTM+B, a fast and seamless live reconfiguration framework for fast in-memory transaction systems. It adopted a pre-copy based approach, which allows minimal interference between live reconfiguration and normal execution. DrTM+B further made a novel reuse of replication for fault tolerance in several ways to accelerate data transfer in pre-copy phase and minimize the downtime in commit phase. Evaluations using typical OLTP workloads with different skewed workloads confirmed the benefits of designs in DrTM+B.

Acknowledgments

We thank our shepherd Eddie Kohler and the anonymous reviewers for their insightful suggestions. This work is supported in part by the National Key Research & Development Program (No. 2016YFB1000500), the National Natural Science Foundation of China (No. 61402284, 61572314, 61525204), the National Youth Top-notch Talent Support Program of China, and Singapore NRF (CREATE E2S2).

References

[1] Daily Deals and Flash Sales: All the Stats You Need to Know. http://socialmarketingfella.com/daily-deals-flash-sales-stats-need-know/, 2016.

[2] A. Adya, D. Myers, J. Howell, J. Elson, C. Meek, V. Khemani, S. Fulger, P. Gu, L. Bhuvanagiri, J. Hunter, R. Peon, L. Kai, A. Shraer, A. Merchant, and K. Lev-Ari. Slicer: Auto-sharding for Datacenter Applications. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 739–753, Berkeley, CA, USA, 2016. USENIX Association.

[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'12, pages 53–64, New York, NY, USA, 2012. ACM.

[4] S. Barker, Y. Chi, H. J. Moon, H. Hacigümüş, and P. Shenoy. "Cut Me Some Slack": Latency-aware Live Migration for Databases. In *Proceedings of the International Conference on Extending Database Technology*, EDBT'12, pages 432–443, New York, NY, USA, 2012. ACM.

[5] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen. Fast and General Distributed Transactions using RDMA and HTM. In *Proceedings of the European Conference on Computer Systems*, EuroSys'16, page 26. ACM, 2016.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the USENIX Conference on Symposium on Networked Systems Design & Implementation*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[7] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: A Workload-driven Approach to Database Replication and Partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57, Sept. 2010.

[8] S. Das, D. Agrawal, and A. El Abbadi. ElasTraS: An Elastic, Scalable, and Self-managing Transactional Database for the Cloud. *ACM Trans. Database Syst.*, 38(1):5:1–5:45, Apr. 2013.

[9] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi. Live Database Migration for Elasticity in a Multitenant Database for Cloud Platforms. *CS, UCSB, Santa Barbara, CA, USA, Tech. Rep*, 9:2010, 2010.

[10] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi. Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud Using Live Data Migration. *Proc. VLDB Endow.*, 4(8):494–505, May 2011.

[11] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15, pages 54–70, New York, NY, USA, 2015. ACM.

[12] A. J. Elmore, V. Arora, R. Taft, A. Pavlo, D. Agrawal, and A. El Abbadi. Squall: Fine-Grained Live Reconfiguration for Partitioned Main Memory Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'15, pages 299–313, New York, NY, USA, 2015. ACM.

[13] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'11, pages 301–312, New York, NY, USA, 2011. ACM.

[14] Y.-J. Hong and M. Thottethodi. Understanding and mitigating the impact of load imbalance in the memory caching tier. In *Proceedings of the Annual Symposium on Cloud Computing*, SOCC'13, pages 13:1–13:17, New York, NY, USA, 2013. ACM.

[15] A. Kalia, M. Kaminsky, and D. G. Andersen. FaSST: fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 185–201. USENIX Association, 2016.

[16] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-Store: A High-performance, Distributed Main Memory Transaction Processing System. *Proc. VLDB Endow.*, 1(2):1496–1499, Aug. 2008.

[17] A. Khandelwal, R. Agarwal, and I. Stoica. BlowFish: Dynamic Storage-Performance Tradeoff in Data Stores. In *13th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'16, pages 485–500, Santa Clara, CA, Mar. 2016. USENIX Association.

[18] L. Lamport, D. Malkhi, and L. Zhou. Vertical Paxos and Primary-backup Replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC'09, pages 312–313, New York, NY, USA, 2009. ACM.

[19] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware Automatic Database Partitioning in Shared-nothing, Parallel OLTP Systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD'12, pages 61–72, New York, NY, USA, 2012. ACM.

[20] O. Schiller, N. Cipriani, and B. Mitschang. ProRea: Live Database Migration for Multi-tenant RDBMS with Snapshot Isolation. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT'13, pages 53–64, New York, NY, USA, 2013. ACM.

[21] M. Serafini, E. Mansour, A. Aboulnaga, K. Salem, T. Rafiq, and U. F. Minhas. Accordion: Elastic Scalability for Database Systems Supporting Distributed Transactions. *Proc. VLDB Endow.*, 7(12):1035–1046, Aug. 2014.

[22] M. Serafini, R. Taft, A. J. Elmore, A. Pavlo, A. Aboulnaga, and M. Stonebraker. Clay: Fine-grained Adaptive Partitioning for General Database Schemas. *Proc. VLDB Endow.*, 10(4):445–456, Nov. 2016.

[23] A. Shraer, B. Reed, D. Malkhi, and F. P. Junqueira. Dynamic Reconfiguration of Primary/Backup Clusters. In *Proceedings of the USENIX Annual Technical Conference*, USENIX ATC'12, pages 425–437, 2012.

[24] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker. E-Store: Fine-grained Elastic Partitioning for Distributed Transaction Processing Systems. *Proc. VLDB Endow.*, 8(3):245–256, Nov. 2014.

[25] The H-Store Team. SmallBank Benchmark. `http://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/`.

[26] The Transaction Processing Council. TPC-C Benchmark V5.11. `http://www.tpc.org/tpcc/`.

[27] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy Transactions in Multicore In-memory Databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP'13, pages 18–32. ACM, 2013.

[28] R. Van Renesse and F. B. Schneider. Chain Replication for Supporting High Throughput and Availability. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation*, volume 4 of *OSDI'04*, pages 91–104, 2004.

[29] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast In-memory Transaction Processing Using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15, pages 87–104, New York, NY, USA, 2015. ACM.

[30] W. Zheng, S. Tu, E. Kohler, and B. Liskov. Fast Databases with Fast Durability and Recovery Through Multicore Parallelism. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 465–477. USENIX Association, 2014.