Fast and General Distributed Transactions using RDMA and HTM

Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, Haibo Chen

Shanghai Key Laboratory of Scalable Computing and Systems
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Contacts: {rongchen, haibochen}@sjtu.edu.cn

Abstract

Recent transaction processing systems attempt to leverage advanced hardware features like RDMA and HTM to significantly boost performance, which, however, pose several limitations like requiring priori knowledge of read/write sets of transactions and providing no availability support. In this paper, we present DrTM+R, a fast in-memory transaction processing system that retains the performance benefit from advanced hardware features, while supporting general transactional workloads and high availability through replication. DrTM+R addresses the generality issue by designing a hybrid OCC and locking scheme, which leverages the strong atomicity of HTM and the strong consistency of RDMA to preserve strict serializability with high performance. To resolve the race condition between the immediate visibility of records updated by HTM transactions and the unready replication of such records, DrTM+R leverages an optimistic replication scheme that uses *seqlock*-like versioning to distinguish the visibility of tuples and the readiness of record replication. Evaluation using typical OLTP workloads like TPC-C and SmallBank shows that DrTM+R scales well on a 6-node cluster and achieves over 5.69 and 94 million transactions per second without replication for TPC-C and SmallBank respectively. Enabling 3-way replication on DrTM+R only incurs at most 41% overhead before reaching network bottleneck, and is still an order-of-magnitude faster than a state-of-the-art distributed transaction system (Calvin).

1. Introduction

Transaction [18] is a very powerful abstraction that simplifies the processing of relational data. With the increase of data volume and concurrency level, many systems like Web service, stock exchange, and e-commerce demand the support of low-latency and high-throughput transaction processing systems. However, traditional systems are usually with low efficiency such that only a small portion of the wall-clock time is spent on useful data processing [44].

Recent advanced hardware features like large (non-volatile) memory, hardware transactional memory (HTM) and fast interconnect with RDMA pose new opportunities for fast transaction processing: large memory volume enables a new paradigm of in-memory transactions, which significantly reduces buffering and I/O overhead; the hardware support for atomicity, consistency, and isolation (ACI) properties makes it very promising to offload concurrency control to CPU [39, 52]; and the RDMA feature further enables fast distributed transactions within a local cluster [17, 54]. However, while prior systems have demonstrated the feasibility of combining such advanced hardware features for fast distributed transactions, they fall short in several aspects. They either leverage only parts of the features [17, 39, 52], or place several restrictions on transaction such as knowing read/write sets in advance and providing no availability support [54], or both.

This paper presents DrTM+R, a fast and general distributed transaction processing system. Like prior systems, DrTM+R supports in-memory transactions by leveraging battery-backed memory as the main storage for database records and combines HTM and RDMA for fast distributed transactions. Unlike prior systems [54], DrTM+R places no restrictions on transactional workloads and provides full replication support for high availability.

To address the generality issue, DrTM+R leverages an opportunistic concurrency control (OCC) design [52] for local transactions, while leveraging HTM to protect the validation and write phases. To glue together distributed transactions across machines, DrTM+R leverages the strong consistency of RDMA to detect the conflict between a remote (distributed) transaction from local transactions. To prevent a remote transaction from updating a record in the read set of a local transaction, DrTM+R additionally introduces a remote locking phase before the validation phase and the write

EuroSys '16, April 18-21, 2016, London, United Kingdom

DOI: http://dx.doi.org/10.1145/2901318.2901349

phase of OCC. As DrTM+R knows all read and write sets of a transaction after the execution phase, DrTM+R does not require priori knowledge of read/write sets before transaction execution.

There is a challenge in providing replication for DrTM+R, due to the fact that no I/O operations such as RDMA operations are allowed within an HTM transaction. Hence, DrTM+R cannot replicate local updates within a transaction to a remote machine, while replicating updates outside the HTM transaction would cause a race condition that a transaction is considered as committed but not replicated. DrTM+R addresses this issue by leveraging an optimistic replication scheme that decouples local commit from replication (i.e., full commit). Specifically, it assigns a version number to each record and uses a seqlock[1]-like versioning scheme. DrTM+R increases the version number into "odd" within the HTM transaction, indicating that the records are committed but not replicated. It then increases the records again to "even" after replication outside the HTM transaction to indicate that the records have been replicated. An inflight transaction is allowed to read an unreplicated record but cannot commit until the records have been replicated.

We have implemented DrTM+R, which extends a prior OCC-based multicore database design [52] with the support for distributed transactions. To demonstrate the efficiency of DrTM+R, we have conducted a set of evaluations of DrTM+R's performance using a 6-node cluster connected by InfiniBand NICs with the RDMA feature. Each machine of the cluster has two 10-core RTM-enabled Intel Xeon processors. Using popular OLTP workloads like TPC-C [46] and SmallBank [45], we show that DrTM+R can perform over 5.69 and 94 million transactions per second without replication for TPC-C and SmallBank respectively. A simulation of running multiple logical nodes over each machine shows that DrTM+R may be able to scale out to a larger cluster with tens of nodes. A comparison with a state-of-the-art distributed transaction system (i.e., Calvin without replication) shows that DrTM+R is at least 26.8X faster for TPC-C. We further show that enabling 3-way replication on DrTM+R only incurs at most 41% overhead before reaching network bottleneck.

In summary, the contributions of this paper are:

- The design and implementation of a distributed transaction processing system with the combination of HTM and RDMA (§3), yet without restrictions in prior work like knowing read/write sets of transactions in advance.
- An HTM/RDMA friendly concurrency control scheme using OCC and remote locking to glue together multiple concurrent transactions across machines (§4).
- An efficient optimistic replication scheme that provides durability and high availability while retaining the benefits of combining RDMA and HTM (§5).
- A set of evaluations that confirm extremely high performance and high availability of DrTM+R with strict serializability (§7).

2. Background and Motivation

2.1 Advanced Hardware Features

HTM and Strong Atomicity. Hardware transactional memory (HTM) has recently been commercially available in the form of Intel's restricted transactional memory (RTM). The goal of HTM is to be an alternative of locking, by providing the simplicity of coarse-grained locking yet having the performance of fine-grained locking. By enclosing a set of operations (including memory operations) as a hardware transaction[2], the CPU ensures that a set of operations execute with the properties of atomicity, consistency and isolation (ACI). Some recent hardware proposals even further propose adding durability to HTM [53]. Intel RTM provides a set of interfaces including `XBEGIN`, `XEND`, and `XABORT`, which will begin, end, and abort an RTM transaction accordingly.

As a hardware supported one, Intel's RTM provides strong atomicity [7] within a single machine. This means that a non-transactional operation will unconditionally abort an HTM transaction when their accesses conflict. To simplify hardware implementation, RTM uses the first-level cache to track the write set and an implementation-specific structure (e.g., a bloom filter) to track the read set. It relies on the cache coherence protocol to detect conflicts, upon which at least one transaction will be aborted.

There are also several limitations with Intel's RTM [51, 52], which prevents a direct use of RTM to protect database transactions. The first limitation is that an RTM transaction can only track limited read/write sets. Consequently, the abort rate of an RTM transaction will increase significantly with the increase of working set size. Thus, running a program with a number of memory operations will cause high abort rate or even no forward progress. The second limitation is that no I/O operations are allowed within an RTM transaction. This prevents us from running RDMA operations within an RTM transaction for remote memory operations. Last but not least, RTM is only a compelling hardware feature for single machine platform but provides no support across machines.

RDMA and Strong Consistency. Remote Direct Memory Access (RDMA) is a fast cross-machine memory access technique commonly seen in high-performance computing area. Due to completely bypassing target operating systems and/or CPU, it has high speed, low latency, and

[1] Seqlock, i.e., sequential lock, is special locking scheme used in Linux kernel that allows fast writes to a shared memory location among multiple racy accesses.

[2] This paper uses HTM/RTM transaction to denote the transactional code executed under HTM's protection, and uses transaction to denote the original user-written transaction.

	Silo [49, 60]	DBX [52]	Calvin [48]	FaRM [16, 17]	DrTM [54]	DrTM+R
Performance	High	High	Low	High	Very High	Very High
Scale-out	No	No	Yes	Yes	Yes	Yes
Availability	No	No	Yes	Yes	No	Yes
Priori Knowledge	None	None	Read/Write Sets	None	Read/Write Sets	None
Hardware Features	None	HTM	None	RDMA	HTM/RDMA	HTM/RDMA

Table 1. A comparison of various in-memory transaction systems.

low CPU overhead. Generally, RDMA-capable NICs provide three commutation patterns, including IPoIB that emulates IP over InfiniBand, SEND/RECV Verbs that provide message exchanges in user-space with kernel bypassing, and one-sided RDMA that further bypasses target CPU. Prior work has shown that one-sided RDMA provides the highest performance among other alternatives [16, 21, 34, 36]. One interesting feature of RDMA operations is its strong consistency with respect to the memory operations in the target CPU: an RDMA operation is cache coherent with the memory operations [54]. Thus, when combining with the strong consistency of HTM, an RDMA operation will unconditionally abort a conflicting HTM transaction in the target machine. However, while powerful, one-sided RDMA operation only provides limited interfaces: read, write, and two atomic operations (*fetch-and-add* and *compare-and-swap*).

Non-volatile Memory: Some data-centers provide an interesting feature called distributed UPS that allows flushing data from volatile memory to persistent storage (like SSD) [33]. Besides, NVDIMM [43] has been commercially available by major memory vendors like Micron, Viking, JEDEC, and Fusion-IO. Finally, 3DXpoint [50], a new non-volatile memory technology from Intel and Micron, has been predicted to be available to the market soon. With logging to disk accounting for a large portion of transaction execution, these non-volatile memory devices can largely mitigate the logging overhead of transactions.

2.2 Issues with Prior Systems

While HTM and RDMA are promising hardware features to boost transaction executions, prior systems either only leverage one of them, or fall short in only providing limited transaction features. Table 1 illustrates a comparison of recent designs with DrTM+R.

General designs: Silo [49, 60] is a fast in-memory transaction for multicore. It adopts fine-grained locking with scalable transaction ID to provide scalable transaction processing in a single machine. However, it is not designed to be scale-out to multiple machines. Calvin [48] leverages deterministic execution for scalable distributed transaction processing. Yet, its performance is at least an order of magnitude less than that of DrTM+R, due to not exploiting advanced hardware features like HTM and RDMA. Besides, Calvin requires priori knowledge of remote read and write sets, which would limit its applicability to various types of transactions.

RDMA-only designs: FaRM [16] and its successor [17] leverage RDMA to provide general distributed transactions. It abstracts a cluster as a partitioned global address space and leverages RDMA to fetch remote database records to be processed locally. It leverages the low-latency feature of RDMA operations to implement an optimized four-phase commit with replication, and can recover a failure in less than 50ms. In contrast, DrTM+R further combines HTM with RDMA to process distributed transactions and may have better performance under the same setting.

HTM-only designs: DBX [52] combines HTM with an OCC protocol to implement efficient transactions on multi-core machines. To reduce the working set of an HTM transaction, it separates execution from commit by only leveraging HTM transactions to protect the validation and write phases. This effectively reduces the working set of an HTM transaction from all data to all metadata of database records accessed in a database transaction. DBX-TC [39] implements an optimized transaction chopping algorithm to decompose a set of large transactions into smaller pieces, and thus only uses HTM transactions to protect each piece. Due to mostly offloading concurrency control to HTM, it shows better performance than DBX. However, both of them are limited to single-machine transactions.

HTM/RDMA designs: DrTM [54] is the closest work with DrTM+R. Like DrTM, DrTM+R also leverages HTM and RDMA to provide efficient transaction processing. Unlike DrTM, DrTM+R places no restrictions on transaction features such that it requires no priori knowledge of transaction working set and provides efficient replication for high availability, two important features that are missing in DrTM. Thus, unlike DrTM which combines two-phase locking (2PL) with HTM, DrTM+R provides a hybrid concurrency control protocol that combines optimistic concurrency control (OCC) with remote locking.

3. Overview

Setting. DrTM+R assumes a modern cluster that is connected with high-speed, low-latency network with RDMA features. Each processor in the cluster is equipped with HTM and each machine contains a portion of battery-backed non-volatile memory for data and logging. DrTM+R targets OLTP workloads over a large volume of data; it scales by partitioning data into a large number of shards across multiple machines. DrTM+R employs a worker-thread model by

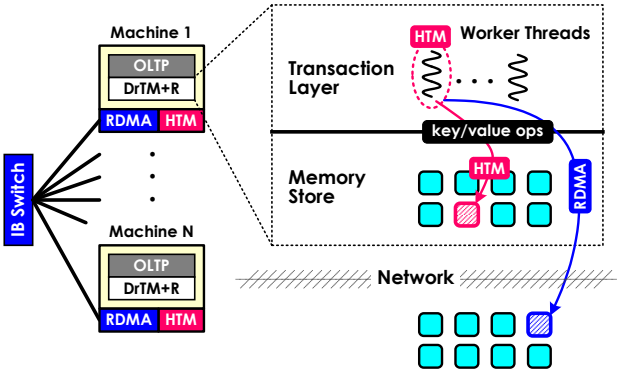

Figure 1. The architecture overview of DrTM+R.

running n worker threads atop n cores; each worker thread executes and commits a transaction at a time.

Approach Overview. Like prior work [52, 54], DrTM+R comprises of two independent components: transaction layer and memory store, as shown in Figure 1. DrTM+R leverages a hybrid OCC and locking scheme using HTM and RDMA (§4). On each machine, DrTM+R utilizes an OCC-based scheme to provide transaction support. Like typical OCC, DrTM+R separates execution from commit by first tracking the read/write sets of a transaction in the execution phase, validating the read set in the validation phase and finally committing the updates in the commit phase. The last two phases require atomicity, which DrTM+R guarantees using HTM transactions. DrTM+R exposes a partitioned global address space such that remote records and local records are explicitly distinguished using their address identifiers. DrTM+R follows a local execution model. To access remote data in a transaction, DrTM+R first fetches remote records into the hosting machine, makes necessary updates and then sends the records back to the remote machine. Remote accesses in DrTM+R are mainly done using one-sided RDMA operations for efficiency. As there is no apparent way to guarantee the atomicity between accessing a record in a remote machine and validating the record in the local machine, DrTM+R introduces a remote locking phase (§4.4) before the validation and commit phases, which leverages one-sided RDMA operations to lock remote records.

The high availability of DrTM+R is guaranteed by efficient replication of database records before fully committing a transaction (§5). This is achieved by leveraging a revised commit protocol (§5.1). DrTM+R leverages ZooKeeper [20] to reach an agreement on the current configuration among surviving machines. Inspired by FaRM [17], an RDMA-base protocol is used to manage leases, detect failures, and coordinate recovery(§5.2). Thanks to the fast interconnect, DrTM+R can detect a failure in a very short time with high accuracy. During recovery, DrTM+R first reconfigures the cluster and then recovers the state of crashed machines by leveraging surviving machines.

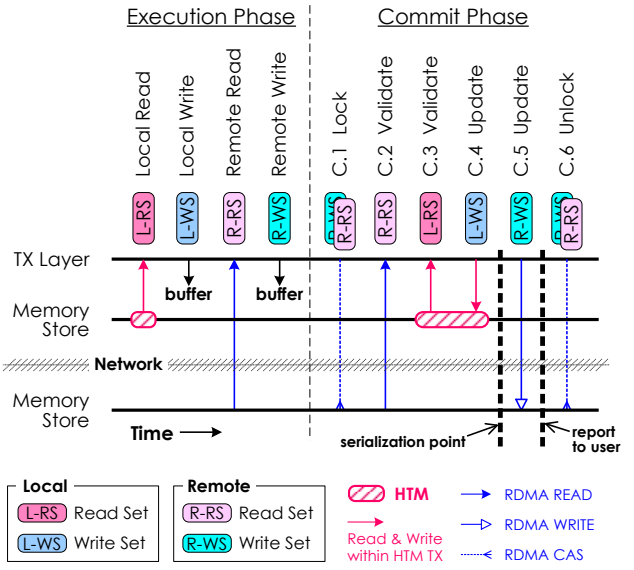

Figure 2. The concurrency control protocol in DrTM+R.

4. Supporting Distributed Transactions

DrTM+R uses an HTM-friendly optimistic concurrency control (OCC) protocol to provide transaction support within a single machine and further adopts an RDMA-friendly optimistic concurrency control (OCC) protocol to coordinate accesses to remote records for distributed transactions.

4.1 HTM/RDMA-friendly OCC Protocol

Since an HTM transaction provides strong atomicity and one-sided RDMA operations are cache-coherent, the prior system [54] leverages them to bridge the HTM and two-phase locking (2PL) protocol for distributed transactions. However, it requires priori knowledge of read/write sets of transactions for proper locking to implement the 2PL-like protocol. Unfortunately, this is not always the case for some general transaction workloads, like TPC-C [46][3] and TPC-E [47], which have dependent transactions.

DrTM+R addresses this limitation by designing a hybrid OCC and locking protocol to provide strictly serializable transactions. The main observation is that the read-/write of an transaction will be known after the execution phase in OCC, due to its separation of execution from commit. Hence, DrTM+R has all read/write sets known after the execution phase. However, any RDMA operation inside an HTM transaction will unconditionally cause an HTM abort and thus we cannot directly access remote records through RDMA within an HTM transaction. To this end, DrTM+R adjusts the traditional OCC protocol by distinguishing operations on local and remote records, which will be protected using HTM and RDMA-based locking mechanisms respectively.

[3] DrTM leverages transaction chopping to address this issue for TPC-C

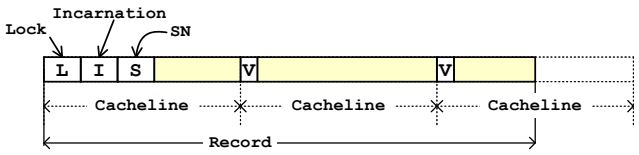

Figure 3. The structure of a record across three cachelines.

As shown in Figure 2, DrTM+R organizes the concurrency protocol into two phases: execution and commit. In the execution phase, DrTM+R provides different interfaces to run transaction code with read and write accesses to local and remote records. For read accesses, DrTM+R ensures consistent accesses to local and remote records by HTM and RDMA-based versioning respectively, and maintains the locations (i.e., virtual address or RDMA address) and versions (i.e., sequence number) of the records in local and remote read sets (i.e., `L_RS` and `R_RS`). For write accesses, DrTM+R buffers all updates locally for both local and remote records, and maintains the location and the buffer of the records in local and remote write sets (i.e., `L_WS` and `R_WS`) (§4.3). In the commit phase, DrTM+R attempts to atomically commit the transaction using HTM for local records and RDMA-based locking for remote records. DrTM+R follows the traditional OCC protocol to first validate that any record in read set is not changed, and then updates all local buffers in write set to actual records (§4.4).

Since DrTM+R uses different mechanisms that protect the accesses and commits of local records by HTM and remote records by RDMA-based versioning and locking, DrTM+R cannot simply combine the HTM-based OCC protocol from an HTM-friendly local database (e.g, DBX [52]) and RDMA-based OCC protocol (e.g., FaRM [17]). For example, a conflicting local read within HTM may still read an *inconsistent* record, since the update from a remote transaction using one-sided RDMA WRITE is only cache-coherent within a cache line. To this end, DrTM+R must carefully cooperate operations on different types of records.

4.2 Data Structure of Records

Based on the general key-value store interface provided by the memory store layer, the transaction layer implements typed database tables, namely collections of records. To facilitate our hybrid protocol, the transaction layer further encodes the following metadata to each record, as shown in Figure 3. Note that HTM tracks reads and writes at the granularity of a cache line, DrTM+R enforces each record starts at a new cache line, avoiding unnecessary HTM aborts due to false sharing.

Lock (64-bit): locks the record by distributed transactions on remote machines. It is used to ensure the isolation of transactions during the commit phase.

Incarnation (64-bit): tracks the number of frees at the key-value entry. It is used to detect whether the record has been freed or not during the commit phase.

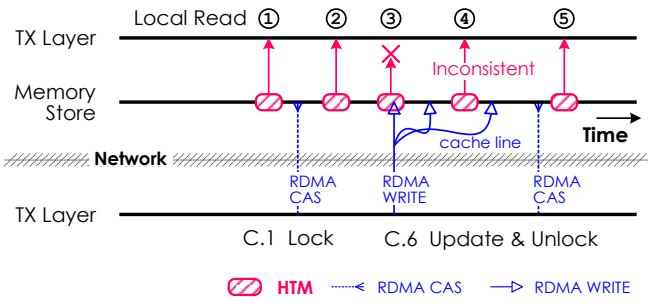

Figure 4. The consistency of local read.

Sequence Number (64-bit): tracks the number of updates on the record. It is used to detect read-write conflict during the commit phase.

Version (16-bit): identifies the version (the low-order bits of sequence number) of data in each cache line. It is used to check the consistency of a remote access across multiple cache lines during the execution phase.

4.3 Execution Phase

In the execution phase of the hybrid protocol, DrTM+R only needs to ensure consistent read accesses to records. All write operations will be stored in a local private buffer.

For local read accesses, inspired by DBX [52], DrTM+R uses an HTM transaction to guarantee the consistency of a local read, as long as the update to the record is also protected by an HTM transaction. However, the update from a remote machine may cause inconsistent local read since one RDMA WRITE to a record larger than one cache line size will result in separate writes on multiple cache lines. As shown in Figure 4, the results of the first, second, and fifth local reads are consistent, and the RDMA WRITE can correctly abort the third conflicting local read. However, if the local read runs between updates on multiple cache lines by the RDMA WRITE, the result will be inconsistent (e.g., the fourth read to a record crossing three cache lines, two updated and one dated).

Fortunately, we observe that a remote transaction always needs to lock the records before updating. Therefore, DrTM+R can simply check the lock field of record before reading to ensure consistency. If the record is locked, a local read can manually abort its HTM transaction, which will retry with a randomized interval until the record is unlocked before entering into a fallback handler. This avoids complicated mechanisms for consistent read across multiple cache lines, such as versioning [16] or checksum [34]. However, this also means that a locked record cannot be read by a local transaction even the value is consistent. For example, the second local read in Figure 4 will abort and retry. Note that it will not affect the correctness but only incur some *necessary* false aborts. The false abort is necessary because the locked record is likely to be updated by the remote transaction soon, the local transaction is inevitable to abort when validating its local read set during the commit phase, even if

```
LOCAL_READ(key)
  foreach <_key, _value> in local_writeset
    if (_key == key)  return _value
  XBEGIN() //HTM TX begin                    HTM Transaction
  if rec[key].lock == LOCKED //remote commit
    XABORT() //HTM TX abort, then retry
  _value = rec[key].value
  local_readset.add(key, rec[key].SN)
  XEND()   //HTM TX end
  return _value

LOCAL_WRITE(key, value)
  local_writeset.add(key, value)
```

Figure 5. The pseudo-code of local read and write in DrTM+R.

```
REMOTE_READ(key)
  foreach <_key, _value> in remote_writeset
    if (_key == key)  return _value
L:rec = RDMA_READ(key)
  if !MATCHING_VERSIONS(rec)
    goto L //RETRY: inconsistent read
  remote_readset.add(key, rec.SN)
  return rec.value

REMOTE_WRITE(key, value)
  remote_writeset.add(key, value)
```

Figure 6. the pseudo-code of remote read and write in DrTM+R.

it reads a consistent but stale record in the execution phase. Figure 5 illustrates the pseudo-code of local read and write in DrTM+R.

For remote accesses, inspired by FaRM [16], DrTM+R uses *versioning* to implement a lock-free and consistent read. It mainly relies on the strong consistency provided by one-sided RDMA READ, which ensures atomic read of the record written by a local write or a one-sided RDMA WRITE. However, since it only works in a single cache line, DrTM+R places the version of record at the start of each cache line (except the first), as shown in Figure 3 and requires a record write to update each version. Further, the remote read requires to match all versions of the record for the consistency of value. To save space, DrTM+R uses the least significant 16 bits of the sequence number as the version, which is usually sufficient to avoid overflow within a single remote read [16]. Note that the versions of the record are invisible to the user-written transactions. Figure 6 illustrates the pseudo-code of remote read and write in DrTM+R. Different from the versioning in FaRM [16], DrTM+R does not check the lock field of a record, because the record may be locked by a distributed transaction on a remote machine even only for read during the commit phase (see §4.4). However, because DrTM+R will lock records in the remote read set during the commit phase, uncommitted read to a remote record in the execution phase can be detected, and the transaction can safely abort to ensure strict serializability.

Table 2 summarizes different mechanisms used by DrTM+R to ensure consistent read accesses to local and remote records in the execution phase. DrTM+R leverages a combination of HTM and versioning to guarantee the consistency between local/remote reads/writes.

CONSISTENCY	COMMIT/L	COMMIT/R
READ/L	HTM	HTM†
READ/R	Versioning	Versioning

Table 2. A summary of different mechanisms used to ensure *consistency* of local and remote reads. **L** and **R** stand for Local and Remote records. (†) READ/R needs to check the lock within HTM to avoid inconsistent read.

Finally, the insert/delete operations in DrTM+R will be shipped to the host machine using SEND/RECV Verbs and also locally executed within an HTM transaction. As in prior work [16, 54], DrTM+R adopts incarnation mechanism to detect invalidation between read and insert/delete operations. The incarnation is initially zero and is monotonously increased by insert/delete operations.

4.4 Commit Phase

To fully leverage advanced hardware features, DrTM+R proposes an HTM/RDMA-friendly six-step commit phase, which uses HTM for local records and RDMA-based locking for remote records. For the remote accesses, DrTM+R uses one-sided RDMA operations instead of messaging used in FaRM [17], since the later increases the number of interrupts and context switches on the remote machine, which will unconditionally abort the HTM transactions even without access conflicts. Figure 7 illustrates the pseudo-code of the commit phase in DrTM+R.

C.1 Lock (remote read/write sets). DrTM+R exclusively locks remote records in both read and write sets using one-sided RDMA CAS (compare-and-swap), which provides an equal semantic to the normal CAS instruction (i.e., local CAS). However, there is an atomicity issue between local CAS and RDMA CAS operations. The atomicity of RDMA CAS is hardware-specific [32], which can implement any one of the three levels: IBV_ATOMIC_NONE, IBV_ATOMIC_HCA, and IBV_ATOMIC_GLOB. The RDMA CAS can only correctly work with local CAS under IBV_ATOMIC_GLOB level, while our InfiniBand NIC[4] only provides the IBV_ATOMIC_HCA level of atomicity. This means that RDMA CASs can only correctly lock each other. Fortunately, the lock will only be acquired and released by remote accesses using RDMA CAS in our protocol. The local access will only check the state of a lock, which can correctly work with RDMA CAS due to the cache coherency of RDMA operations.

C.2 Validate (remote read set). DrTM+R performs read validation to remote read records using one-sided RDMA READs. It checks whether the current sequence numbers of records are equal to those acquired by remote reads in the

[4] Mellanox ConnectX-3 MCX353A 56Gbps InfiniBand NIC.

```
COMMIT()
  //C.1 Lock remote writes and reads
  foreach <key> in remote_set
    if RDMA_CAS(key, LOCKED, UNLOCKED) != UNLOCKED
      ABORT() //ABORT: conflict TX

  //C.2 Validate remote reads
  foreach <key, SN> in remote_readset
    rec = RDMA_READ(key)
    if SN != rec.SN
      ABORT() //ABORT: conflict TX

  XBEGIN() //HTM TX begin        HTM Transaction
  //C.3 Validate local reads
  foreach <key, SN> in local_readset
    if SN != rec[key].SN
      ABORT() //ABORT: conflict TX

  //C.4 Update local writes
  foreach <key, value> in local_writeset
    if rec[key].lock == LOCKED
      ABORT() //ABORT: conflict TX
    rec[key] = <value, SN++>
  XEND() //HTM TX end

  //C.5 Update remote writes
  foreach <key, value> in remote_writeset
    rec = <value, SN++>
    RDMA_WRITE(key, rec)

  //C.6 Unlock remote writes and reads
  foreach <key> in remote_set
    RDMA_CAS(key, UNLOCKED, LOCKED)
```

Figure 7. The pseudo-code of commit in DrTM+R.

ISOLATION	COMMIT/L	COMMIT/R
COMMIT/L	HTM	HTM & Locking
COMMIT/R	HTM & Locking	Locking

Table 3. A summary of different mechanisms used to ensure *isolation* of the commitment on local and remote records. **L** and **R** stand for Local and Remote records.

```
LOCAL_READ_RO(key)
  return LOCAL_READ(key)

REMOTE_READ_RO(key)
L:rec = RDMA_READ(key)
  if rec.lock == LOCKED || !MATCHING_VERSIONS(rec)
    goto L //RETRY: inconsistent read
  remote_readset.add(key, rec.seqno)
  return rec.value

COMMIT_RO()
  //CR.1 Validate remote reads
  foreach <key, seqno> in remote_readset
    rec = RDMA_READ(key)
    if rec.seqno != seqno
      ABORT() //ABORT: conflict TX

  //CR.2 Validate local reads
  foreach <key, seqno> in local_readset
    if rec[key].seqno != seqno
      ABORT() //ABORT: conflict TX
```

Figure 8. The pseudo-code of read-only transaction interface in DrTM+R.

execution phase. If any one is changed, the transaction is aborted. Different from traditional OCC protocol, DrTM+R must lock the remote records even for reads to ensure strict serializability, since local records in the write set are not protected by HTM until the next step. However, this lock will not block remote read in the execution phase. Further, in an NIC with the IBV_ATOMIC_GLOB atomicity level, the lock field can be encoded in the sequence number of a record, so that DrTM+R can lock and validate the record using a single RDMA CAS. Unfortunately, this is not available in our NIC, and DrTM+R first uses RDMA CAS to lock a remote record and then fetch the sequence number to validate the record locally.

C.3 Validate (local read set) and C.4 Update (local write set). DrTM+R performs read validation to local records in the read set first, and commits buffered updates and increased sequence number to local records in the write set. The entire accesses on local records are protected by an HTM transaction. The HTM transaction provides strong atomicity with any concurrent local accesses to the same records, and any remote conflicting RDMA accesses will also abort the HTM transaction. The only exception is that a remote transaction may lock a local record in the write set before the start of the HTM transaction. To remedy this issue, DrTM+R adds an additional check before the update and manually abort the transaction upon a failed check.

C5. Update (remote write set) and C.6 Unlock (remote read/write sets) DrTM+R writes back updates to remote records and increases their sequence numbers using RDMA WRITEs, and after that it reports the transaction as committed to user. Finally, DrTM+R unlocks all remote records using RDMA CAS.

Table 3 summarizes different mechanisms used by DrTM+R to atomically commit the updates on local and remote records.

4.5 Read-only Transactions

Read-only transaction is a special case which usually has a very large read set involving up to hundreds or even thousands of records. Thus, it will likely abort an HTM transaction in the commit phase due to read validation on a large local read set, since HTM tracks reads and writes at the granularity of a cache line even if only several bytes (e.g., sequence number) are accessed. To address this issue, DrTM+R provides a separate protocol to execute read-only transactions without HTM and locking in the commit phase.

Figure 8 shows the pseudo-code and interfaces for read-only transactions. In the execution phase, the read access to local records is the same to that of read-write transactions, since it only reads one record at a time. In contrast, the read accesses to remote records requires an additional check for the lock to avoid reading uncommitted reads. If the record

is concurrently updated by a remote transaction, the record must be locked. On the other hand, if the record is concurrently updated by a local transaction, the remote read using RDMA READ will abort the transaction. In the commit phase, DrTM+R can validate only the sequence numbers of records in both local and remote read set without any protection of HTM or locking.

4.6 Strict Serializability

This section gives an informal argument on the strict serializability of our hybrid concurrency control protocol. We argue it by reduction that our protocol is equal to traditional optimistic concurrency control (OCC) [23].

Committed read-write transactions are serializable at the point of the end of HTM transaction in the commit phase. This is because the versions of all records in read and write sets are the same as the versions seen in the execution phase. First, locking ensures this for remote write records. Second, the HTM transaction ensures this for local write records. Finally, the validation phase ensures this invariant for all read records. Even if the read validation on remote records is earlier than the HTM transaction, locking on them makes it equivalent to the validation within the HTM transaction. Committed read-only transactions are serializable at the point of their last read. The validation phase ensures the versions of all records in the read set at the serialization point are the same as the versions seen in the execution phase. Therefore, this is equivalent to executing and committing the entire transaction atomically at the serialization point. Further, the serialization point of transactions is always between receiving the request to start the transaction and reporting the committed transaction to user, which ensures strict serializability.

5. Replication

Many in-memory transaction systems [12, 17, 25, 48] adopt replication on remote machines to support durability and availability. However, any I/O operation including RDMA inside an HTM transaction will unconditionally cause an HTM abort. Therefore, prior HTM-based transaction systems [52, 54] only preserve durability rather than availability by logging to local reliable storage in case of machine failures.

5.1 Primary-backup Replication

DrTM+R follows FaRM [17] to use vertical Paxos [24] with primary-backup replication to provide durability and availability. DrTM+R should send all updates of records to the non-volatile logs on backup machines after read validation and before updating the primaries of both local and remote records, and ask them to truncate logs at the end of a transaction. Note that using auxiliary threads to truncate logs will not impact worker threads to update primaries since the backups of records will only be used in recovery.

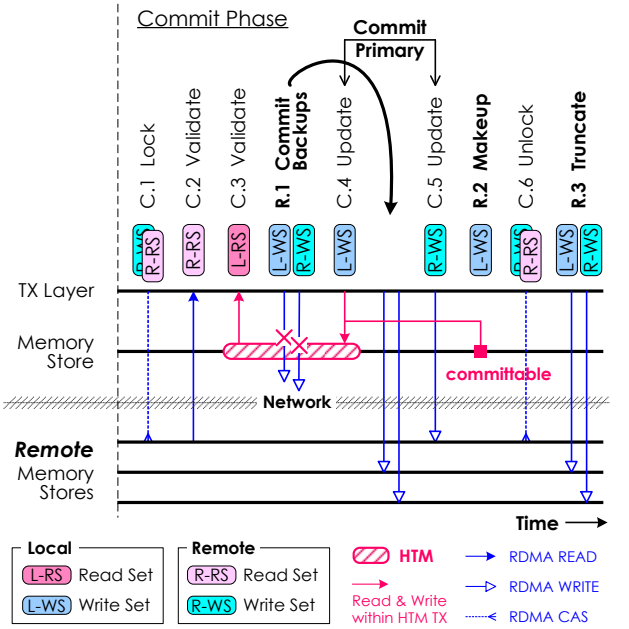

Figure 9. The primary-backup replication in DrTM+R.

However, using HTM and RDMA to implement distributed transactions raises a key challenge for replication. Both validation (C.3) and update (C.4) to local records are performed within an HTM transaction, and thus replication to remote machines (R.1) through network I/O inside an HTM transaction is not allowed inside the HTM transaction, as shown in Figure 9.

An intuitive solution is to directly move updating backups (R.1) after the HTM transaction commits. However, all machines can immediately observe the local updates after the HTM transaction commits (i.e., `XEND`) through local or RDMA read. Consequently, a subsequent transaction may read the updates and then commit, while this transaction may not commit due to machine failures and the backups does not receive updates.

DrTM+R proposes an *optimistic* replication scheme, which can cooperate with our hybrid OCC protocol. The key idea is to leave all locally written records in an *uncommittable* status after the HTM transaction commits; they will transform to a *committable* status until both primaries and backups of the written records have been applied. The *uncommittable* record can be optimistically read in the execution phase. In the commit phase, the record cannot be updated when being *uncommittable*, and read validation will fail if the record is still *uncommittable* or has been changed. The optimistic replication scheme preserves strict serializability since all subsequent transactions observing the records updated by a prior transaction will not commit until the prior transaction commits.

DrTM+R reuses the sequence number (SN) of records to implement *optimistic* replication. An *odd* sequence number indicates *uncommittable*, and an *even* sequence num-

	OCC	OCC+OR
C.4 Update L_WS Primary	SN_{new}+1	SN_{new}+1
R.1 Update L_WS Backup	/	SN_{new}+2
R.1 Update R_WS Backup	/	SN_{new}+2
R.2 Makeup L_WS Primary	/	SN_{new}+1
C.5 Update R_WS Primary	SN_{new}+1	SN_{new}+2

	OCC	OCC+OR
	Condition of Validation	
C.2 R_RS	$SN_{old} == SN_{cur}$	$(SN_{old} + 0x1)$ & $\sim 0x1 == SN_{cur}$
C.2 R_WS	/	SN_{cur} & $0x1 == 0x0$
C.3 L_RS	$SN_{old} == SN_{cur}$	$(SN_{old} + 0x1)$ & $\sim 0x1 == SN_{cur}$
C.4 L_WS	/	SN_{cur} & $0x1 == 0x0$

Table 4. (a) the change on sequence number (SN) and (b) the condition of validation for records within different read/write sets in COMMIT. **OR** stands for optimistic replication. **L** and **R** stand for Local and Remote. **WS** and **RS** stand for Write Set and Read Set.

ber indicates *committable*, which is similar to the seqlock used in Linux. Table 4 summarizes the change on HTM-based OCC to support *optimistic* replication. For backups of changed records (R.1) and primaries of remote written records (C.5), DrTM+R directly increases the sequence number by 2 when updating them. For primaries of local write records, DrTM+R increases the sequence number by 1 when updating them within the HTM transaction (C.4) and the makeup phase (R.2) accordingly. There is no change to the sequence number for local and remote read in the execution phase. In the commit phase, DrTM+R validates all written records using the condition that the current sequence number should be *even*, and changes the validation condition to all read records as the current sequence number should be equal to the closest *committable* sequence number of acquired sequence number in the execution phase. Since the write set is generally a subset of the read set (blind write is rare) and the condition for written records is included in the condition for read records, leveraging *optimistic* replication only incurs small overhead to the commit phase and has no impact to the execution phase.

5.2 Failure Recovery

DrTM+R uses similar failure models as prior work [17, 25]. All logs are stored in non-volatile memory, and the logs will not lose upon machine failures (e.g., relying on an uninterruptible power supply). A machine in a cluster may crash at any time, but only in a fail-stop manner instead of arbitrary failures like Byzantine failures [11, 22].

DrTM+R adopts primary-backup replication in non-volatile memory with $f+1$ copies. Therefore, it can provide durability even under a complete cluster failure and losing at most f copies for each record. DrTM+R can also provide availability with at least 1 copy of each record on surviving machines.

DrTM+R uses the same mechanisms in FaRM [17] to detect machine failures and reconfigure the cluster, but a varied failure-recovery protocol for transaction state recovery. The main difference is that DrTM+R directly lock and unlock remote records using one-sided RDMA CAS. This reduces the latency of transactions compared to FaRM, which sends a LOCK message to the logs on target machines and relies on target worker threads to lock records and sends back responses. However, it may cause dangling locks after a failure since there is no log to find records locked by failed machines. To avoid suspending the whole cluster and checking all records on each machine, DrTM+R adopts a passive approach to releasing such records. DrTM+R will first encode the owner machine ID into the lock of records. Further, the worker thread will check whether the owner of the locked record is the member of the current configuration or not. If the owner is absent, the worker thread will unlock the record before aborting and retrying the transaction. It should be noted that the additional check will not incur perceptibly overhead to DrTM+R since it is not on the critical path of normal execution.

6. Implementation Issues

We have implemented DrTM+R using Intel's Restricted Transactional Memory (RTM) and Mellanox ConnectX-3 56Gbps InfiniBand. This section describes some specific implementation issues.

6.1 Fallback Handler and Contention Management

As a best-effort mechanism, an RTM transaction does not have guaranteed forward progress even in the absence of conflicts. A fallback handler will be executed after the number of RTM aborts exceeds a threshold. In typical implementation, the fallback handler first acquires a coarse-grained exclusive lock, and then directly updates all records. To cooperate with the fallback handler, the RTM transaction needs to check this lock before entering its RTM region.

In DrTM+R, however, since local records will also be remotely accessed by other transactions, the fallback handler may inconsistently update the record out of an RTM region. Therefore, DrTM+R needs to lock and validate the local records similar to those required for remote read/write records. To avoid deadlocks, the fallback handler should release all owned remote locks first, and then acquire appropriate locks for all records in a sorted order. After that, the fallback handler can execute the validation as usual.

6.2 Atomicity Issues

As mentioned in §4.4, even if RDMA CAS on our InfiniBand NIC cannot preserve the atomicity with the local CAS, it will not incur consistency issues in the normal execution of transactions. However, in the RTM's fallback handler of the commit phase, DrTM+R has to lock both local and remote records. In fact, calling fallback handler in DrTM+R is rare (lower than 1%) since the conflicts between transactions are

mainly detected by read validation of OCC. Therefore, even the current performance of RDMA CAS is two orders of magnitude slower than the local counterpart, DrTM+R still uniformly uses RDMA CAS to lock local records in fallback handlers.

6.3 Memory Store

The memory store layer of DrTM+R provides a general key-value store interface to the upper transaction layer. The most common usage of this interface is to read or write records by given keys. To optimize for different access patterns [5, 28, 29], DrTM+R provides both an ordered store in the form of a B^+-tree and an unordered store in the form of a hash table. For the ordered store, we use the B^+-tree in DBX [52], which uses HTM to protect the major B^+-tree operations and was shown to have comparable performance with state-of-the-art concurrent B^+-tree [31]. For the unordered store, we use the HTM/RDMA-friendly hash table in DrTM [54], which uses one-sided RDMA operations for both reads and writes, as well as provides an RDMA-friendly, location-based and host-transparent cache to reduce RDMA lookup cost.

6.4 Local Record Update

Since RTM tracks writes using L1 cache, the write working set (32K) is much smaller than that of read. To reduce the working set for local write within the HTM region in the commit phase, DrTM+R updates local records by swapping the pointer of local buffer instead of overwriting actual records. However, this optimization can only apply to records that are always locally accessed. For example, the `NEW_ORDER` and `CUSTOMER` tables used by new-order and delivery transactions in TPC-C.

7. Evaluation

This section presents the evaluation of DrTM+R, with the goal of answering the following questions:

- How does the performance of DrTM+R with RTM and RDMA compare to that of the state-of-the-art systems without using such features?
- Can DrTM+R scale out with the increase of threads and machines?
- How does each design decision affect the performance of DrTM+R?
- How fast can DrTM+R recover from failures?

7.1 Experimental Setup

The performance evaluation was conducted on a local cluster with 6 machines. Each machine has two 10-core RTM-enabled Intel Xeon E5-2650 v3 processors with 64GB of DRAM. Each core has a private 32KB L1 cache and a private 256KB L2 cache, and all 10 cores on a single processor share a 24MB L3 cache. We disabled hyper-threading on all machines. Each machine is equipped with a ConnectX-3 MCX353A 56Gbps InfiniBand NIC via PCIe 3.0 x8 connected to a Mellanox IS5025 40Gbps InfiniBand Switch, and an Intel X520 10GbE NIC connected to a Force10 S4810P 10/40GbE Switch. All machines run Ubuntu 14.04 with Mellanox OFED v3.0-2.0.1 stack. We reserve two cores to run auxiliary threads on each processor for log truncation. DrTM+R=3 represents 3-way replication enabled for providing high availability, and will replicate to standby machines when running on less than 3 machines.

TPC-C	NEW	PAY	DLY	OS	SL
Ratio	45%	43%	4%	4%	4%
Type	d+rw	d+rw	l+rw	l+ro	l+ro

SmallBank	SP	AMG	BAL	DC	WC	TS
Ratio	25%	15%	15%	15%	15%	15%
Type	d+rw	d+rw	l+ro	l+rw	l+rw	l+rw

Table 5. The transaction mix ratio in TPC-C and SmallBank. **`d`** and **`l`** stand for distributed and local. **`rw`** and **`ro`** stand for read-write and read-only. The default probability of cross-warehouse accesses for **`NEW`** and **`PAY`** in TPC-C is 1% and 15% respectively.

We use two standard benchmarks to evaluate DrTM+R: TPC-C [46] and SmallBank [4]. TPC-C is a widely-used OLTP benchmark that simulates principal transactions of an order-entry environment. These transactions include entering and delivering orders (new-order and delivery), recording payments (payment), checking the status of orders (order-status), and monitoring the level of stock at the warehouses (stock-level). TPC-C scales by partitioning a database into multiple warehouses spreading across multiple machines. As specified by the benchmark, the throughput of TPC-C is defined as how many new-order transactions per second a system processed while the system is executing four other transactions types. We run the standard-mix but report the throughput of new-order transactions, which are 45% of total transactions.

SmallBank models a simple banking application where transactions perform simple read and write operations on user accounts. The access patterns of transactions are skewed such that a few accounts receive most of the requests. SmallBank is a mix of six types of transactions for send-payment (SP), balance (BAL), deposit-checking (DC), withdraw-from-checking (WC), transfer-to-savings (TS), and amalgamate (AMG) procedures.

Table 5 shows the percentage of each transaction type and its access pattern in TPC-C and SmallBank.

It is often hard for cross-system comparison especially for distributed systems. We keep the settings among different systems to be identical for each benchmark. We use the latest Calvin [48] (released in Mar. 2015) and DrTM[54] for comparison in our experiments. As Calvin is hard-coded to use 8 worker threads per machine, we have to skip it from

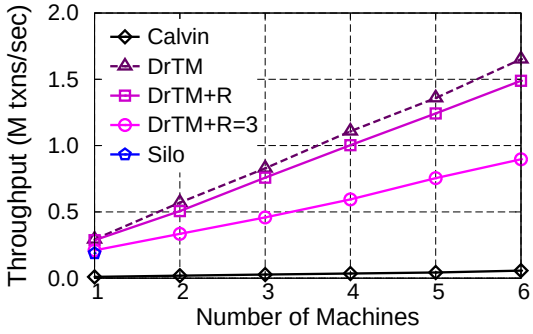

Figure 10. The throughput of new-order transactions in TPC-C with the increase of machines while fixing 8 threads each.

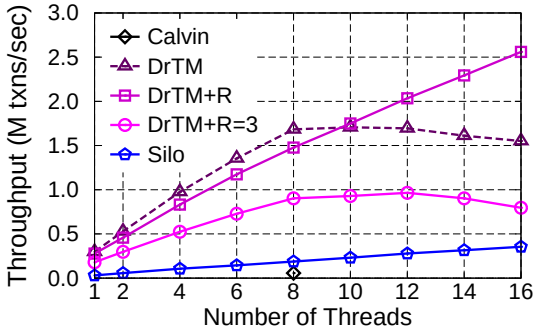

Figure 11. The throughput of new-order transactions in TPC-C with the increase of threads while fixing 6 machines.

the experiment with varying numbers of threads. We run Calvin on our InfiniBand network using IPoIB as it was not designed to use RDMA features, and the released code of Calvin does not provide logging or replication. We also run Silo [49] (with logging disabled), a state-of-the-art single-machine multicore database, on one machine of our cluster.

In all experiments, we dedicate one processor to run up to 8 worker threads and 2 auxiliary threads. We use the same machine to generate requests to avoid the impact of networking between clients and servers as done in prior work [48, 49, 52]. All experimental results are the average of five runs.

7.2 Performance and Scalability

TPC-C: We first run TPC-C with the increase of machines to compare the performance with DrTM and Calvin. Each machine is configured to run 8 worker threads and each of them hosts 1 warehouse with 10 districts. All warehouses in a single machine shares one memory store. Figure 10 shows the throughput of the new-order transaction in TPC-C's standard-mix workload. Compare to DrTM, DrTM+R trades roughly 9.8% (from 2.2%) performance for generality. The main overhead is due to manually maintaining the local read/write buffers of transactions.

DrTM+R can scale well in term of the number of machines and provide more than 1.49 million new-order and 3.31 million standard-mix transactions per second (txns/sec)

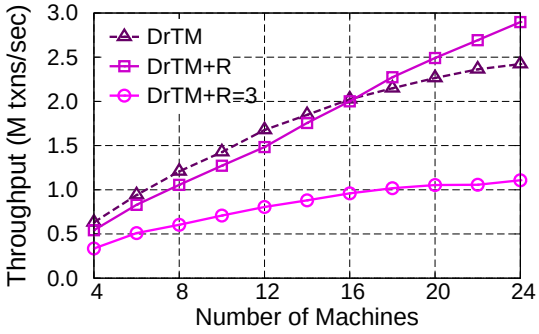

Figure 12. The throughput of new-order transactions in TPC-C with the increase of separate logical machines while fixing 4 threads each.

on 6 machines with 8 threads each, which outperforms Calvin by up to 29.3X (from 26.8X). DrTM+R=3 only incurs up to 41% (from 27%) overhead compared to DrTM+R due to enabling 3-way replication, and still can outperform Calvin by up to 21.5X (from 15.8X). However, the performance of DrTM+R=3 stops scaling after 12 threads (see Figure 11). The reason is that DrTM+R=3 requires many RDMA operations for replication and thus saturate the limit of NIC. This finding is consistent with FaRM, whose successive version [17] uses two 56Gbps RDMA-capable NICs per machine to overcome this bottleneck. We believe if we similarly deploy two NICs per machine, DrTM+R will scale much better.

We further study the scalability of DrTM+R with the increase of worker threads using 6 machines. As shown in Figure 11, the performance of DrTM drops over 8 threads due to cross-socket overhead and large working set in HTM. DrTM+R can scale well up to 16 threads, reaching 2.56 million new-order and 5.69 million standard-mix transactions per second. The speedup of throughput using 16 threads can reach 9.21X due to small working set in HTM and much lower HTM abort rate (less than 1%).

As an aside, DrTM+R also has very good single-node performance. Under 8 threads where all systems scale, the per-machine throughput for DrTM+R is 150,487 and 248,142 new-order transactions per second for with and without 3-way replication accordingly, which is comparable or even faster than Silo without logging (187,747 txns/sec[5]). Under 16 threads, the per-machine throughput of Silo and DrTM+R is 354,579 and 426,628 txns/sec accordingly. This confirms that our HTM/RDMA-friendly concurrency protocol does not sacrifice per-machine efficiency.

To overcome the restriction of existing cluster size, we scale up to 4 separate logical nodes on a single machine to emulate the scalability experiment, each of which has fixed 4 worker threads. The interaction between logical nodes still uses our RDMA-based OCC protocol even on the same

[5] Silo only reports standard-mix transactions per second which we multiplied by 45% to get the new order transactions per second.

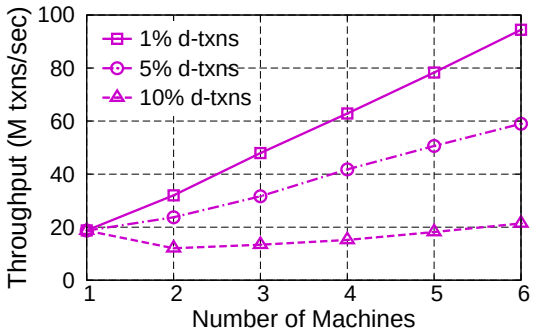

Figure 13. The throughput of standard-mix in SmallBank on DrTM+R with the increase of machines using different probability of cross-machine accesses for **SP** and **AMP**.

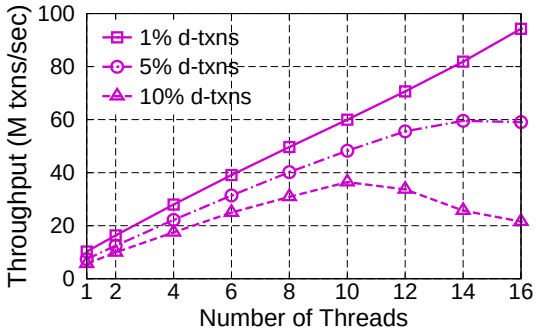

Figure 14. The throughput of standard-mix in SmallBank on DrTM+R with the increase of threads using different probability of cross-machine accesses for **SP** and **AMP**.

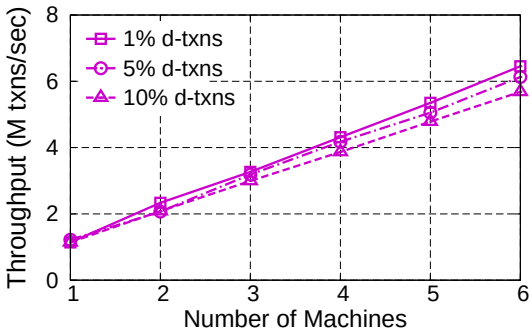

Figure 15. The throughput of standard-mix in SmallBank on DrTM+R=3 with the increase of machines using different probability of cross-machine accesses for **SP** and **AMP**.

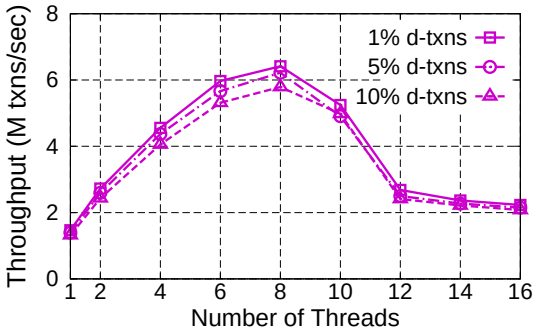

Figure 16. The throughput of standard-mix in SmallBank on DrTM+R=3 with the increase of threads using different probability of cross-machine accesses for **SP** and **AMP**.

machine. As shown in Figure 12, DrTM+R can scale well on 24 logical nodes, reaching 2.89 million new-order and 6.43 million standard-mix transactions per second.

SmallBank: We further study the performance and scalability of SmallBank with varying probability of distributed transactions on DrTM+R. Figure 13 and Figure 14 show the throughout of SmallBank on DrTM+R (3-way replication disabled) with the increase of machines and threads accordingly. For a low probability of distributed transactions (1%), DrTM+R provides high performance and can scale well in two dimensions. It can achieves over 94 million transactions per second using 6 machines with 16 threads each, and the speedup of throughput reaches more than 5.0X for 6 machines and 9.2X for 16 threads respectively. With the growth of distributed transactions, DrTM+R still performs stable throughput increase from 2 machines and scale-well within a single socket.

Figure 15 and Figure 16 show the throughput of SmallBank on DrTM+R=3 (3-way replication enabled) with the increase of machines and threads accordingly. DrTM+R=3 can scale well with the increase of machines, but only scale up to 8 threads (6.4 million txns/sec) because the single 56Gbps InfiniBand NIC on each machine becomes the bottleneck. Each transaction requires at least four RDMA WRITEs for replication. Further, the peak throughput of DrTM+R=3 is much lower than that of DrTM+R since all transactions in SmallBank only perform a few accesses to the records (i.e., 1 read and 1 write). Consequently, additional RDMA operations for replication will dominate the execution time of transactions. Similarly, deploying more NICs per machine would make DrTM+R=3 scale much better.

7.3 Impact from Distributed Transactions

We further investigate the performance of DrTM+R for distributed transactions. We adjust the probability of cross-warehouse accesses for new-order transactions from 1% to 100%, the default setting is 1% according to TPC-C specification. Since the average number of items accessed in the new-order transaction is 10, 10% of cross-warehouse accesses will result in approximate 57.2% of distributed transactions.

Figure 17 shows the throughput of new-order transaction on different systems with increasing cross-warehouse accesses. The 100% cross-warehouse accesses results in 73.1% and 81.7% slowdown for DrTM+R with and without 3-way replication respectively, because all transactions are distributed and any accesses are remote ones. However, the performance slowdown for 5% cross-warehouse accesses (close to 35% distributed transaction) is moderate

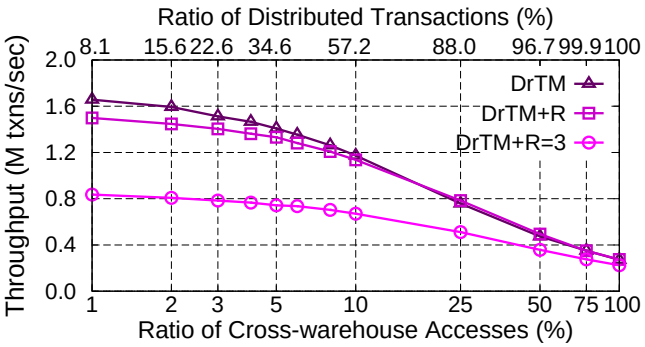

Figure 17. The throughput of new-order transaction in TPC-C with the increase of cross-warehouse accesses while fixing 6 machines and 8 threads each.

(11.0% and 11.1%). In addition, the performance gap between DrTM+R and DrTM becomes narrow with increasing distributed transactions, since both of them adopts a similar mechanism to update remote records.

7.4 Impact from High Contention

To evaluate the performance with a high contention scenario, we configure each machine to use *only one* warehouse for TPC-C. Figure 18 shows the throughput of new-order transaction in TPC-C for DrTM+R and DrTM on 6 machines with the increase of threads. DrTM+R can still outperform DrTM when there are less than 10 worker threads per machine, this is mainly because DrTM would fall back to a slow path with locking more frequently under high contentions. As an optimistic concurrency control scheme, DrTM+R incurs more overhead with the increase of threads due to more contention and increased read-write conflicts in the commit phase.

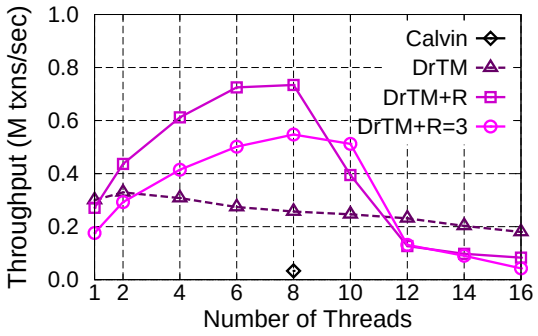

Figure 18. The throughput of new-order transaction in TPC-C with increasing threads while fixing one warehouse per machine.

7.5 Impact from Data Size

To investigate the impact on throughput from the growing of database, we configure TPC-C with up to 384 warehouses (64 warehouses per machine), which uses approximately 28GB and 9GB of DRAM on each machine for DrTM+R with and without 3-way replication respectively. As shown in Figure 19, the throughput of new-order transaction on

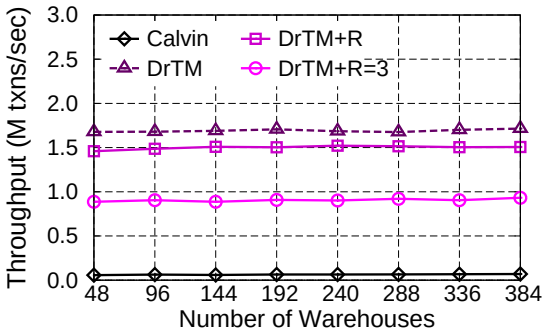

Figure 19. The throughput of new-order transaction in TPC-C with the increase of warehouses while fixing 6 machines and 8 threads each.

each system is stable and even increasing slightly from 48 warehouses. A large database may increase the cache miss rate, but it also reduces the contention on the database.

7.6 Replication

To investigate the performance cost for replication, we evaluate how throughput and latency changes for TPC-C with 3-way replication. Table 6 shows the performance difference on 6 machines and each with 8 threads. Due to additional RDMA WRITEs to backups for 3-way replication, the throughput of the new-order transaction on DrTM+R drops by 40% and the latency increases 10-20μs. However, we can see that the rate of validation abort and executing fallback handler is almost no change, since replicating log to backups is outside the HTM transaction and does not impact read validation. Compared to DrTM, DrTM+R trades less than 10% throughput for generality (1,651,763 vs. 1,488,850). In addition, the increase of steps in commit phase of DrTM+R does not apparently harm the 50^{th} and 90^{th} percentile latency compared to DrTM (6.55μs and 23.67μs). Specially, for the long-tail (99^{th} percentile) latency, DrTM+R is even better than DrTM (86.96μs vs. 80.95μs) due to thoroughly eliminating RTM capacity aborts and avoiding mostly fallback path execution (from 10.02% to 0.67%)

		DrTM+R	DrTM+R=3
Standard-mix (txn/sec)		3,308,555	1,849,224
New-order (txn/sec)		1,488,850	902,924
Latency (μs)	**50%**	8.93	18.98
	90%	26.24	55.33
	99%	80.95	91.51
Validation Abort Rate (%)		3.50	3.90
Fallback Path Rate (%)		0.67	0.68

Table 6. The impact of 3-way replication on throughput and latency for TPC-C on 6 machines with 8 threads each.

7.7 Recovery

To evaluate performance of DrTM+R=3 upon failures, we run TPC-C on our 6-node cluster with 8 worker threads each. DrTM+R=3 enables 3-way replication and conservatively

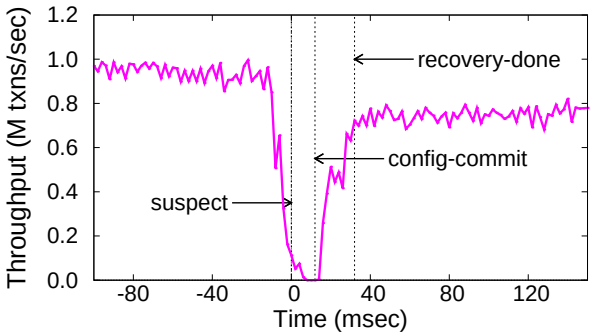

Figure 20. The throughput timeline for new-order transaction in TPC-C with failure.

sets the leases of machines to 10ms. During evaluation, we kill one machine by turning off its networking, and the instance on failed machine will be recovered on one of the surviving machines. Figure 20 shows the timeline with the throughput of new-order transactions in TPC-C aggregated at 2ms intervals, which is a zoomed-in view around the failure. It also shows the time at which the failed machine is detected due to lease expired ("*suspect*"); the time at which the new configuration was committed at all surviving machines ("*config-commit*"); the time at which the recovery on all machines is done ("*recovery-done*").

As shown in Figure 20, the throughput drops notably upon failure but rises rapidly again in about 40ms, 10ms of which is spent for suspecting a failure and the rest is used for recovery. The regained throughput of TPC-C is approximately 80% of original peak throughput, because the instance on failed machine is revived on a surviving machine and there are only 5 machines to handle the workload. Besides, two instances will slightly interfere with each other due to sharing a single InfiniBand NIC.

8. Other Related Work

Distributed transactions: Providing low-latency, high-throughput transactions has been a long line of research [3, 12, 13, 15, 17, 26, 35, 37, 38, 48, 49, 55, 56, 58–60]. Rococo [35] reorders conflicting pieces of contended transactions to reduce conflicts while retaining serializability. Callas [56] instead provides a modular concurrency control scheme that partitions a set of transactions into a set of groups and enforces the serializability of each group separately. Yesquel [3] instead leverages a distributed balance tree to provide scalable transactions across a cluster of machines. RIFL [26] and Tapir [58] instead boost transaction processing by providing a different underlying mechanism for transaction layer: RIFL provides exact-once RPC semantics to implement linearizable transactions; Tapir instead builds a consistent transaction layer atop an inconsistent replication layer to remove redundant support for consistency in both layers. DrTM+R has much better performance than these systems due to the use of advanced hardware features like HTM and RDMA.

Distributed transactional memory: There have been some effort to investigate the feasibility of distributed transactional memory using hardware of software approaches. Herlihy and Sun [19] propose a hierarchical cache coherence protocol that takes distance and locality into account to support transactional memory in a cluster. However, there is no actual implementation and evaluation of the proposed schemes. Researchers have also investigated the design and implementation of distributed transactional memory [8, 10, 30], which, however, usually have inferior performance than its hardware counterpart. Like DrTM [54], DrTM+R also leverages the strong consistency of RDMA and strong atomicity of HTM to support fast database transactions, but further provides availability and requires no priori knowledge of read/write sets.

Concurrency control: There have been multiple approaches to implement concurrency control, including two-phase locking [1, 9, 14, 23], timestamp ordering [6, 27] and commit ordering [41, 42]. There are also other varieties that leverages dependencies to improve performance, like dependency-aware software transactional memory [40], ordered sharing lock [2] and balanced concurrency control [57]. Besides DBX, DBX-TC, and DrTM, Leis et al. [27] combines time-stamp ordering with HTM to provide scalable transactions. DrTM+R is built atop prior concurrency control approaches by combining OCC with HTM and RDMA to derive a hybrid approach to general transaction processing.

9. Conclusion

This paper described DrTM+R, an in-memory transaction processing system that leverages advanced hardware features like HTM and RDMA to provide high performance and low latency, while preserving the generality and providing high availability. DrTM+R leverages a hybrid concurrency scheme that combines OCC with remote locking for distributed transactions, and uses a progressive replication scheme to tackle the race condition between replication and HTM transaction commits. Evaluations using typical OLTP workloads like TPC-C and SmallBank confirmed the benefit of designs in DrTM+R.

Acknowledgments

We sincerely thank our shepherd Dushyanth Narayanan and the anonymous reviewers for their insightful suggestions. This work is supported in part by China National Natural Science Foundation (61402284, 61572314), Doctoral Fund of Ministry of Education of China (No. 20130073120040), National Youth Top-notch Talent Support Program of China, the Shanghai Science and Technology Development Fund for high-tech achievement translation (No. 14511100902), Zhangjiang Hi-Tech program (No. 201501-YP-B108-012) and Singapore NRF (CREATE E2S2).

References

[1] AGRAWAL, D., BERNSTEIN, A. J., GUPTA, P., AND SENGUPTA, S. Distributed optimistic concurrency control with reduced rollback. *Distributed Computing 2*, 1 (1987), 45–59.

[2] AGRAWAL, D., EL ABBADI, A., JEFFERS, R., AND LIN, L. Ordered shared locks for real-time databases. *The VLDB Journal 4*, 1 (1995), 87–126.

[3] AGUILERA, M. K., LENERS, J. B., KOTLA, R., AND WALFISH, M. Yesquel: scalable sql storage for web applications. In *SOSP* (2015), ACM.

[4] ALOMARI, M., CAHILL, M., FEKETE, A., AND RÖHM, U. The cost of serializability on platforms that use snapshot isolation. In *IEEE 24th International Conference on Data Engineering* (2008), ICDE'08, IEEE, pp. 576–585.

[5] BATOORY, D., BARNETT, J., GARZA, J. F., SMITH, K. P., TSUKUDA, K., TWICHELL, B., AND WISE, T. Genesis: An extensible database management system. *IEEE Transactions on Software Engineering 14*, 11 (1988), 1711–1730.

[6] BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. *Concurrency control and recovery in database systems*, vol. 370. Addison-wesley New York, 1987.

[7] BLUNDELL, C., LEWIS, E. C., AND MARTIN, M. M. Subtleties of transactional memory atomicity semantics. *Computer Architecture Letters 5*, 2 (2006).

[8] BOCCHINO, R. L., ADVE, V. S., AND CHAMBERLAIN, B. L. Software transactional memory for large scale clusters. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2008), PPoPP'08, ACM, pp. 247–258.

[9] BORAL, H., ALEXANDER, W., CLAY, L., COPELAND, G., DANFORTH, S., FRANKLIN, M., HART, B., SMITH, M., AND VALDURIEZ, P. Prototyping bubba, a highly parallel database system. *Knowledge and Data Engineering, IEEE Transactions on 2*, 1 (1990), 4–24.

[10] CARVALHO, N., ROMANO, P., AND RODRIGUES, L. Asynchronous lease-based replication of software transactional memory. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware* (2010), Middleware'10, Springer-Verlag, pp. 376–396.

[11] CASTRO, M., AND LISKOV, B. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (1999), OSDI'99, USENIX Association, pp. 173–186.

[12] CORBETT, J. C., DEAN, J., EPSTEIN, M., FIKES, A., FROST, C., FURMAN, J. J., GHEMAWAT, S., GUBAREV, A., HEISER, C., HOCHSCHILD, P., HSIEH, W., KANTHAK, S., KOGAN, E., LI, H., LLOYD, A., MELNIK, S., MWAURA, D., NAGLE, D., QUINLAN, S., RAO, R., ROLIG, L., SAITO, Y., SZYMANIAK, M., TAYLOR, C., WANG, R., AND WOODFORD, D. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (2012), OSDI'12, USENIX Association, pp. 251–264.

[13] COWLING, J., AND LISKOV, B. Granola: low-overhead distributed transaction coordination. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference* (2012), USENIX ATC'12, USENIX Association.

[14] DEWITT, D. J., GHANDEHARIZADEH, S., SCHNEIDER, D. A., BRICKER, A., HSIAO, H.-I., AND RASMUSSEN, R. The gamma database machine project. *Knowledge and Data Engineering, IEEE Transactions on 2*, 1 (1990), 44–62.

[15] DIACONU, C., FREEDMAN, C., ISMERT, E., LARSON, P.-A., MITTAL, P., STONECIPHER, R., VERMA, N., AND ZWILLING, M. Hekaton: SQL server's memory-optimized OLTP engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), SIGMOD'13, ACM, pp. 1243–1254.

[16] DRAGOJEVIĆ, A., NARAYANAN, D., HODSON, O., AND CASTRO, M. FaRM: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (2014), NSDI'14, USENIX Association, pp. 401–414.

[17] DRAGOJEVIĆ, A., NARAYANAN, D., NIGHTINGALE, E. B., RENZELMANN, M., SHAMIS, A., BADAM, A., AND CASTRO, M. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP'15, ACM, pp. 54–70.

[18] GRAY, J., AND REUTER, A. *Transaction processing: Concepts and Techniques*. Morgan Kaufmann Publishers, 1993.

[19] HERLIHY, M., AND SUN, Y. Distributed transactional memory for metric-space networks. In *Proceedings of the 19th International Conference on Distributed Computing* (2005), DISC'05, Springer-Verlag, pp. 324–338.

[20] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (2010), USENIX ATC'10, USENIX Association, pp. 11–11.

[21] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Using rdma efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (2014), SIGCOMM'14, ACM, pp. 295–306.

[22] KOTLA, R., ALVISI, L., DAHLIN, M., CLEMENT, A., AND WONG, E. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (2007), SOSP'07, ACM, pp. 45–58.

[23] KUNG, H. T., AND ROBINSON, J. T. On optimistic methods for concurrency control. *ACM Trans. Database Syst. 6*, 2 (June 1981), 213–226.

[24] LAMPORT, L., MALKHI, D., AND ZHOU, L. Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2009), PODC'09, ACM, pp. 312–313.

[25] LEE, C., PARK, S. J., KEJRIWAL, A., MATSUSHITA, S., AND OUSTERHOUT, J. Implementing linearizability at large scale and low latency. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 71–86.

[26] LEE, C., PARK, S. J., KEJRIWAL, A., MATSUSHITA, S., AND OUSTERHOUT, J. Implementing linearizability at large

scale and low latency. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP15)*. (2015).

[27] LEIS, V., KEMPER, A., AND NEUMANN, T. Exploiting hardware transactional memory in main-memory databases. In *IEEE 30th International Conference on Data Engineering* (2014), ICDE'14, IEEE, pp. 580–591.

[28] LINDSAY, B., MCPHERSON, J., AND PIRAHESH, H. A data management extension architecture. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (1987), SIGMOD'87, ACM, pp. 220–226.

[29] MAMMARELLA, M., HOVSEPIAN, S., AND KOHLER, E. Modular data storage with Anvil. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (2009), SOSP '09, ACM, pp. 147–160.

[30] MANASSIEV, K., MIHAILESCU, M., AND AMZA, C. Exploiting distributed version concurrency in a transactional memory cluster. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2006), PPoPP'06, ACM, pp. 198–208.

[31] MAO, Y., KOHLER, E., AND MORRIS, R. T. Cache craftiness for fast multicore key-value storage. In *Proceedings of the 7th ACM European Conference on Computer Systems* (2012), EuroSys'12, ACM, pp. 183–196.

[32] MELLANOX TECHNOLOGIES. RDMA aware networks programming user manual. http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf.

[33] MICROSOFT. Open cloudserver ocs v2 specification: Ocs open cloudserver powersupply v2.0. http://www.opencompute.org/wiki/Server/ SpecsAndDesigns, 2015.

[34] MITCHELL, C., GENG, Y., AND LI, J. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (2013), USENIX ATC'13, USENIX Association, pp. 103–114.

[35] MU, S., CUI, Y., ZHANG, Y., LLOYD, W., AND LI, J. Extracting more concurrency from distributed transactions. In *Proc. of OSDI* (2014).

[36] MURRAY, D. G., MCSHERRY, F., ISAACS, R., ISARD, M., BARHAM, P., AND ABADI, M. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), SOSP'13, ACM, pp. 439–455.

[37] NARULA, N., CUTLER, C., KOHLER, E., AND MORRIS, R. Phase reconciliation for contended in-memory transactions. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), OSDI'14, USENIX Association, pp. 511–524.

[38] POROBIC, D., LIAROU, E., TOZUN, P., AND AILAMAKI, A. Atrapos: Adaptive transaction processing on hardware islands. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on* (2014), IEEE, pp. 688–699.

[39] QIAN, H., WANG, Z., GUAN, H., ZANG, B., AND CHEN, H. Exploiting hardware transactional memory for efficient in-memory transaction processing. Tech. rep., Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, 2015.

[40] RAMADAN, H. E., ROY, I., HERLIHY, M., AND WITCHEL, E. Committing conflicting transactions in an stm. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2009), PPoPP '09, ACM, pp. 163–172.

[41] RAZ, Y. The principle of commitment ordering, or guaranteeing serializability in a heterogeneous environment of multiple autonomous resource mangers using atomic commitment. In *Proceedings of the 18th International Conference on Very Large Data Bases* (1992), Morgan Kaufmann Publishers Inc., pp. 292–312.

[42] RAZ, Y. Serializability by commitment ordering. *Information processing letters 51*, 5 (1994), 257–264.

[43] SNIA. Nvdimm special interest group. http://www.snia.org/forums/sssi/NVDIMM, 2015.

[44] STONEBRAKER, M. http://hpts.ws/papers/2013/allwrong.pdf, 2013.

[45] THE H-STORE TEAM. SmallBank Benchmark. http://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/.

[46] THE TRANSACTION PROCESSING COUNCIL. TPC-C Benchmark V5.11. http://www.tpc.org/tpcc/.

[47] THE TRANSACTION PROCESSING COUNCIL. TPC-E Benchmark V1.14. http://www.tpc.org/tpce/.

[48] THOMSON, A., DIAMOND, T., WENG, S.-C., REN, K., SHAO, P., AND ABADI, D. J. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), SIGMOD'12, ACM, pp. 1–12.

[49] TU, S., ZHENG, W., KOHLER, E., LISKOV, B., AND MADDEN, S. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), SOSP'13, ACM, pp. 18–32.

[50] UNG, G. M. Intel, micron announce new 3d xpoint memory type that's 1,000 times faster than nand. http://www.pcworld.com/article/2951864/storage/intel-micron-announce-new-3dxpoint-memory-type-thats-1000-times-faster-than-nand.html, 2015.

[51] WANG, Z., QIAN, H., CHEN, H., AND LI, J. Opportunities and pitfalls of multi-core scaling using hardware transaction memory. In *Proceedings of the 4th Asia-Pacific Workshop on Systems* (2013), APSys'13, ACM, pp. 3:1–3:7.

[52] WANG, Z., QIAN, H., LI, J., AND CHEN, H. Using restricted transactional memory to build a scalable in-memory database. In *Proceedings of the Ninth European Conference on Computer Systems* (2014), EuroSys'14, ACM, pp. 26:1–26:15.

[53] WANG, Z., YI, H., LIU, R., DONG, M., AND CHEN, H. Persistent transactional memory. *IEEE Computer Architecture Letters* (2015).

[54] WEI, X., SHI, J., CHEN, Y., CHEN, R., AND CHEN, H. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th Symposium on Operating Systems*

Principles (New York, NY, USA, 2015), SOSP '15, ACM, pp. 87–104.

[55] XIE, C., SU, C., KAPRITSOS, M., WANG, Y., YAGHMAZADEH, N., ALVISI, L., AND MAHAJAN, P. Salt: Combining ACID and BASE in a distributed database. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), OSDI'14, USENIX Association, pp. 495–509.

[56] XIE, C., SU, C., LITTLEY, C., ALVISI, L., KAPRITSOS, M., AND WANG, Y. High-performance acid via modular concurrency control. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), ACM, pp. 279–294.

[57] YUAN, Y., WANG, K., LEE, R., DING, X., AND ZHANG, X. Bcc: Reducing false aborts in optimistic concurrency control with affordable cost for in-memory databases. In *New England Database Day* (2015).

[58] ZHANG, I., SHARMA, N. K., SZEKERES, A., KRISHNAMURTHY, A., AND PORTS, D. R. K. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 263–278.

[59] ZHANG, Y., POWER, R., ZHOU, S., SOVRAN, Y., AGUILERA, M. K., AND LI, J. Transaction chains: Achieving serializability with low latency in geo-distributed storage systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), SOSP'13, ACM, pp. 276–291.

[60] ZHENG, W., TU, S., KOHLER, E., AND LISKOV, B. Fast databases with fast durability and recovery through multicore parallelism. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), OSDI'14, USENIX Association, pp. 465–477.