DrTM: Fast In-memory Transaction Processing using RDMA and HTM

Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen
Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University

Presented by Henggang Cui
RDMA: Remote direct memory access

- Cross-machine accesses with high speed, low latency, and low CPU overhead
  - Some advanced NICs
  - Direct access to the DRAM of a remote machine
  - By passing remote CPU and OS kernel
HTM: Hardware transactional memory

Locking:
```c
void deposit(account, amount){
    lock(account);
    int t = bank.get(account);
    t = t + amount;
    bank.put(account, t);
    unlock(account);
}
```

Transactional memory:
```c
void deposit(account, amount){
    atomic {
        int t = bank.get(account);
        t = t + amount;
        bank.put(account, t);
    }
}
```

- One way of synching shared memory among threads
  - No locking
  - Access and abort on conflicts
  - Can be understood as optimistic concurrency control
DrTM overview
Transaction layer

• Supporting distributed transactions
  – HTM within a single machine
  – Two-phase locking for accessing remote records
Transactions

\[\text{START}(\text{remoteWriteset}, \text{remoteReadset})\]

\[
// lock remote key and fetch value
\]

\[
\text{foreach key in remoteWriteset} \quad \text{REMOTE_WRITE}(\text{key})
\]

\[
\text{end_time} = \text{now} + \text{duration}
\]

\[
\text{foreach key in remoteReadset} \quad \text{end_time} = \text{MIN}(\text{end_time},)
\]

\[
\quad \text{REMOTE_READ}(\text{key, end_time})
\]

\[
\text{XBEGIN}() \quad // HTM TX begin
\]

\[
\text{READ}(\text{key})
\]

\[
\text{if key.is_remote() == true} \quad \text{return Read_cache[key]}
\]

\[
\text{else return LOCAL_READ(key)}
\]

\[
\text{WRITE}(\text{key, value})
\]

\[
\text{if key.is_remote() == true} \quad \text{write_cache[key] = value}
\]

\[
\text{else LOCAL_WRITE(key, value)}
\]

\[
\text{COMMIT}(\text{remoteWriteset, remoteReadset})
\]

\[
// confirm all leases are still valid
\]

\[
\text{if !VALID(end_time)}
\]

\[
\text{ABORT()} \quad // ABORT: invalid lease
\]

\[
\text{XEND}() \quad // HTM TX end
\]

\[
// write back value and unlock remote key
\]

\[
\text{foreach key in remoteWriteset} \quad \text{REMOTE_WRITE_BACK(key, write_cache[key])}
\]
Coordinate with other remote txns
Coordinate local and remote txns
Lease-based shared lock

• They use lease-based shared lock
  – To allow concurrent remote reads
  – Remote read acquires a lease
  – Local and remote write will check the leases
  – And abort itself when the lease is not expired
DrTM memory store

- In-memory key-value store for transaction layer
  - W/ highly optimized hash table based on RDMA and HTM
- They use cluster chaining, as opposed to
  - Cuckoo hashing in Pilaf
  - Hopscotch hashing in FaRM
- They do one-side RDMA for both READ and WRITE
DrTM’s cluster chaining

Cluster Chaining

Logical

Main Header

Physical

Bucket Slot

Incarnation Version

Key/64 I/32 V/32 State/64 Value/N

0 1 2 N

Entry

Type/2

Lossy Incarnation

Cached

0 1 2 N

T=11 T=10
Caching

• They cache locations instead of values
  – No need for invalidation or synchronization on cache
  – Cached entry location can be shared by threads
  – Size of cached data is smaller
Evaluation

• Experimental setup
  – 6-node cluster
  – Connected by Mellanox ConnectX-3 56Gbps IB
  – Each machine has two 10-core Intel Xeon processors and 64GB of DRAM
DrTM memory store performance

Dataset: YCSB uniform distribution

**Throughput**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrTM-KV</td>
<td>DrTM-KV with caching</td>
</tr>
<tr>
<td>DrTM-KV/$</td>
<td>DrTM-KV with caching</td>
</tr>
<tr>
<td>FaRM-KV/O</td>
<td>FaRM-KV that puts key-value pairs <strong>inside</strong> header slots</td>
</tr>
<tr>
<td>FaRM-KV/I</td>
<td>FaRM-KV that puts key-value pairs <strong>outside</strong> header slots</td>
</tr>
</tbody>
</table>

**Latency**

DrTM-KV/$: DrTM-KV with caching
FaRM-KV/I: FaRM-KV that puts key-value pairs **inside** header slots
FaRM-KV/O: FaRM-KV that puts key-value pairs **outside** header slots
DrTM memory store performance

Throughput (M ops/sec)

Cache Size (MBytes)

- Zipf/Warm
- Uniform/Warm
- Zipf/Cold
- Uniform/Cold
- No Caching
DrTM overall performance

Dataset: TPC-C

DrTM(S): logical node with 8 worker threads on each socket of a machine

DrTM is 18x faster than Calvin
(Cui: but they use different number of threads, see next slide)
DrTM overall performance

Using 6 machines

DrTM: logical node with 8 worker threads on each socket of a machine
Calvin hard-codes number of threads to 8
Conclusions

• Fast Distributed TxNs using RDMA + HTM
• HTM/RDMA friendly hash table