Trafodion Distributed Transaction Management

November, 2014
Trafodion Distributed Transaction Management

Scalable Architecture

Implemented using HBase coprocessors
Trafodion Distributed Transaction Management

Peeling the onion – Component Architecture

SQL Transaction initiating process

Transaction Manager Library
- begin/commit/abort(tx1)
- register(region, tx1)
- commitRequest/commit/abort(tx1)
- get/put/delete(tx1)
- open/perform/close scanner(tx1)

Transaction Manager
- DTM Core
- Branch I/F
- Transaction Management JNI
- TransactionManager
- TrxRegionEndpoint coprocessor client

Resource Manager Library
- Transactional operations
- Transactional Table
- Transactional Scanner
- TrxRegionEndpoint coprocessor client

HBase RegionServer
- HRegion

HBase Write Ahead Log
- Region level transaction info

Transaction Manager Log
- Meta info/state about transaction (more than one region in a txn)
- HBase table(s)

HLOG
- Each region server has its own HLOG

C++

Java

Coprocessors implemented
- TrxRegionEndpoint implements SQL transactions
- TrxRegionObserver implements recovery process
**Trafodion Distributed Transaction Management**

**BEGIN WORK – BEGINTRANSACTION**

1. SQL Transaction initiating process
   - Transaction Manager Library
     - begintransaction request
     - transid

2. Transaction Manager
   - Txn Object
   - DTM Core
   - Branch I/F
   - Transaction Management JNI
   - TransactionManager
   - Txn State
   - TrxRegionEndpoint coprocessor client

3. New transaction object allocated
   - Assign new transid

4. TransactionManager.beginTransaction(transid)

5. New TransactionState

6. Resource Manager Library
   - Transactional operations
   - Transactional Table
   - Transactional Scanner
   - TrxRegionEndpoint coprocessor client

**HBase RegionServer**

- HRegion
- HRegion
- HRegion

**transid is allocated by TM for distributed support**

**new transaction object allocated**

**Assign new transid**

**TransactionManager.beginTransaction(transid)**

**new TransactionState**
**Trafodion Distributed Transaction Management**

*get / put / delete*

---

**SQL Transaction initiating process**

1. SQL Transaction initiating process
2. HBase RegionServer
3. Transaction Manager Library
4. Resource Manager Library
5. Transaction Manager
6. TransactionManager.registerRegion which adds it to the TransactionState.participating Regions list
7. HBase RegionServer
8. TransactionalTable.put(TransactionState, region)
9. TrxRegionEndpoint.put(transid) for region
10. TrxRegionEndpoint.beginTransaction(transid) for region
11. Update region transaction object context to include puts, deletes

---

**Transaction Manager**

- Txn Object
- DTM Core
- Branch I/F
- Transaction Management JNI
- TransactionManager
- TrxRegionEndpoint coprocessor client
- Txn State

---

**Resource Manager Library**

- Transactional operations
- get/put/delete
- TrxRegionEndpoint coprocessor client
- TransactionalTable
- TransactionalScanner

---

**HBase RegionServer**

- HRegion
- HRegion
- HRegion

---

© Copyright 2014 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
Trafodion Distributed Transaction Management

**Commit Work – ENDTRANSACTION – Phase 1, prepare**

1. **SQL Transaction initiating process**
   - **ENDTRANSACTION**
   - **Transaction Manager Library**
     - **Transaction Manager**
       - **Txn Object**
       - **DTM Core**
       - **Branch I/F**
       - **Transaction Management JNI**
       - **TransactionManager**
       - **TrxRegionEndpoint coprocessor client**
     - **Txn State**

2. **endtransaction**
   - **Abort if conflict encountered**

3. **prepare_branches**
   - **HBaseTxn.prepareCommit**

4. **TransactionManager.prepareCommit**

5. **TrxRegionEndpoint.commitRequest(transid)**
   - against each region in participatingRegions list, sent in parallel

6. **Region endpoint coprocessor checks**
   - for conflicts / concurrency issue
   - If conflict, votes abort
   - else checks for writes in region
   - If no writes, votes read-only & is excluded from phase 2

7. **If writes performed**, context for transid, including all updates, is **forced** written to HLOG

8. **Votes abort, read-only, or commit to transaction manager**

9. **SQL Transaction initiating process**
   - **Transaction Manager Library**
     - **Resource Manager Library**
       - **Transaction Manager**
         - **TrxRegionEndpoint coprocessor client**
       - **Transactional get/put/delete**
       - **TrxRegionEndpoint coprocessor client**
     - **Transactional Table**
     - **Transactional Scanner**
   - **TransactionManager**
   - **TrxRegionEndpoint coprocessor client**

---

© Copyright 2014 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
Trafodion Distributed Transaction Management

**COMMIT WORK – ENDTRANSACTION – Phase 2, commit**

If at least one region voted commit & the rest voted commit or read-only

1. `commit_branches`
2. `commit_branches HBaseTxn.doCommit`
3. `TransactionManager.doCommit`
4. `forces write of committed trans state record`
5. `TrxRegionEndpoint.commit(transid)` against each region in participatingRegions list, sent in parallel
6. `Context for transid as committed txn is written unforced to HLOG`
7. `All puts and deletes are written to HBase tables from in-memory transaction state`
8. `Unforced write of forgotten trans state record`
9. `doCommit forgets transaction & returns control to TM`
10. `commit`

**SQL Transaction initiating process**

- **Transaction Manager Library**
  - `Txn Object`
  - `Transaction Management JNI`
  - `TransactionManager`
  - `TrxRegionEndpoint`
  - `coprocessor client`
  - `Txn State`

- **Resource Manager Library**
  - `Transactional operations`
  - `TrxRegionEndpoint coprocessor client`
  - `Transactional Table`
  - `Transactional Scanner`

**HBase RegionServer**

- `HRegion`
- `HLOG`

**Transaction Manager**

- `commit_branches`
- `commit_branches HBaseTxn.doCommit`
- `TransactionManager.doCommit`
- `TrxRegionEndpoint.commit(transid)` against each region in participatingRegions list, sent in parallel
- `forces write of committed trans state record`
- `Unforced write of forgotten trans state record`
- `Context for transid as committed txn is written unforced to HLOG`
- `All puts and deletes are written to HBase tables from in-memory transaction state`
### Trafodion Distributed Transaction Management

**ABORT WORK – ABORTTRANSACTION**

1. **Transaction Manager Library**
   - Abort

2. Rollback branches

3. HBaseTxn.abort

4. TransactionManager.abort

5. Does not write trans record
   - If forgotten records not written, presume abort

6. TrxRegionEndpoint.abortTransaction(transid) against each region in participatingRegions list, sent in parallel

7. HRegion.abort sets status of txn to abort
   - Removes txn from commit list if previously voted commit
   - Remove txn from pending list

8. If writes associated with txn, unforced write of abort to HLOG
   - Reduces recovery time

9. TM does forget processing
   - No writes to log
   - Abort forgets transaction & returns control to TM

10. TrxRegionEndpoint, TransactionalOperations, TrxRegionEndpoint, coprocessor client

**SQL Transaction initiating process**

- **Transaction Manager**
  - Transactional get/put/delete
  - Transactional Table
  - Transactional Scanner

- **Resource Manager Library**
  - Transactional get/put/delete
  - Transactional Table
  - Transactional Scanner

- **Transaction Manager**
  - DTM Core
  - Branch I/F
  - Transaction Management JNI
  - TransactionManager
  - TrxRegionEndpoint coprocessor client

- **HBase RegionServer**
  - HRegion
  - HRegion
  - HRegion

- **HLOG**

**Resource Manager Library**

- **Transaction Manager**
  - DTM Core
  - Branch I/F
  - Transaction Management JNI
  - TransactionManager
  - TrxRegionEndpoint coprocessor client

- **HBase RegionServer**
  - HRegion
  - HRegion
  - HRegion

- **HLOG**
### Trafodion Distributed Transaction Management

Transaction Recovery – timing of failure

<table>
<thead>
<tr>
<th>State at failure</th>
<th>Action at failure</th>
<th>Failed region(s) start up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates are in region’s memory</td>
<td>Initiate abort processing</td>
<td>No recovery needed</td>
</tr>
<tr>
<td>Prepare: Some regions complete</td>
<td>Initiate abort processing</td>
<td>If region had not completed Prepare phase before failure, no recovery needed. If it had, then abort processing initiated.</td>
</tr>
<tr>
<td>Prepare: All regions complete</td>
<td>Initiate abort processing</td>
<td>Abort processing for all regions that had updates</td>
</tr>
<tr>
<td>Commit: Some regions complete</td>
<td>Initiate commit processing</td>
<td>Apply updates from HLOG to tables if not committed</td>
</tr>
</tbody>
</table>
When failed server(s) are reintegrated, Region Server reads HLOG. If it's empty, indicating clean shutdown, it resumes txn processing. Otherwise it initiates recovery for regions.

Table names mapped to regions
Accommodates region splits & moves

1. When failed server(s) are reintegrated, Region Server reads HLOG. If it's empty, indicating clean shutdown, it resumes txn processing. Otherwise it initiates recovery for regions.

2. All HRegions build in-doubt txns list by Region Observer coprocessor.

3. Each Region with in-doubt transactions posts need for recovery to Bulletin of TM that originated the transaction if not yet posted.

4. TM builds in-doubt region list and initiates recovery for those regions.

5. Gather in-doubt transactions from in-doubt regions.

6. In-doubt region deregisters from bulletin board

7. Update txn state obj from last state of transid, & participating regions in TLOG

8. Recovery thread re-drives phase 2

9. Commit / Abort will be done as normal for in-doubt txns

10. In-doubt regions deregister from bulletin board

TM disables transactions during recovery

In-doubt transaction
1. No prepare record → abort transaction
2. Commit record → re-drive commit in redo processing
3. Abort record → do nothing
4. Prepare record → needs to know commit or abort?
Trafodion Distributed Transaction Management

**Transaction Recovery – Control Point processing**

**Used to re-drive transaction recovery**
- After server or cluster failure to ensure data consistency given in-flight operations at time of failure
- Control points written to Control Point table at a configurable interval, currently set at 2 minutes
- Re-drives of transaction commits (redo) driven by recovery needs of regions

**Used for aging**
- TLOG entries prior to 2 control points can be aged out and only two entries maintained in control point table
- Commit records rewritten to log, along with writing txn state of all running transactions, until all commits received and forgotten record written

---

1 control point interval is represented by two consecutive control point records where the first record defines the ASN from the start of the interval and the second defines the ASN at the end of the record

2 monotonically increasing value indicating the audit write sequence number
**Trafodion Distributed Transaction Management**

**Addressing the requirements**

**Current State**
- Use HBase Endpoint coprocessors
- Trafodion has Transaction Manager (TM) processes, written in C++, supporting:
  - Recovery on client / region server crash
  - Transactions involving multiple HBase clients (transaction propagation)
- Clients can use a new, transactional client derived from regular HBase client
- Trafodion as single jar file, used by region server, TM & clients. Compatible with HBase 0.98 version.
- Recovery from region server

**Goals**
- Optional, no penalty if not used
- Very low overhead – tight integration with the region server helps
- Avoid additional processes
- Avoid non-Java code
- Avoid version incompatibilities

**How to achieve goals**
- Provide Java implementation of TM code for the following functionality, if needed:
  - Txn propagation
  - Manageability
  - Recovery
Trafodion Distributed Transaction Management

Future opportunities

Optimizations
For single region transactions
Transaction flow

Isolation support
Repeatable read
Snapshot isolation
Serialized snapshot isolation

Row Locking paradigm for certain tables
Tables involved in long running transactions
Pessimistic locking for highly concurrent operations

High Availability
Recovery from TM, or node failure
Accommodate Region Splitting & Balancing

Manageability
Transaction object info/metrics via REST APIs
Thank you