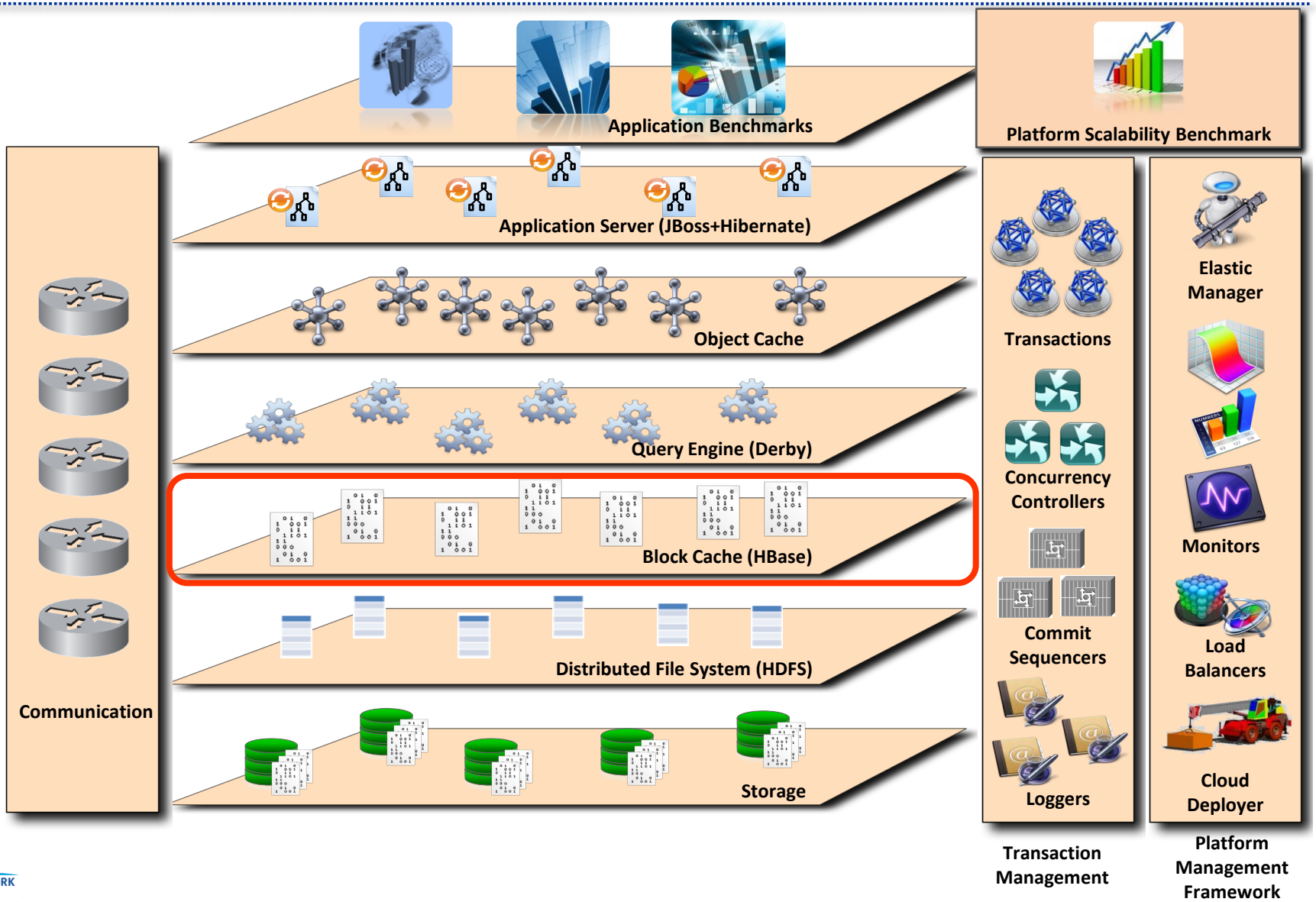
Review Meeting
Brussels
November 2013

CumuloNimbo Block Cache

McGill University

SEVENTH FRAMEWORK
PROGRAMME

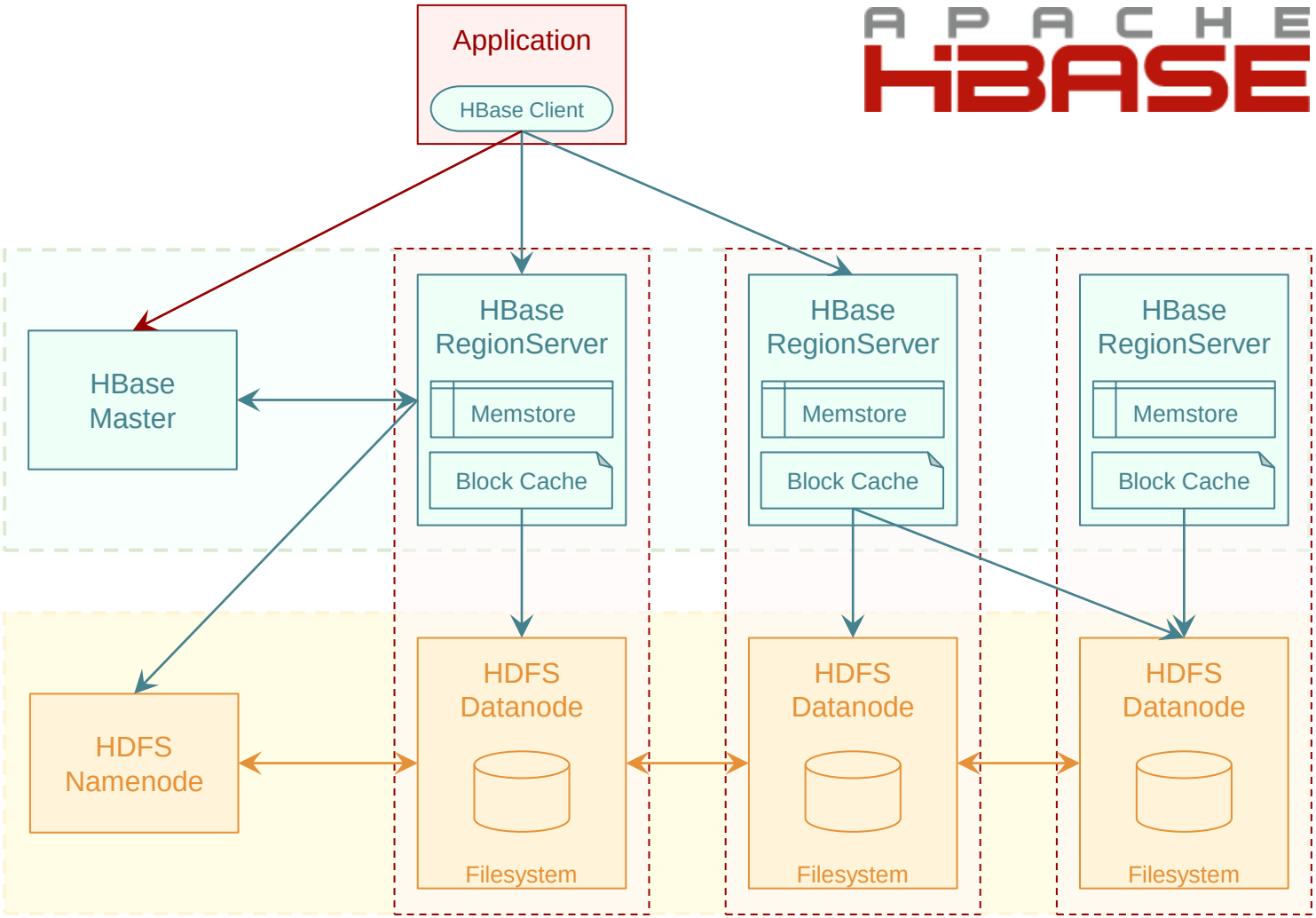
Architecture



Block Cache based on HBase

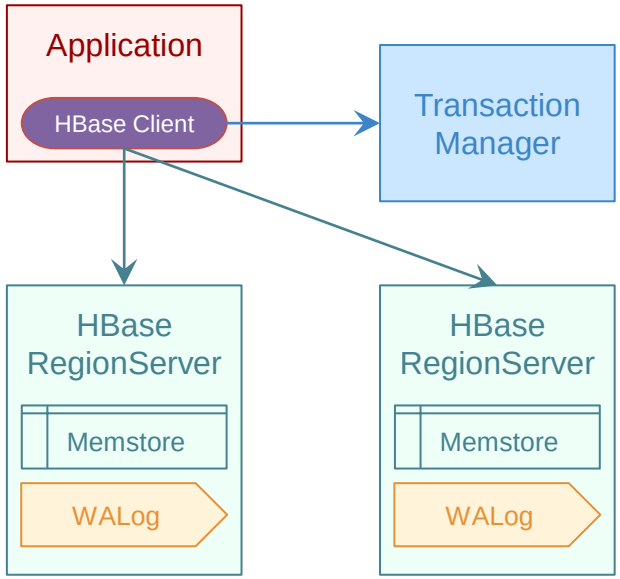
- Already provided
  - Tuple Interface
  - Partitioning
  - Elasticity
- To be adjusted
  - Transactions
    - Lightweight Transaction Support within HBase client
    - Coordination with Transaction Manager
  - Holistic Recovery
  - Indexing

Hbase Architecture



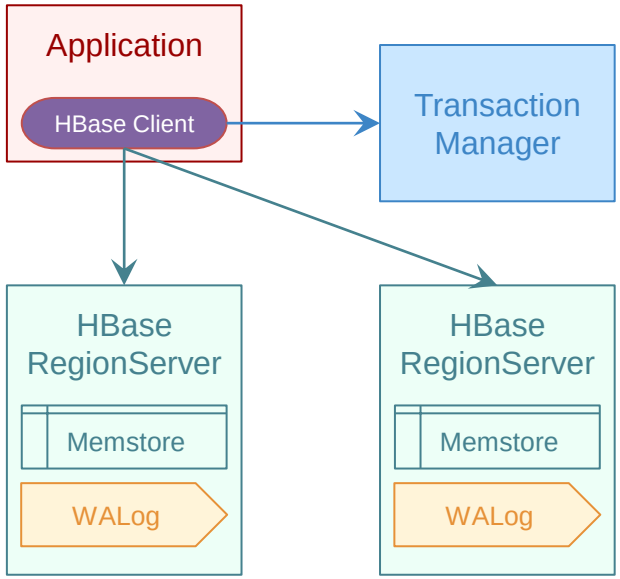
SHBase: Transactional Support

- Transactional interface
  - begin - Obtain new context from TM
    - Snapshot Isolation ; MVCC
  - read - Read from your own snapshot
  - write - Defer updates until commit
  - commit a) Check with TM
    b) Flush & Persist write-set
  - abort - Discard write-set



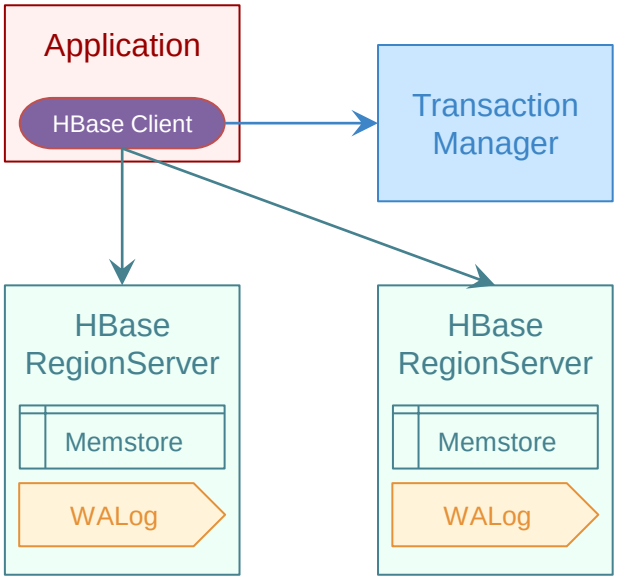
SHBase: Transactional Support

- Transactional interface
  - begin - Obtain new context from TM
    - Snapshot Isolation ; MVCC
  - read - Read from your own snapshot
  - write - Defer updates until commit
  - commit a) Check with TM
    b) Flush & Persist write-set
  - abort - Discard write-set



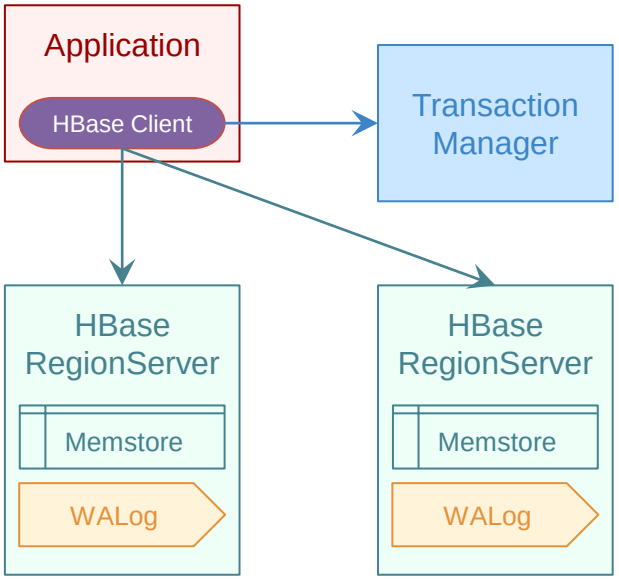
SHBase: Transactional Support

- Transactional interface
  - **begin** - Obtain new context from TM
    - Snapshot Isolation ; MVCC
  - **read** - Read from your own snapshot
  - **write** - Defer updates until commit
  - **commit** a) Check with TM
    b) Flush & Persist write-set
  - **abort** - Discard write-set



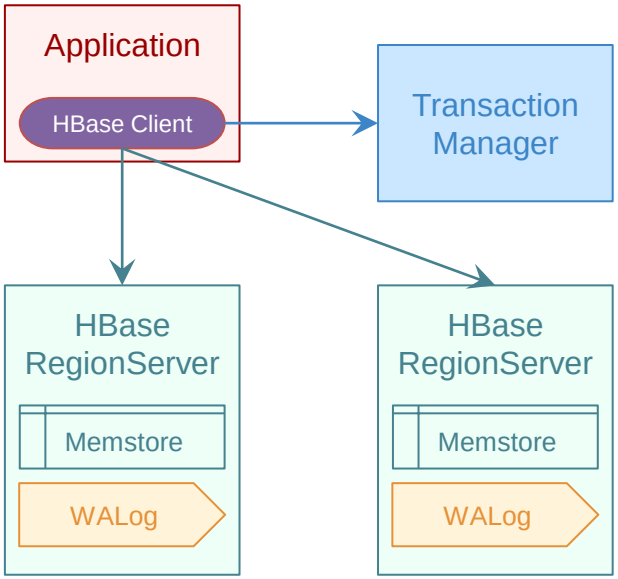
SHBase: Transactional Support

- Transactional interface
  - `begin` - Obtain new context from TM
    - Snapshot Isolation ; MVCC
  - `read` - Read from your own snapshot
  - `write` - Defer updates until commit
  - `commit` a) Check with TM
    b) Flush & Persist write-set
  - `abort` - Discard write-set



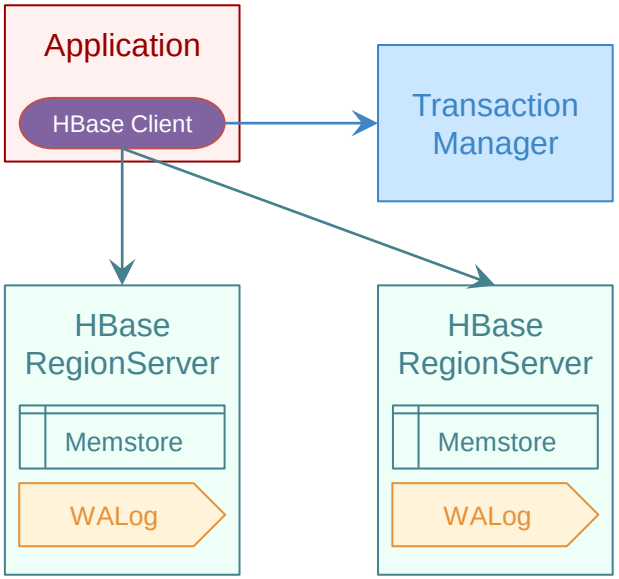
SHBase: Transactional Support

- Transactional interface
  - `begin` - Obtain new context from TM
    - Snapshot Isolation ; MVCC
  - `read` - Read from your own snapshot
  - `write` - Defer updates until commit
  - `commit` a) Check with TM
    b) Flush & Persist write-set
  - `abort` - Discard write-set



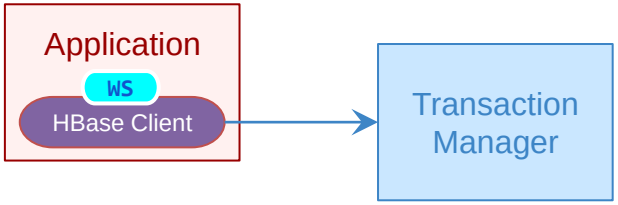
SHBase: Transactional Support

- Transactional interface
  - `begin` - Obtain new context from TM
    - Snapshot Isolation ; MVCC
  - `read` - Read from your own snapshot
  - `write` - Defer updates until commit
  - `commit` a) Check with TM
    b) Flush & Persist write-set
  - `abort` - Discard write-set



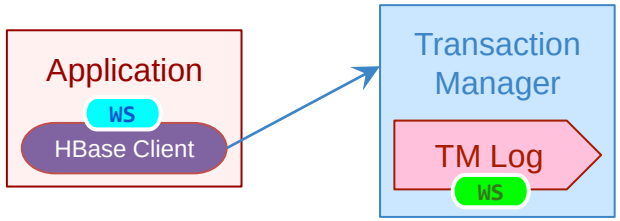
Life Cycle

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



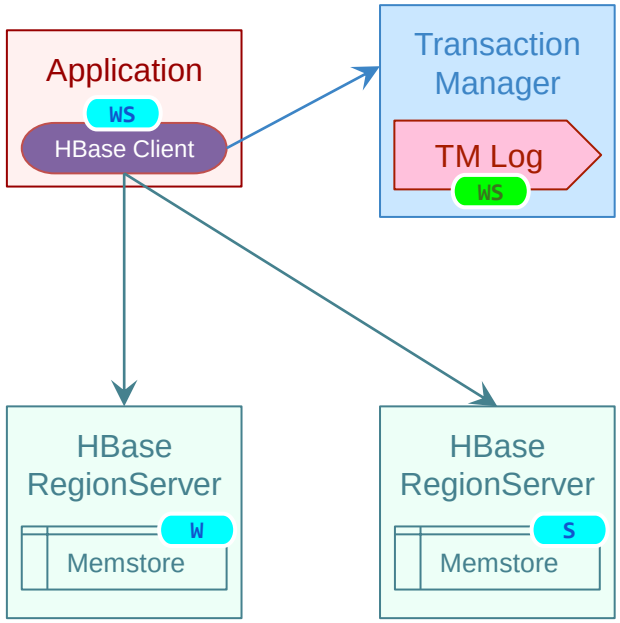
Life Cycle

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



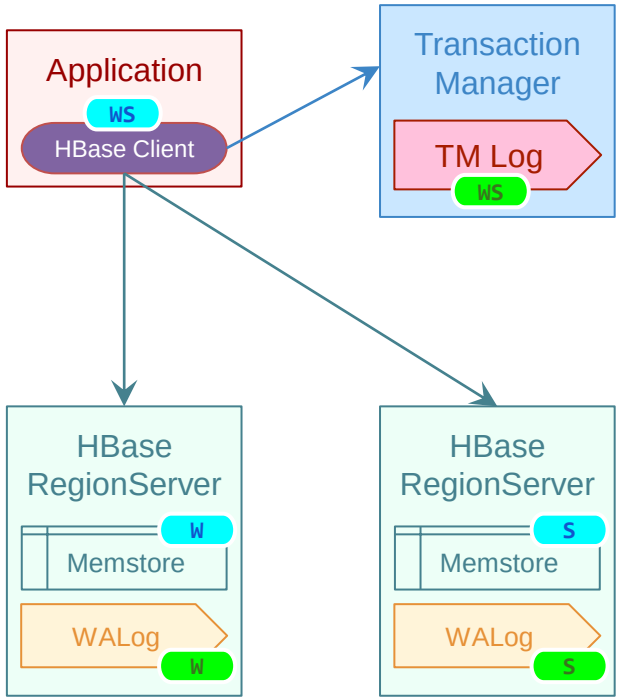
Life Cycle

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



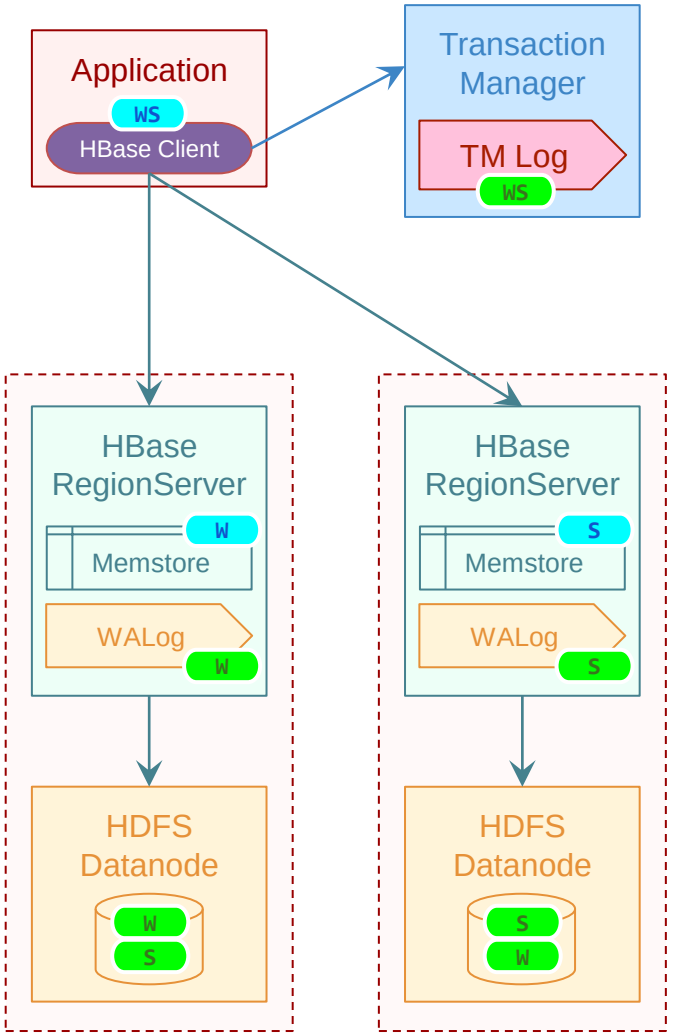
Life Cycle

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



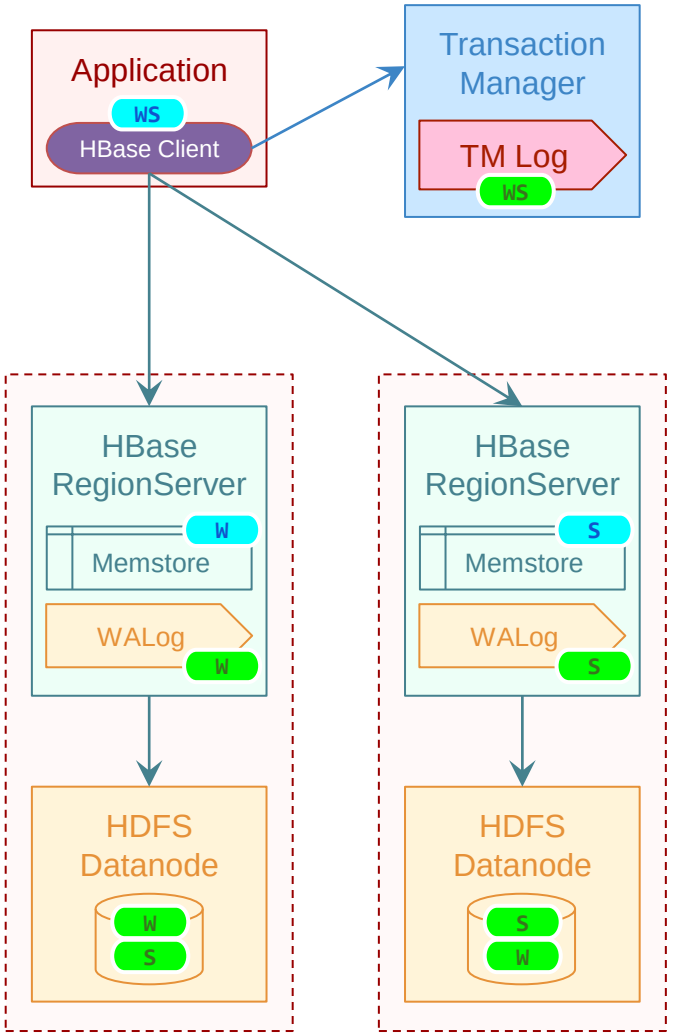
Life Cycle

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



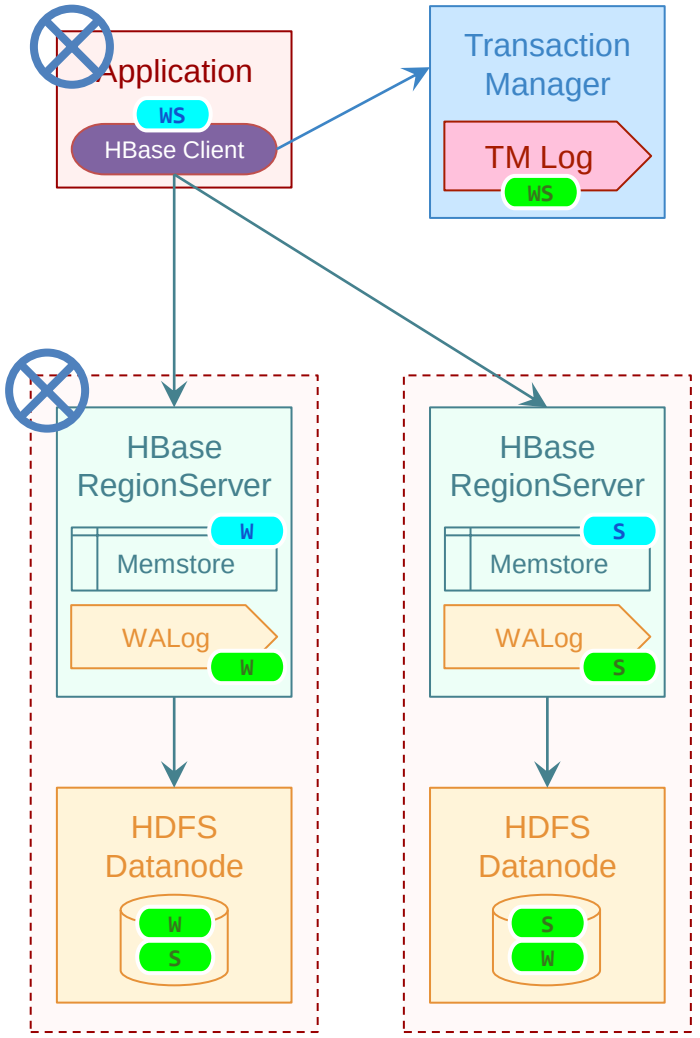
Life Cycle

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



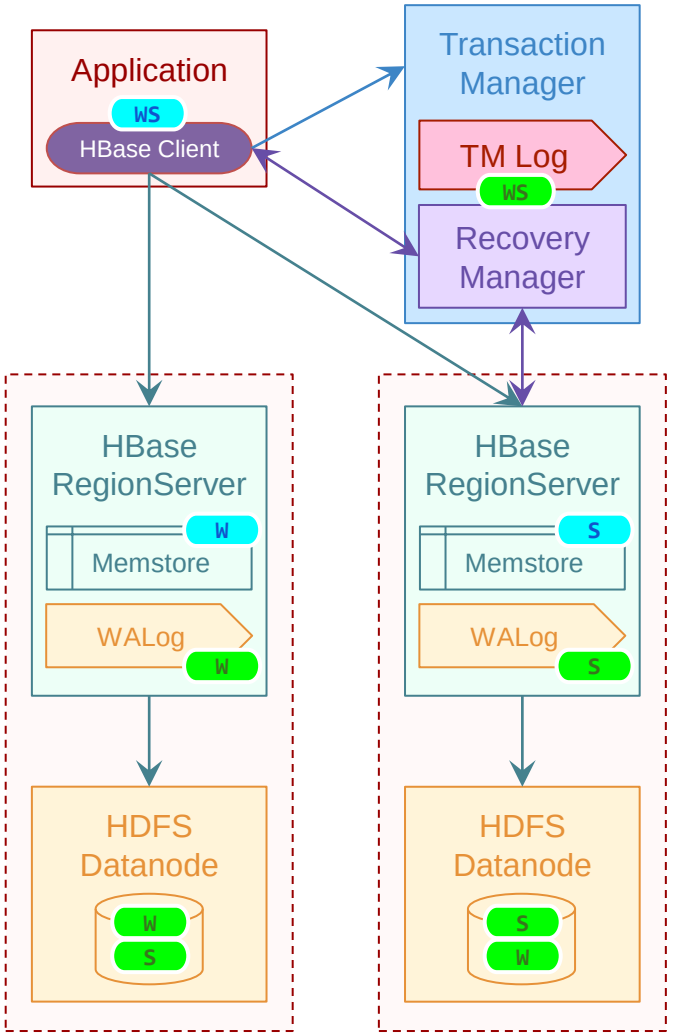
Failure Handling

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Performance
  - Asynchronous persistence
  - Eventual persistence
- Reliability
  - Client failure
  - Server failure



Recovery

- Transaction lifecycle
  - Executing
  - Committed / Aborted
  - Flushed
  - Persisted
- Recovery management
  - Tracking (checkpointing)
  - Failure detection
  - Failure recovery



CUMULO NIMBO
FP7-257993

Recovery last year

- Complete Redesign
- Minimal information kept
- Standardized implementation according to CumuloNimbo practices

SEVENTH FRAMEWORK PROGRAMME

Recovery: Failure Points

- At SHBase **Client**:
  - After log / before flushing to HBase
  - All transactions that haven't been flushed before crash need REPLAY
- At Hbase
  - After receiving the write-set before persisting to HDFS
  - All transactions that haven't been persisted before crash need REPLAY

Recovery: Failure Detection

- Recovery Manager
  - Monitor state messages
    - Clients
    - Region Servers
  - If we don't receive state messages for some time, then the monitored component is declared FAILED

Recovery Manager

- Recovery Manager
  - Tracking transactions and flushes
- Client and Server send state messages on regular intervals
  - Loosely related to checkpointing
- Keeps track of
  - Txn with smallest idea so that no txn with smaller commit timestamp needs replay

State messages

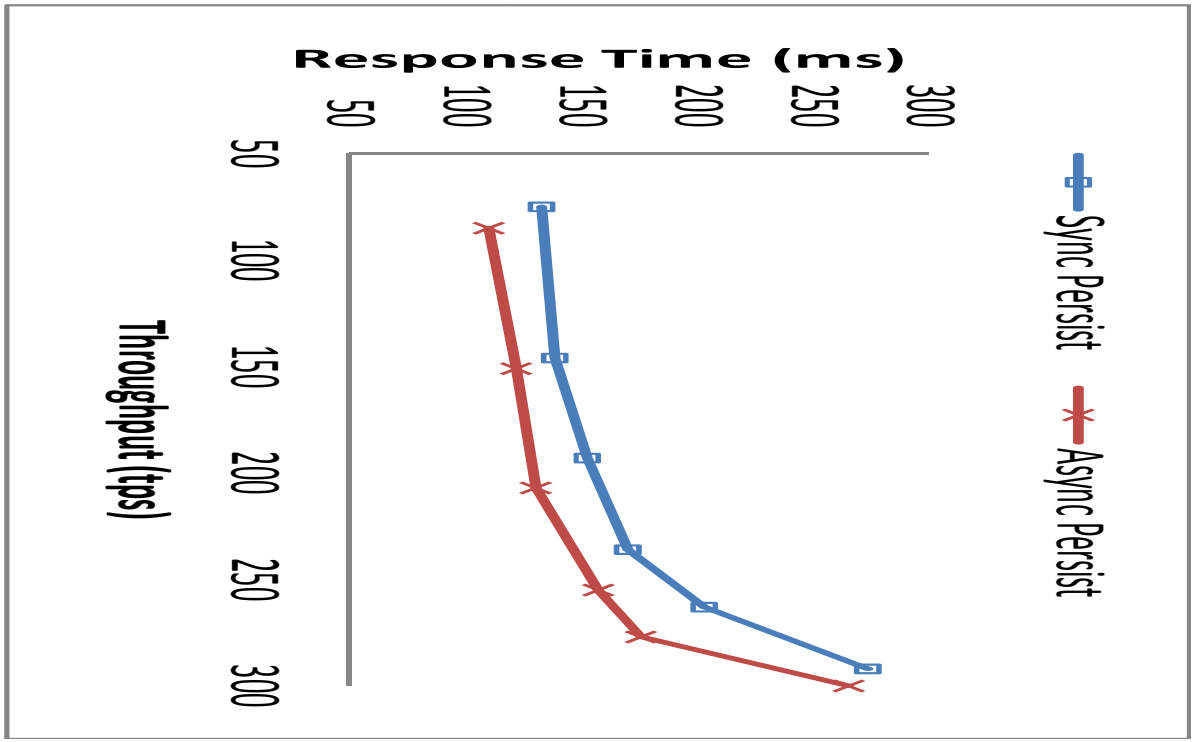
- At client/server
  - Local Tmin: all txn with CT < Tmin are flushed/persistet
  - Send local Tmin to Recovery message in heartbeat
- Recovery Manager
  - Tmin-flushed: min over all Tmin of clients
  - Tmin-persisted: min over all Tmin of servers
- Challenge:
  - How does server know that it will not receive from client a transaction with smaller CT
  - Recovery manager sends Tmin-flushed to region servers

Recovery

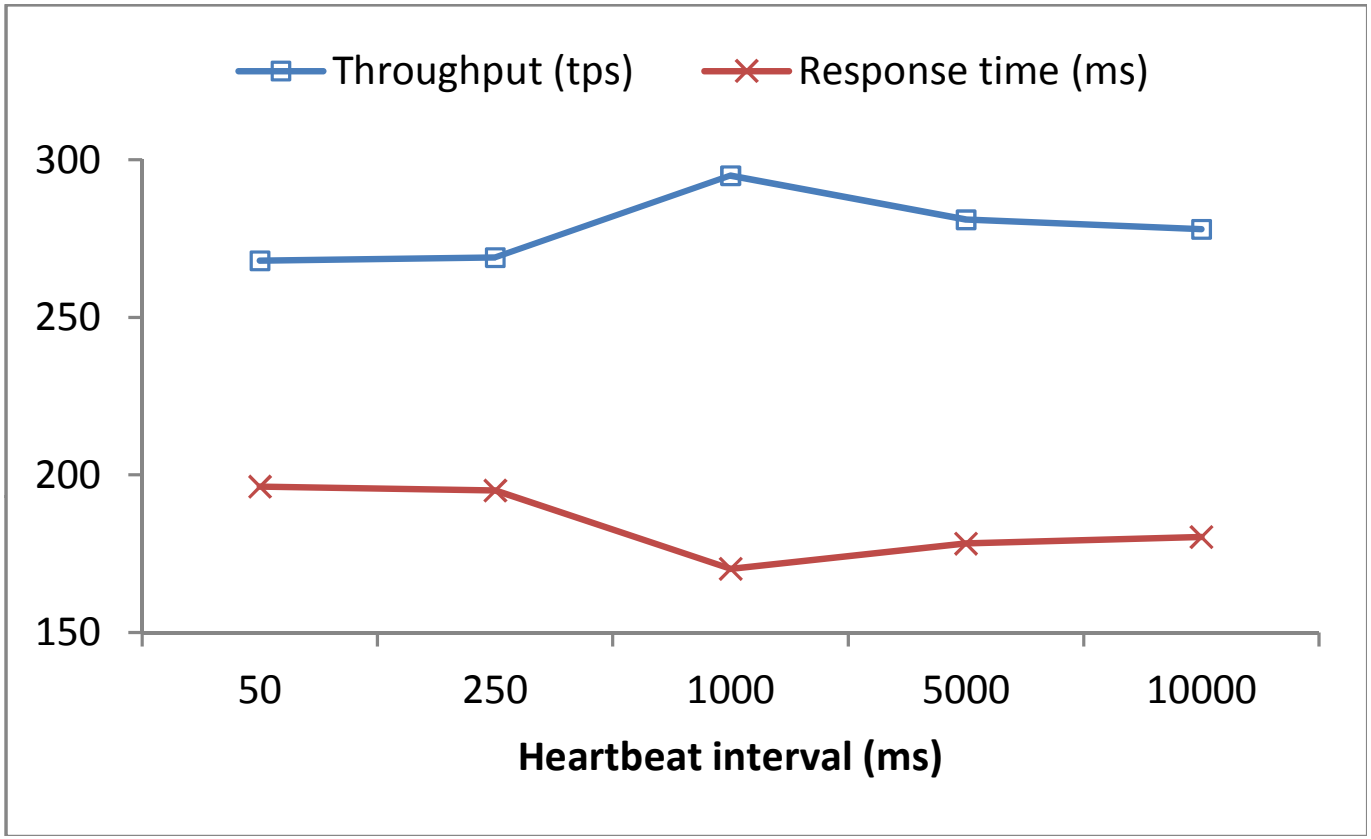
- Simply use log to replay
- Special client that uses same transaction CT as seen in the log

- For server recovery:
  - Regions are taken by other servers (provided by Hbase)
  - Perform HBase recovery
  - Then only replay writes that belong to failed server
  - Tmin of recovering server has to be reset

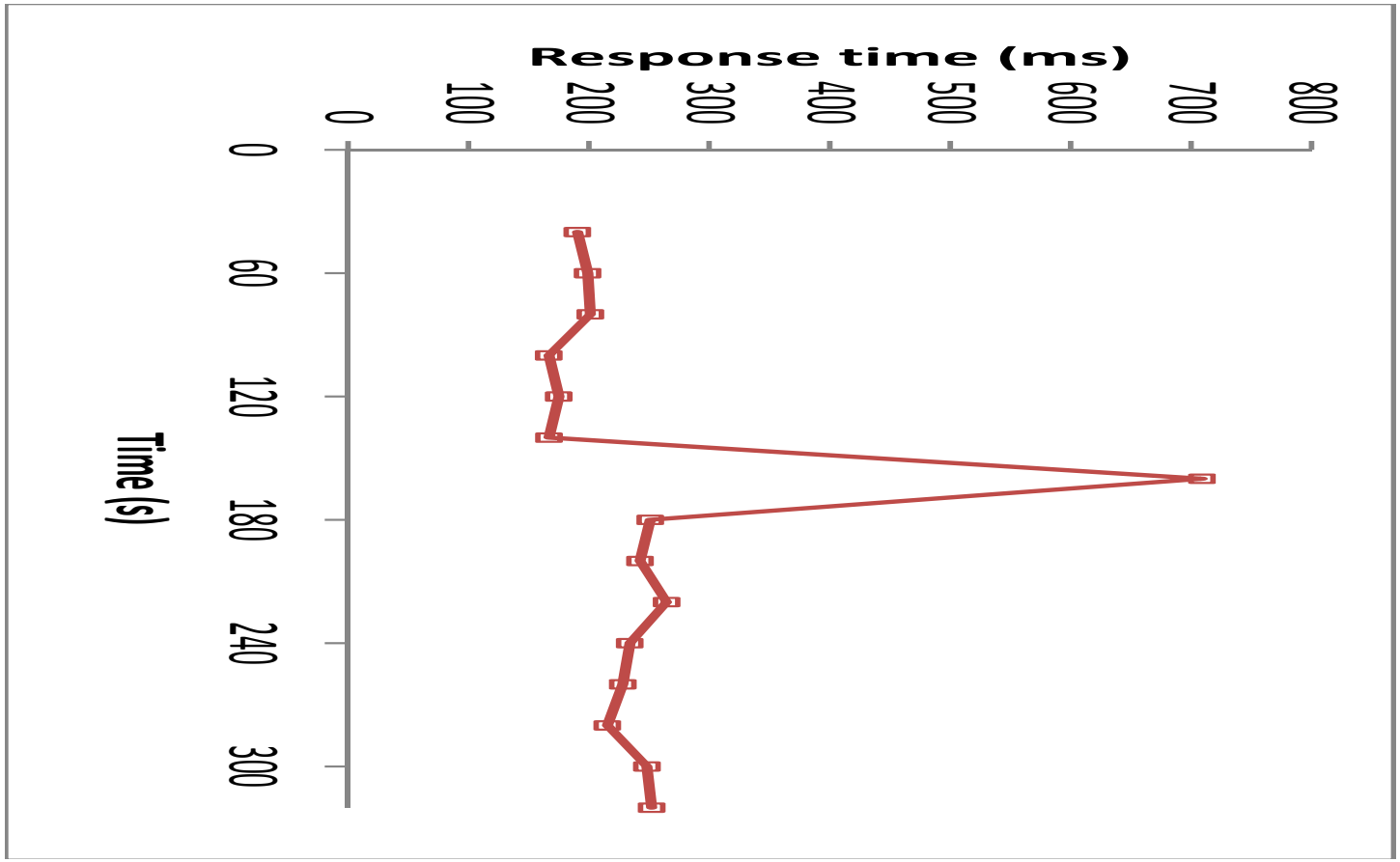
Performance: Throughput



Performance: Heartbeat



Recovery


Response time (ms)
0
100
200
300
400
500
600
700
800
Time (s)
0
60
120
180
240
300

CUMULO NIMBO
FP7-257993

Monitoring

- To PMF
  - Get, Put, Delete: count, time
  - Scan, Multiget: count, size, time
  - Conflicts, Aborts, Commits: count, time, rate
  - Txns, ROTxns, RWTxns: count, time
  - We provide both windowed metrics (window = metrics polling period) and total metrics (over the life of the component's jvm)

- Low level analysis
  - Per operation analysis group by txnid
  - Per operation analysis group by table

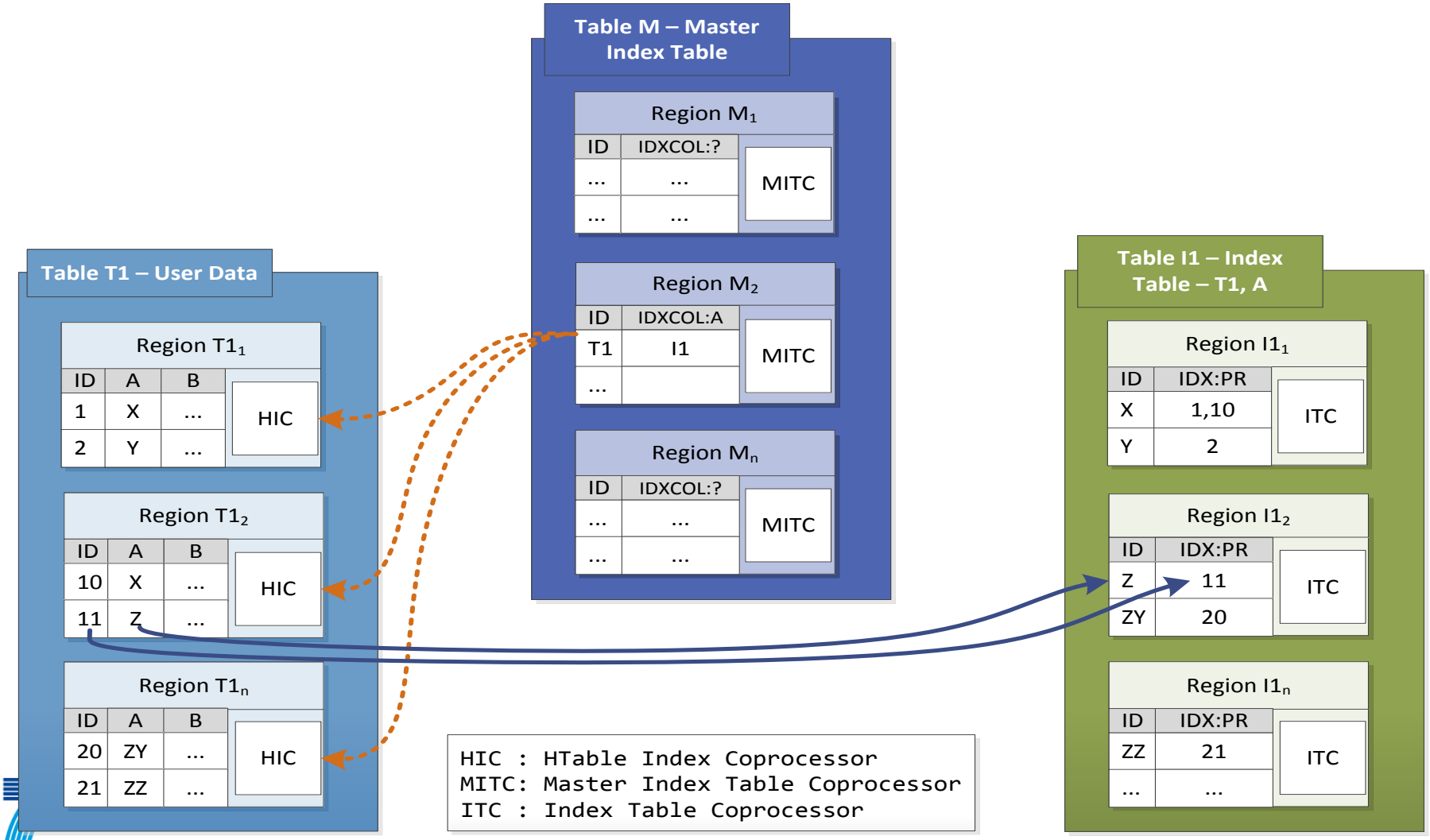
Indexing within HBase

- Feasibility Study whether possible at HBase level
- Two prototypes
  - As HBase tables (just as Derby)
  - Special main-memory only structure within Co-processors
- All functionality within Co-processors (no change to HBase itself)

Functionality

- DDL
  - Create Index (on empty table / table that contains already data)
  - Drop Index
- DML
  - Put (co-processor wrappers)
  - Get with index

Within HBase tables



Performance: Inserts

Results
Insertions – Not Batched
Different Region Servers - Throughput

No Index 200K Rows	Indexed 200K Rows*	Indexed 200K Rows (5%)	Indexed 200K Rows (7%)	Indexed 200K Rows (10%)
1653 ops/s	1323 ops/s	1301 ops/s	1310 ops/s	1326 ops/s
100%	80.0%	78.7%	79.2%	80.2%

* Uniformly distributed from a 2782 word list (~1.4%)

Performance: gets

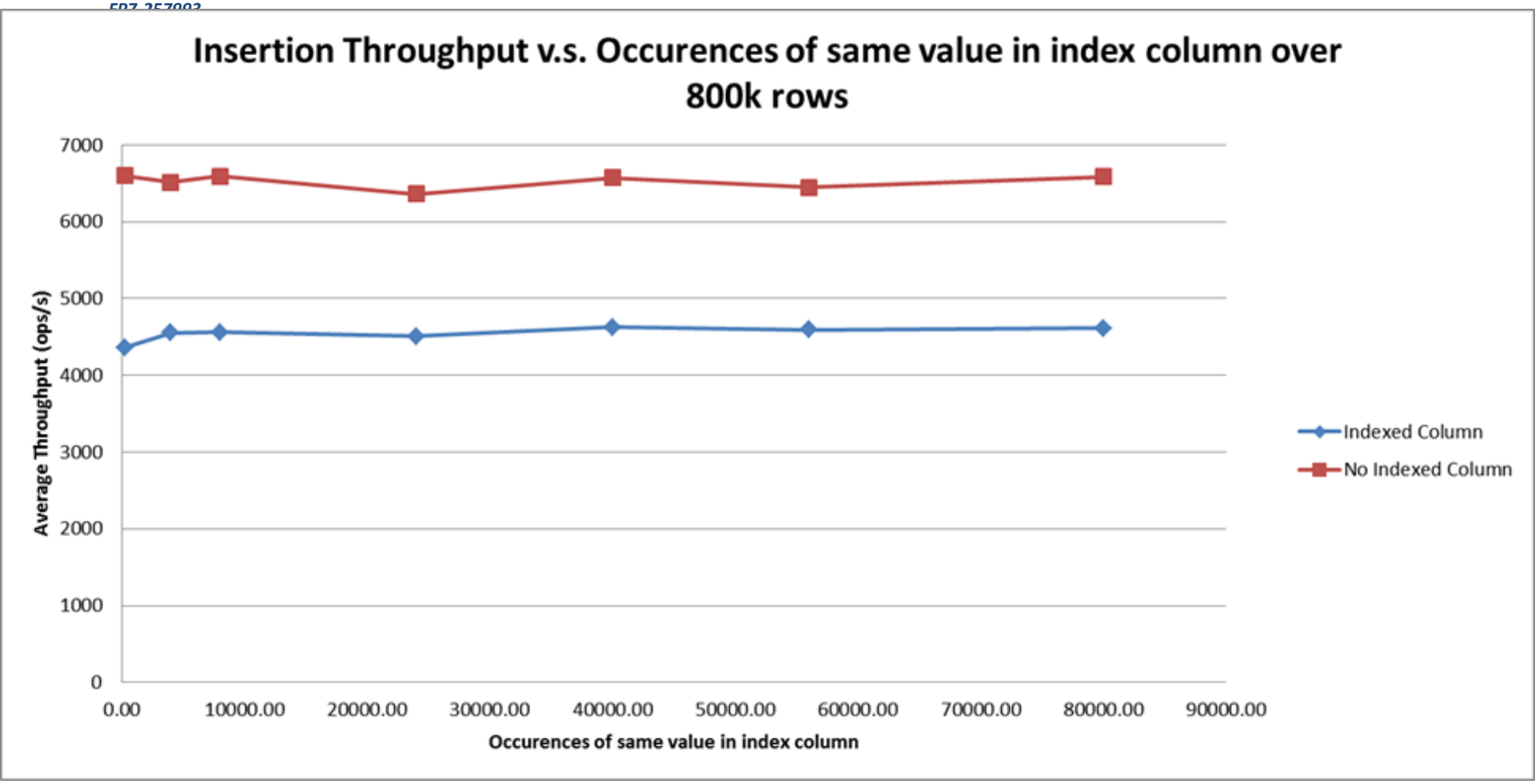
Results - Queries - 200K rows

Queried value occurs	Filtered Scan	Get By Index	Gain
In 5% of rows	3.50 seconds	1.22 seconds	286.9%
In 7% of rows	4.45 seconds	1.63 seconds	273%
In 10% of rows	5.88 seconds	2.15 seconds	273.5%
0 times	0.63 seconds	0.05 seconds	1260%
1 time	0.64 seconds	0.06 seconds	1067%

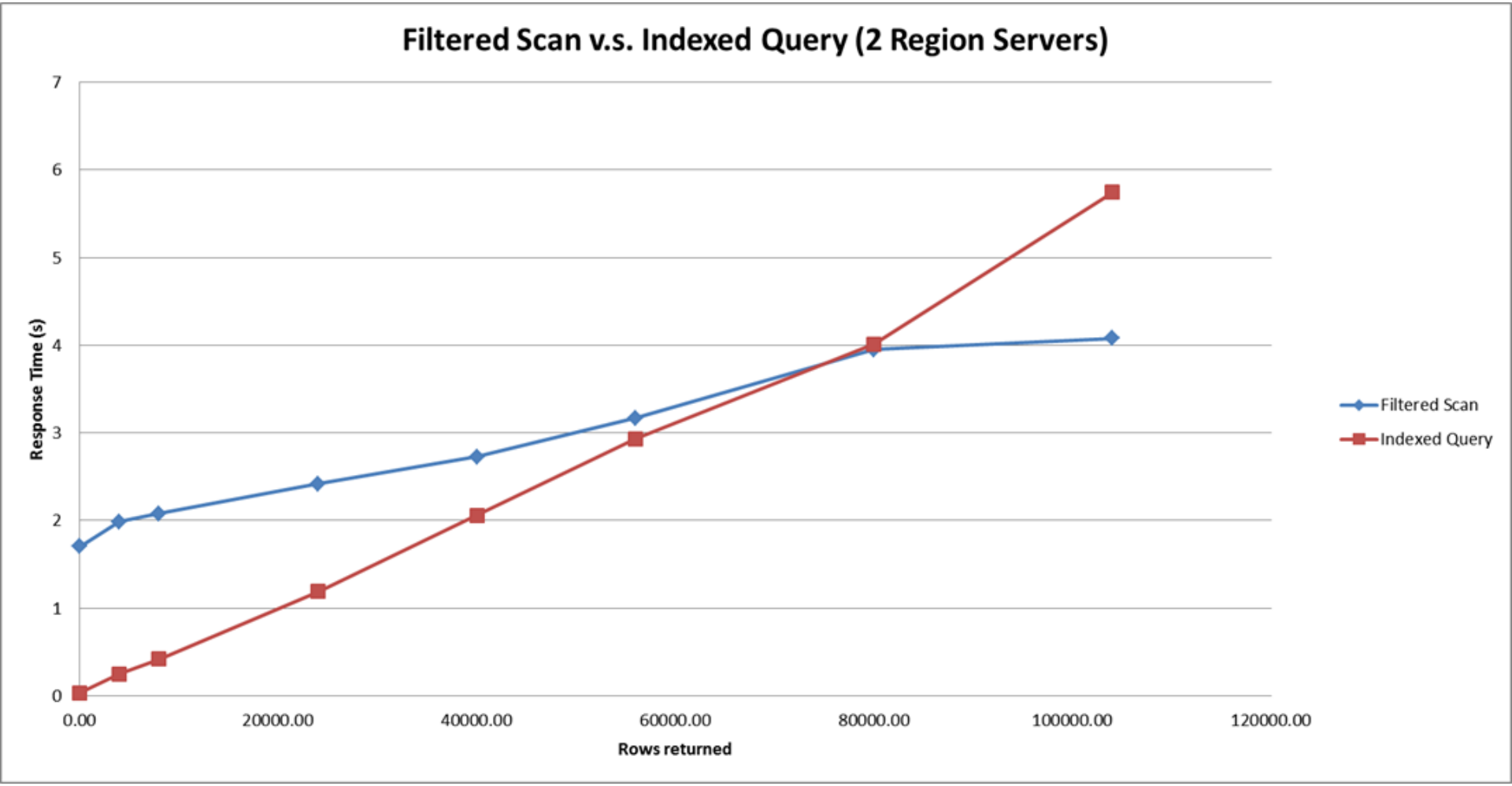
Within Co-processors

- Index data is at same location as Table Data
  - Index region correlated with table region
  - No remote access
- Splits more challenging
- Only main memory: upon recovery recreation

Performance Inserts



Performance: gets



CUMULO NIMBO
FP7-257993

Extension plans

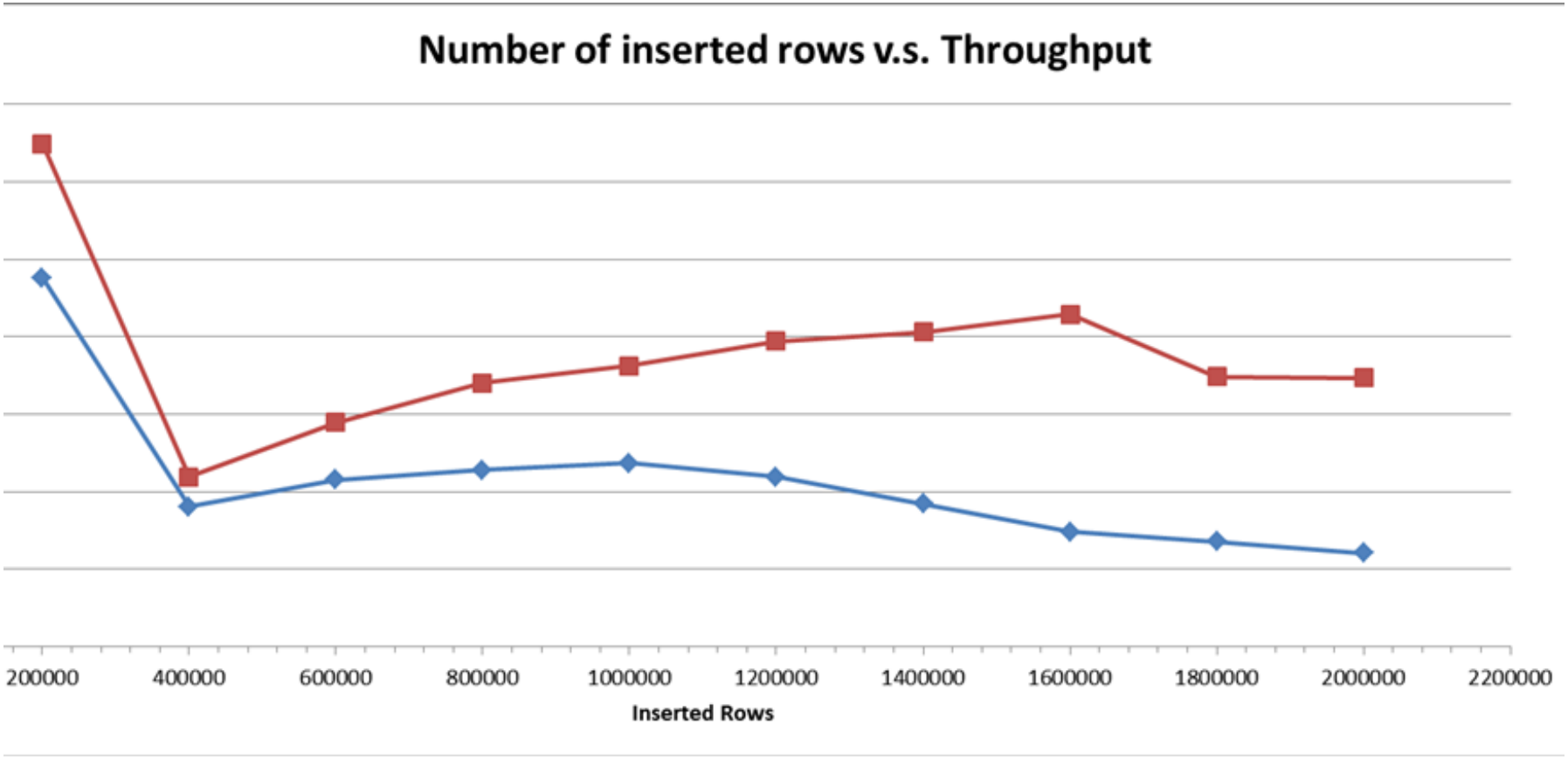
- Range Query support
- Comparison with Derby Indexing

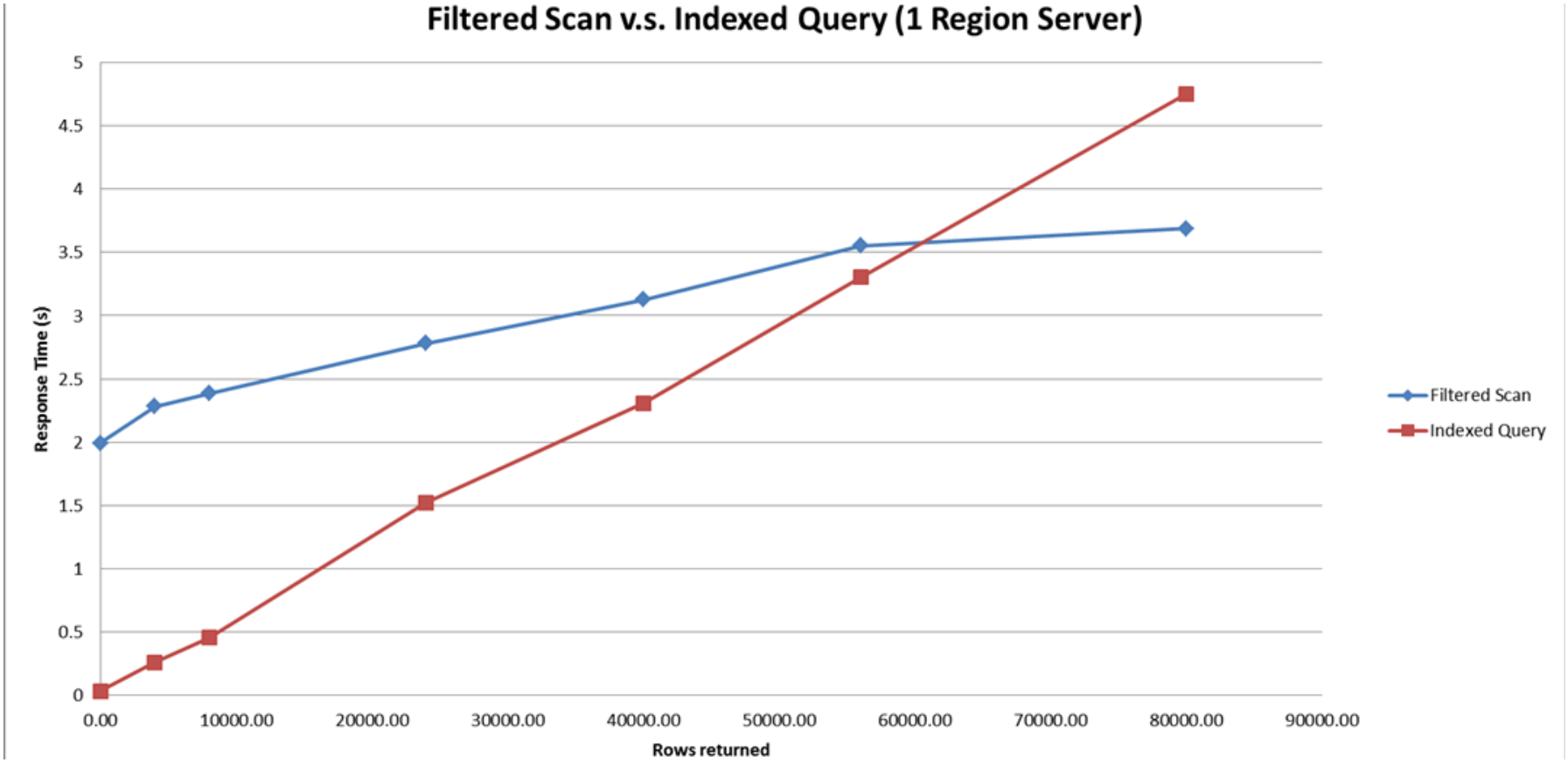
SEVENTH FRAMEWORK PROGRAMME

Results
Insertions – Batched
Different Region Servers - Throughput

No Index 1M Rows	Indexed 200K Rows*	Indexed 200K Rows (5%)	Indexed 200K Rows (7%)	Indexed 200K Rows (10%)
12908 ops/s	1323 ops/s	1301 ops/s	1310 ops/s	1326 ops/s
100%	10.3%	10.1%	10.1%	10.3%

* Uniformly distributed from a 2782 word list (~1.4%)


Number of inserted rows v.s. Throughput
200000
400000
600000
800000
1000000
1200000
1400000
1600000
1800000
2000000
2200000
Inserted Rows


Filtered Scan v.s. Indexed Query (1 Region Server)
Response Time (s)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0.00
10000.00
20000.00
30000.00
40000.00
50000.00
60000.00
70000.00
80000.00
90000.00
Rows returned
Filtered Scan
Indexed Query