Rethinking Main Memory OLTP Recovery

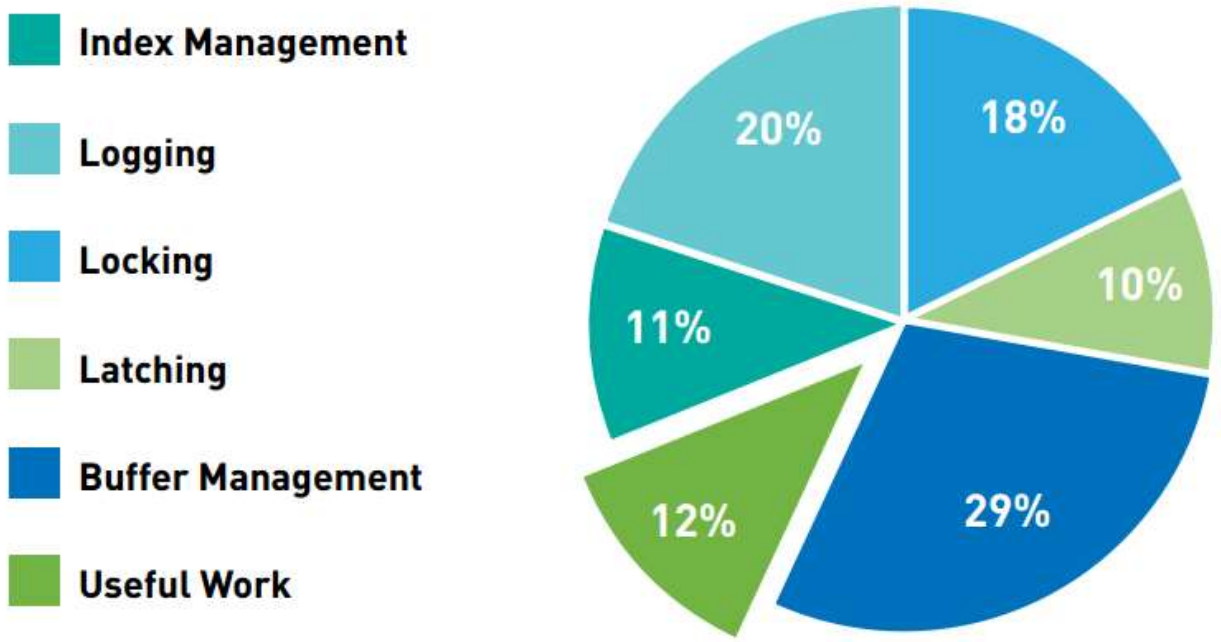
Nirmesh Malviya, Ariel Weisberg, Samuel Madden, Michael Stonebraker

Presented by: Rafi Alam

Agenda

- Introduction to Volt DB
- Command Logging
- ARIES style physiological logging using main memory
- Evaluation
- Conclusion
- Comments

Introduction to Volt DB



How VoltDB handles it

- Partitioning and distribution across virtual nodes, shared-nothing cluster.
- In memory DB for maximum throughput and eliminates the need for buffer management.
- Each single-threaded partition operates autonomously, eliminating the need for locking and latching.
- Data is automatically replicated for intra- and inter-cluster high availability.
- Stored procedure interface for transactions.

Transactions

- Each stored procedure is defined as a transaction. The stored procedure succeeds or rolls back as a whole, ensuring database consistency
- Single-partitioned
- Uses serialized processing
- If a procedure does require data from multiple partitions, one node acts as a coordinator

Transactions

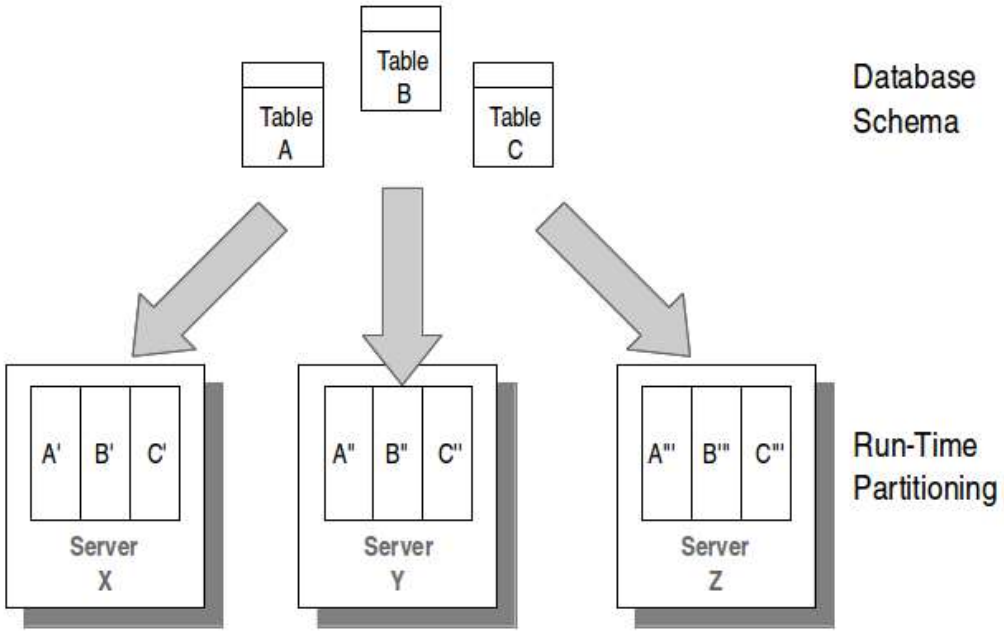
- For Global Transaction Ordering and Replication- initiator generates unique timestamp based transaction-ids
- At each site, transactions received from an initiator are placed in a special priority queue
- For global ordering, this is done by checking if the id of a transaction in the queue is the minimum across prior transactions received from all initiators

Volt DB: Partitioning

- Table is horizontally partitioned on keys.
- Each partition resides in the main memory of a cluster node
- It is replicated across several nodes for high availability
- For better execution, each transaction should be single sited

Partitioning

Figure 1.1. Partitioning Tables

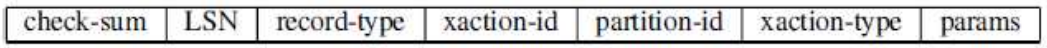


Durability and Async Checkpoints

- For single node failures, replicas ensure availability of DB
- Maintains in-memory undo log for rolling back mid-execution. This is not written to disk.
- Checkpoint mechanism does periodic writes to disk
- Snapshots are created by scanning every row
- 3 bits track each row for changes
- Background process writes the snapshot to disk after applying final changes asynchronously

Command Logging

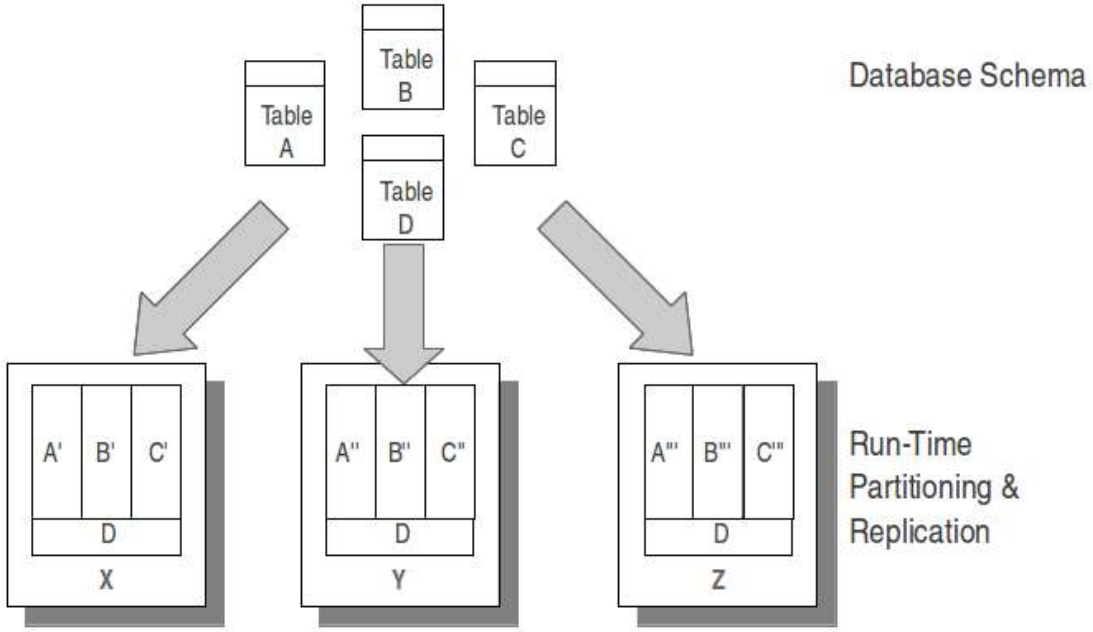
- Write ahead logging approach
- Log entry - (transaction-name, parameter-values).
- Writing log
  - Single-partition transaction
  - Multi partition- done by coordinator node (node with smallest id where tranx was initiated)
  - if replicas are present, the transaction is also logged at all replicas of the site



Partitioned vs. Replicated Tables

- Tables are partitioned in VoltDB based on a column
- The onus is on developer to make intelligent partitions
- VoltDB allows certain database tables to be replicated to all partitions of the cluster.
- E.g. Small read only tables

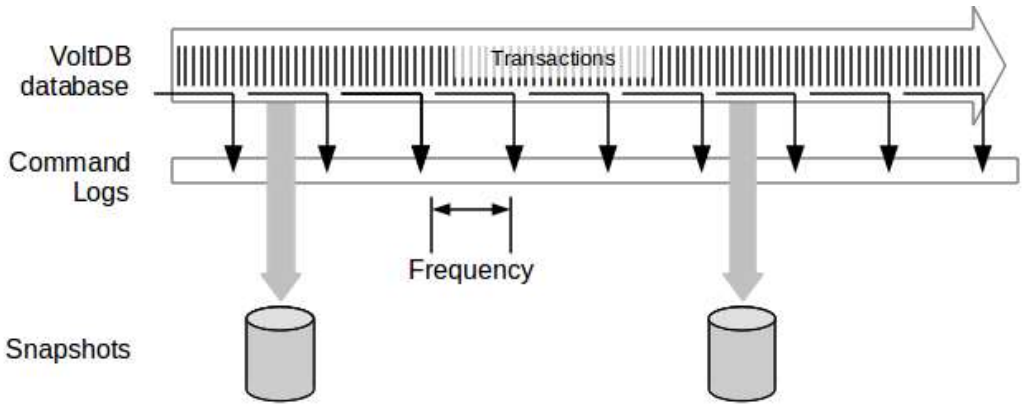
Replicating tables



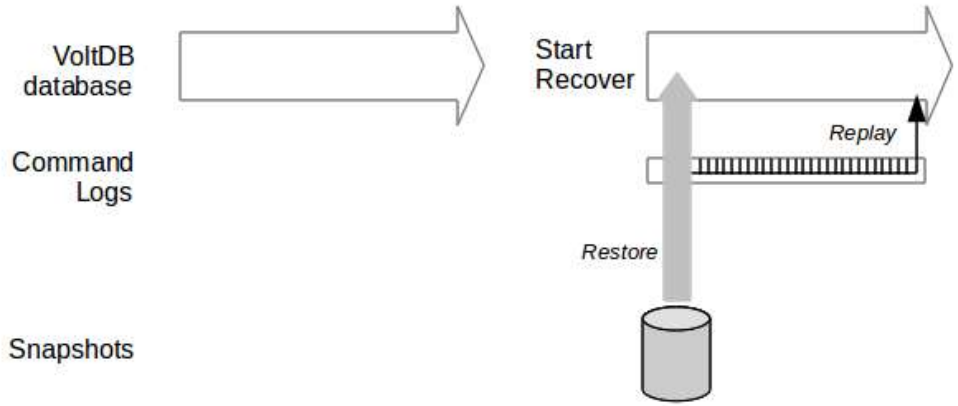
Recovery

- Using the latest database snapshot on disk, database contents are initialized in memory
- Indexes are rebuilt as disk snapshots doesn’t store it.
- Shared command log are read and ordered for recovery
- In case of database crash before transaction finishes but write ahead log has the transaction, the user doesn’t get a notification of success

Logging


VoltDB
database
Transactions
Command
Logs
Frequency
Snapshots

Recovery


VoltDB
database
Command
Logs
Snapshots
Start
Recover
Replay
Restore

Physiological logging in main memory

- It includes an analysis pass, a physical REDO pass and a logical UNDO pass.
- Initial approach- CRUD operations using disk maintaining a dirty page table and transaction table.
- Main memory approach – All operation ca be accessed by probing main memory

(page #, slot#) → (table-id, primary-key)

Optimizations

- Checkpointing: Use transaction-consistent checkpointing - only updates from committed transactions are made persistent.
- Log/node: Having a shared log for all sites as opposed to a log per-execution site makes recovery simpler
- Batched writes: synchronous, per transaction rather than per update based

Checksum	LSN	Record-type	Insert/Update/ Delete	Transaction-id	Partition-id	Table Name	Primary Key	Modified Column List	Before Image	After Image

Recovery in Physiological logging

- An analysis phase, a redo phase and an undo phase.
- Analysis : identify the LSN from which log replay should start.
- Redo : reads every log entry starting from this LSN and reapplies updates in the order the log entries appear.
- Use (table-name, primary-key) to identify log record and apply changes

  different partitions can be recovered parallel using log records.
- Undo pass: transactions which had not committed at the time of the crash; use the before image of the data record to undo the update
- In VoltDB , undo pass can be eliminated altogether as only the transaction executing at the time of crash will need to be rolled back

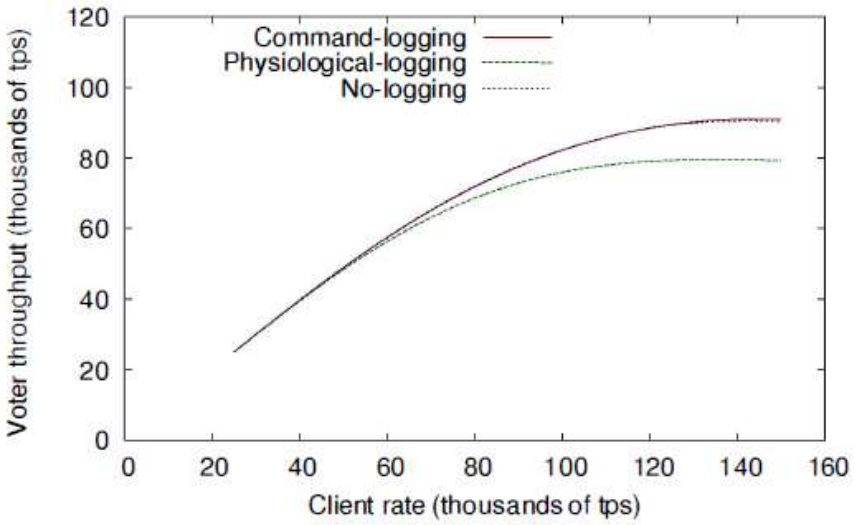
Performance evaluation

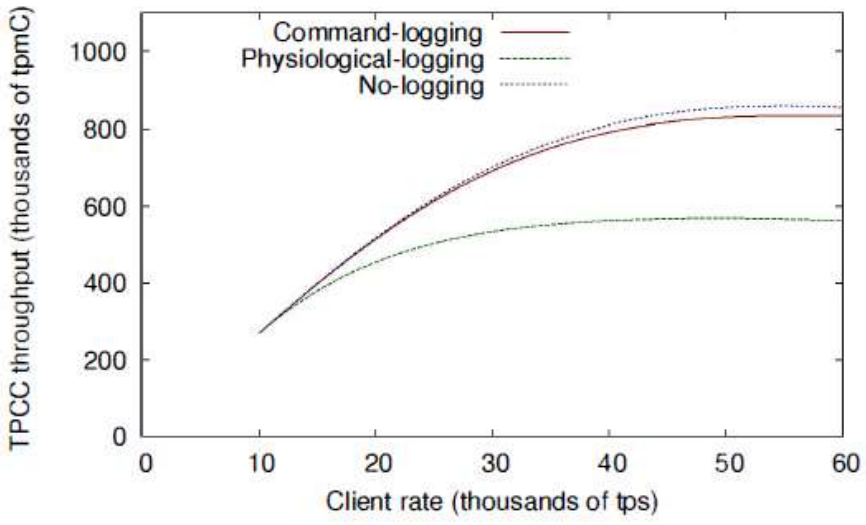
- Implemented command logging and physiological logging in VoltDB
- Synchronous logging
- Voter and TPC-C benchmark
- Voter – very simple with just one stored procedure.
- Intel Xeon dual-socket 2. 4 GHz 8-core server with 24GB of RAM, 12 TB of hard disk
- For distributed transactions- cluster of four identical machines- Intel Xeon dual-socket 12-core server box with a processor speed of 2. 4GHz, 48GB of RAM, 4 TB of hard disk

Set up

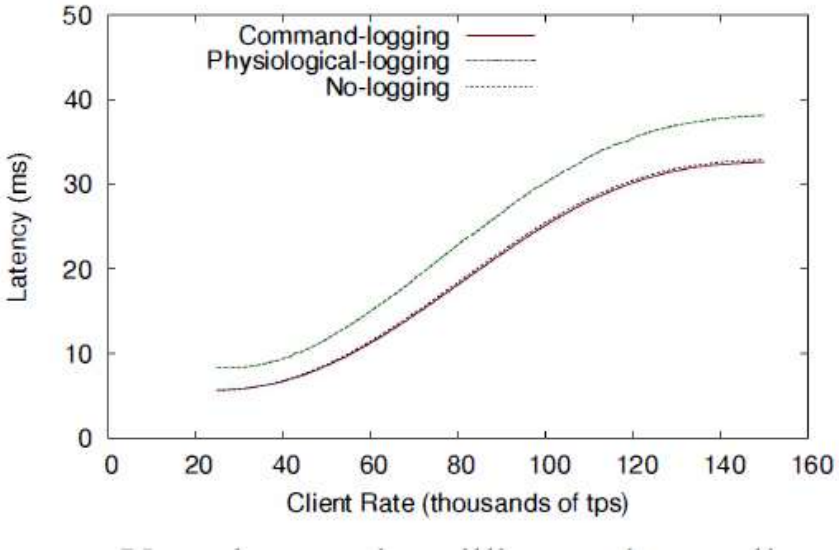
- For an 8-core machine, running six sites works best for the Voter benchmark
- For the TPC-C benchmark, best performance is achieved by using all possible sites (one per core)

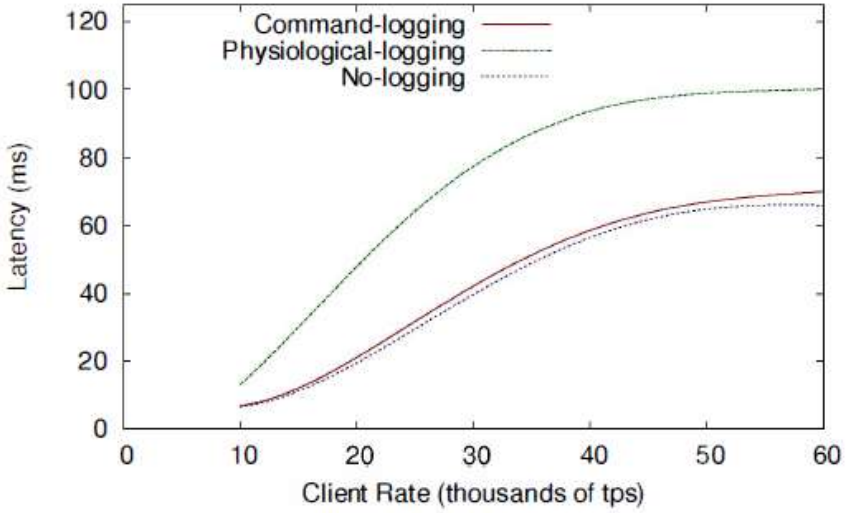
Throughput


Command-logging
Physiological-logging
No-logging
Voter throughput (thousands of tps)
Client rate (thousands of tps)
0
20
40
60
80
100
120
140
160

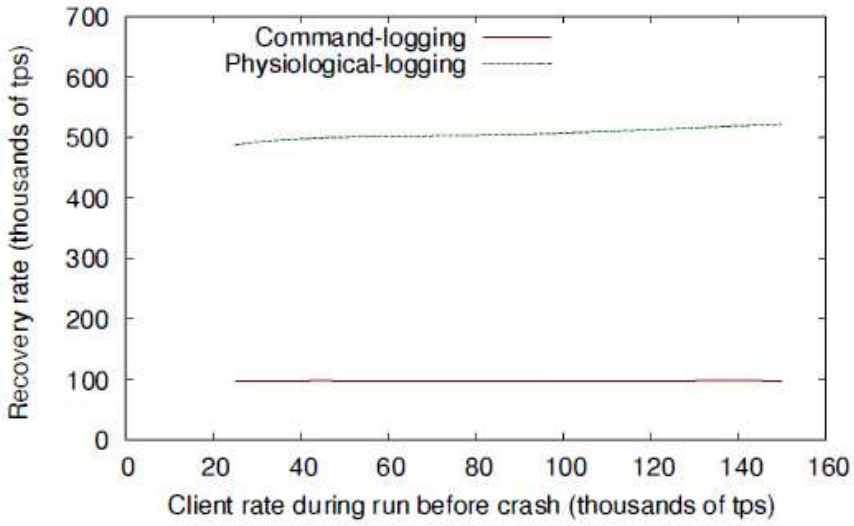

Command-logging
Physiological-logging
No-logging
TPCC throughput (thousands of tpmC)
Client rate (thousands of tps)
0
200
400
600
800
1000
10
20
30
40
50
60

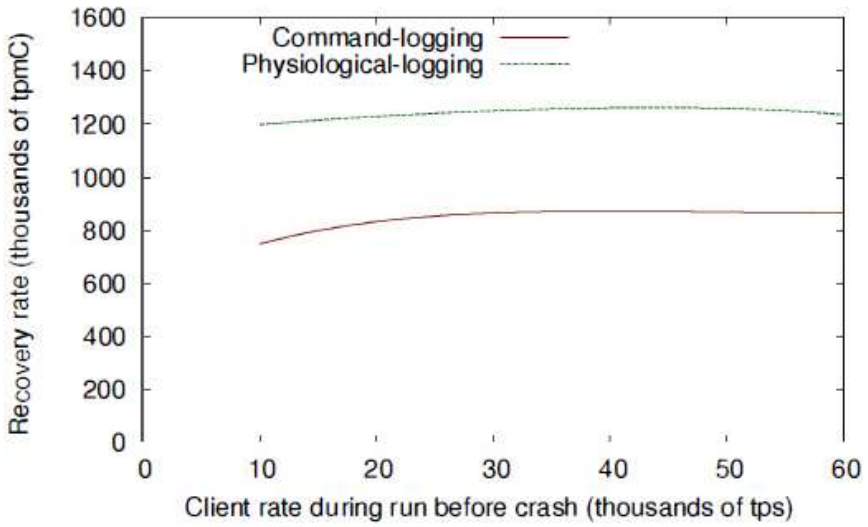
Latency


Command-logging
Physiological-logging
No-logging
Latency (ms)
Client Rate (thousands of tps)
0
10
20
30
40
50
0
20
40
60
80
100
120
140
160

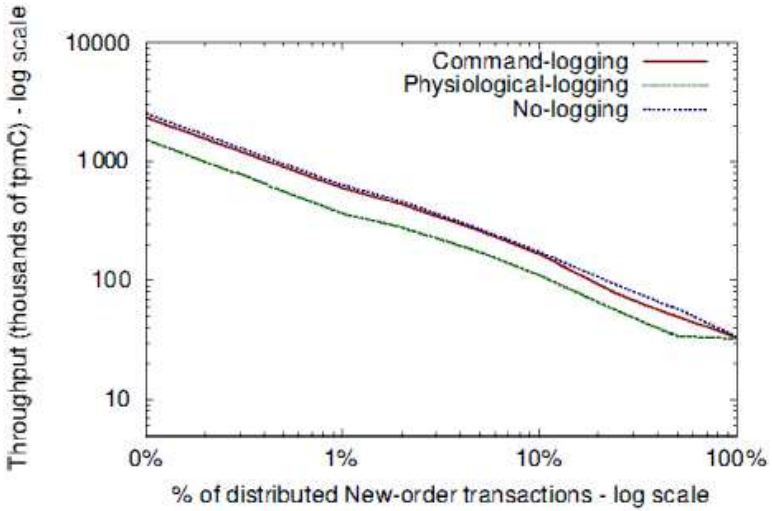

Command-logging
Physiological-logging
No-logging
Latency (ms)
Client Rate (thousands of tps)
0
20
40
60
80
100
120
0
10
20
30
40
50
60

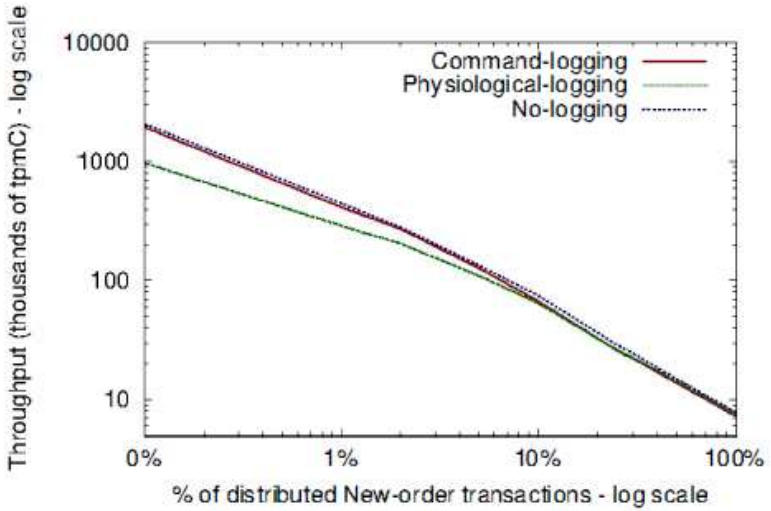
Recovery


Command-logging
Physiological-logging
Recovery rate (thousands of tps)
Client rate during run before crash (thousands of tps)
0
100
200
300
400
500
600
700
0
20
40
60
80
100
120
140
160

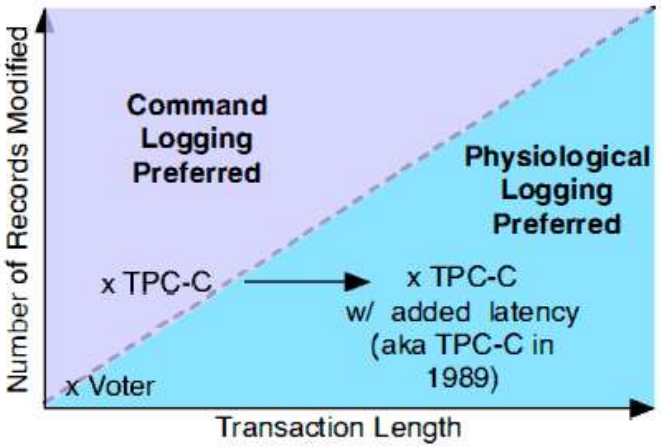

Command-logging
Physiological-logging
Recovery rate (thousands of tpmC)
Client rate during run before crash (thousands of tps)
0
200
400
600
800
1000
1200
1400
1600
0
10
20
30
40
50
60

Throughput for distributed transactions





Command logging vs Physiological logging preference



Conclusion

- Command logging has nearly zero overhead
- 1.5X overall improvement on TPC-C and 1.2X on Voter
- Recovery times ae very high- 1.5X slow on TPC-C and 5X on Voter
- For high fraction of distributed transactions, physiological logging is better

Comments

- How does group commit ensure durability ?
- The recovery time is very high.
- All transactions must be known in advance.
- Queries within a single partition are faster than inter-partition queries so the partitioning and query load has to be computed carefully
- Paper mentions that by analyzing and precompiling the data access logic in the stored procedures, VoltDB can distribute both the data and the processing associated with it to the individual partitions on the cluster, but no details are given