Scalable Atomic Visibility with RAMP Transactions

Peter Bailis, Alan Fekete[2], Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica
UC Berkeley and University of Sydney[2]

Presenter: Suyash Gupta, Purdue University

April 4, 2017

Food For Thought

Can we design an in-expensive strategy that supports multi-partition and multi-operation transactional access wthout employing locking or validation mechanisms?

Lets Refresh !!!

- Transaction
- Atomically Visible Transactional Access
- Read-Write race
- Data consistency

Motivation I – Atomic Visibility

- We need to ensure either all or none of the effects of transaction are visible.
- Example:
  - Say, initially $x = null$ and $y = null$.
  - If transaction $T1$ sets $x = 1$ and $y = 1$, then concurrent transaction $T2$ should not read $x = 1$ and $y = null$.

Motivation II – Locking

- Use Two Phase Locking ?
- Use Optimistic Concurrency Control ?
- Slow !!!
- Unavailable under failure !!!

Motivation III – Foreign Key Constraint

- Database schemas maintain relationships between records in the form of foreign key constraints.
- Databases store bi-directional relationships as two uni-directional relationships.
- Example – a user *like's* a photo on Facebook $\rightarrow$ leads to updates to both the `LIKES` and `LIKED_BY` associations.
- Use of foreign key may lead to inconsistent updates!

Motivation IV – Secondary Indexes

- Data partitioned across servers using `Primary Key`.
- Data access using `Secondary aatribute` slow!
- Use of *Local secondary index* (co-located with primary key) or *Global secondary index* (separate storage of secondary attribute).
- Updation either constly, or inconsistent.

Motivation V – Materialized View Maintenance

- Pre-computed data maintained as view, for faster access.
- LinkedIn's Expresso store's a count of unread mails for each user
- Counters need to be in sync with the messages in mailbox.

RA Isolation – Formal Definition

- Transaction T_j exhibits fractured reads, if another transaction Ti writes versions x_m and y_n, and T_j reads version x_m and version y_k, and $k < n$.
- Read Atomic isolation (RA) prevents:
  - Fractured reads anomalies.
  - Transactions from reading uncommitted, aborted, or intermediate data.
- RA provides transactions with a "snapshot" view of the database that respects transaction boundaries.

RA Implications & Limitations

- RA neither prevents concurrent updates nor provides serial access to the data items.
- Example: RA unsuitable for maintaining bank account balances.
- RA suitable for the "friend" operation.
- RA interpretation easy from programmer's perspectives.

System Model

- Partitioned databases.
- Items in the database spread over multiple servers.
- Single logical copy per item.
- Clients forward operations on each item to it's partition, where they are executed.
- Transaction execution either commits or aborts.
- All data items initialized to null.
- No replication.

Scalability – Synchronization Independence

- One client's transactions cannot block another client's transaction.
- If a partition, responsible for each item in a transaction is reachable, then the transaction will terminate.
- Guarantee of useful progress for each client.
- In the absence of failures, maximum useful concurrency.

Robustness – Partition Independence

- A client does not need to contact partitions that contain no data item accessed by its transactions.
- Effect of partition failure limited!
- Reduced load on servers not involved in a transaction's execution.

RAMP

- Read Atomic Multi-Partition transactions.
- Aimed towards achieving RA Isolation.
- Guarantee synchronization independence and partition independence.
- Do not stall reads or writes – allow reads to *race* writes.
- Detect partial updates autonomously, and repair if needed.

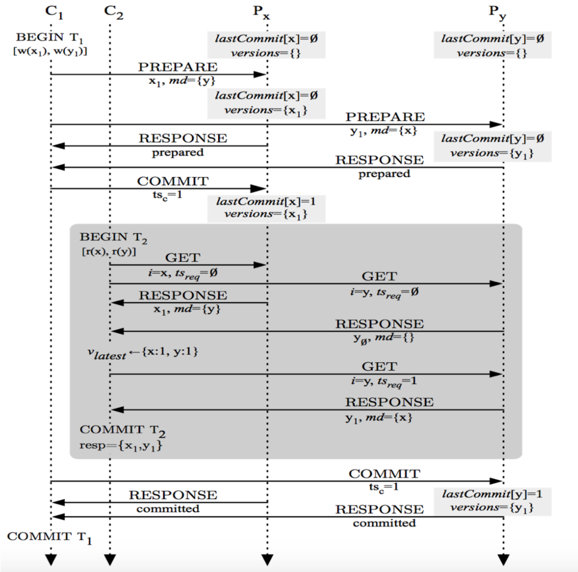
RAMP-Fast

- If race-free, then one Round-Trip Time (RTT) for reads, and two RTTs for writes.
- Meta-data stored as write sets.
- Overhead linear in transaction size.
- RAMP-F Write Transactions – Two phases
  - PREPARE
    - Each timestamped write is placed on its respective partition.
    - Each partition adds the write to its local database.
  - COMMIT
    - Marks versions as committed.
    - Each partition updates an index containing the highest-timestamped committed version of each item.

RAMP-Fast

- RAMP-F Read Transaction
  - Phase I
    - Fetch the last (highest-timestamped) committed version for each item from its respective partition.
    - Each reader calculates whether it is “missing" any versions
    - Generate an item to version (time-stamp) mapping.
  - Phase II
    - If lower timestampped version of an item read, issue a second read to fetch the missing version.
    - Once all missing versions fetched, the client returns.

RAMP-F in Action



RAMP-Small

- Uses constant-size metadata.
- Needs two RTT for reads.
- Read Phase I – Fetch the highest committed timestamp for each item from its respective partition.
- Read Phase II – Retrieve the highest-timestamped version of the item that also appears in the supplied set of timestamps.

RAMP-Small – Example

- T_2's first round read – values fetched are $\{1\}$ and $\{\perp\}$ from partitions P_x and P_y, respectively.
- T_2 sends, the set $\{1, \perp\}$ to both partitions.
- P_x returns x_1 and P_y returns y_1.

RAMP-Hybrid

- RAMP-H – a compromise between Ramp-F and Ramp-S.
- Instead of storing write set, writers store a Bloom Filter representing the transaction write set.
- RAMP-H readers use the RAMP-F style – PHASE I
  - Fetch the last-committed version of each item from its partition.
  - Given the set of versions, compute a list of potentially higher-timestamped writes for each item.
- RAMP-H readers – PHASE II – Fetch any missing versions.

Evaluation Statistics

- RAMP-F, RAMP-H, and often RAMP-S yielded efficient solutions across various workloads while exhibiting overheads within 8%, and less than 48% of peak throughput.
- Algorithms evaulated using YCSB benchmark.
- Several cr1.8xlarge instances also evaluated on Amazon EC2 with a 95% read and 5% write proportion.

Future Thoughts

- BOHM's biggest disadvantage is its need to pre-determine the write-sets of the transaction, prior to its execution.
- Interesting thought can be to design an approach on similar lines for on-line or real-time systems, with obvious tradeoffs.
- Batching transactions entering at same instant.


THANK
YOU!