Coordination Avoidance in Distributed Databases

by Peter Bailis (Standford University)

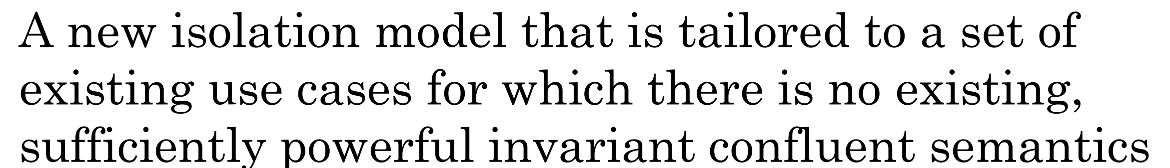
Chapter 5: sections 5.1-5.3, 5.5

Elizaveta S. Prokofyeva

PhD student at the School of Business Informatics

Overview

- Chapter 1- Introduction
- Chapter 2 - Background on coordination and system model
- Chapter 3 - Invariant Confluence property
- Chapter 4 - Invariant Confluence Analysis: Isolation Levels
- Chapter 5 is devoted to Read Atomic Isolation and RAMP Transaction



A new isolation model that is tailored to a set of existing use cases for which there is no existing, sufficiently powerful invariant confluent semantics

Chapter Goal

- A new, non-serializable isolation model - **READ ATOMIC (RA) ISOLATION**
- All or none of each transaction's updates are visible to others and that each transaction reads from an atomic snapshot of database state

READ

ATOMIC

MULTI-

PARTITION

TRANSACTIONS

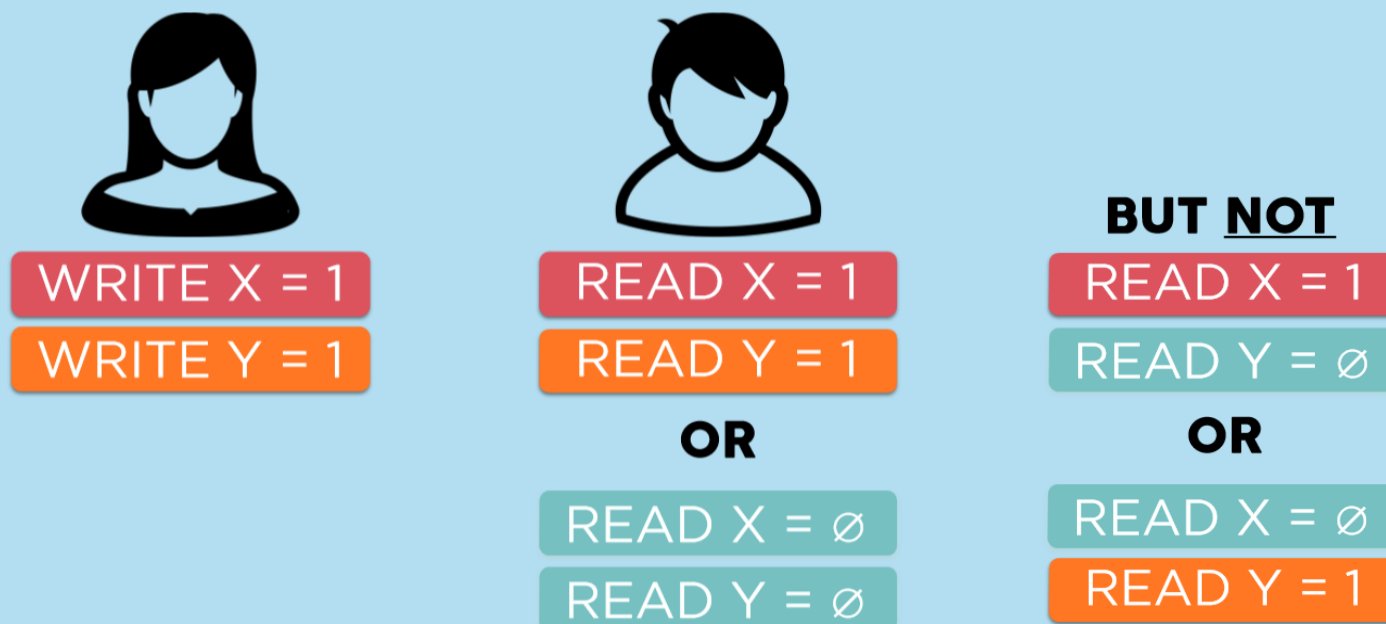
RAMP transactions guarantee scalability and outperform existing atomic algorithms because they satisfy two key scalability constraints:

1. **coordination-free execution:**
one client's transactions cannot cause another client's transactions to stall or fail

2. **partition independence:** clients only contact partitions that their transactions directly reference (i.e., there is no central master, coordinator, or scheduler)

Problem statement

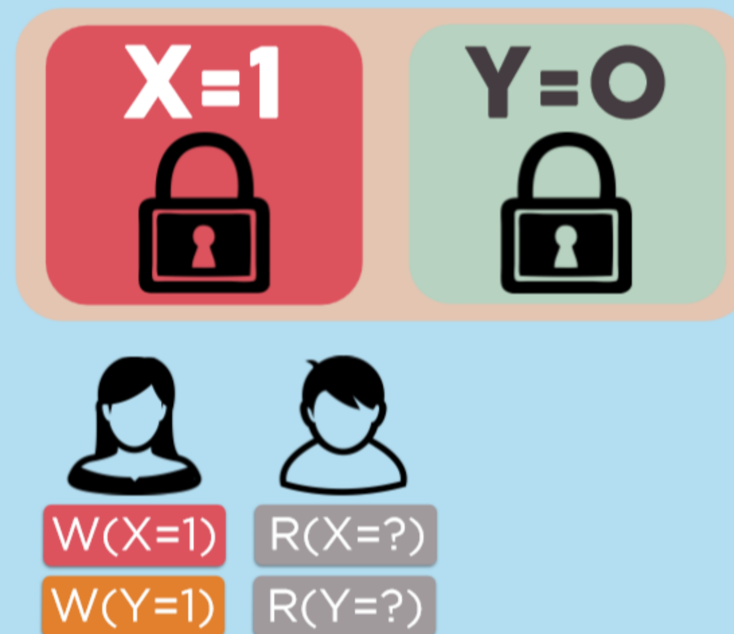
- We consider the problem of making transactional updates atomically visible to readers
- Either all or none of each transaction's updates should be visible to other transactions



Source: Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica UC Berkeley and University of Sydney Overview deck with Cassandra discussion@pbailis

Problem statement

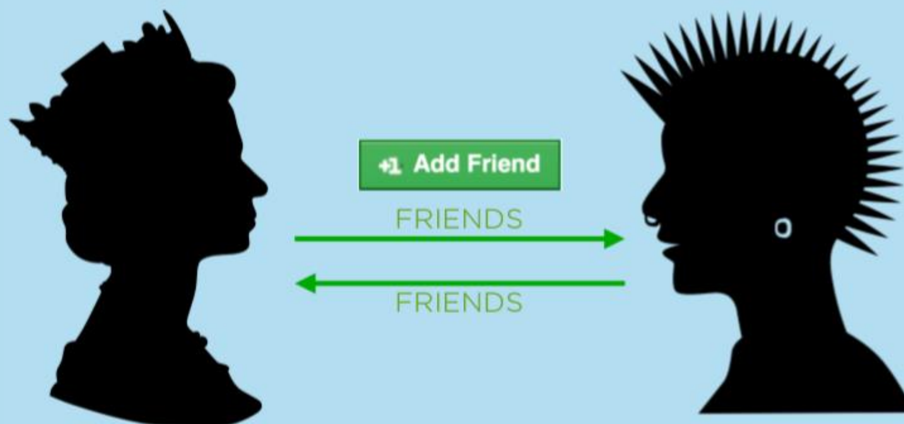
- The classic strategy for providing atomic visibility is **to ensure mutual exclusion** between readers and writers.
- If a transaction like T1 above wants to update data items **x** and **y**, it can acquire exclusive locks for each of **x** and **y**, update both items, then release the locks
- No other transactions will observe partial updates to **x** and **y**, ensuring atomic visibility
- **DRAWBACK:** while one transaction holds exclusive locks on **x** and **y**, no other transactions can access x and y for either reads or writes



Source: Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica UC Berkeley and University of Sydney Overview deck with Cassandra discussion@pbailis

Read Atomic Isolation

- If two users become "friends" (a bi-directional relationship), other users should never see that Sam is a friend of Elizabeth but Elizabeth is not a friend of Sam: either both relationships should be visible, or neither should be
- A transaction under **Read Atomic isolation** would correctly enforce this behavior



Source: Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica UC Berkeley and University of Sydney Overview deck with Cassandra discussion@pbailis

Read Atomic Isolation

- **GENERAL USE CASES**

1. **Foreign key constraints:** Under RA isolation, when inserting new entities, applications can bundle relevant entities from each side of a foreign key constraint into a transaction. When deleting associations, users can avoid dangling pointers by creating a "tombstone" at the opposite end of the association

2. **Secondary indexing:** Under RA isolation, the secondary index entry for a given attribute can be updated atomically with base data. Insertions of new primary data require additions to the corresponding index entry, deletions require removals, and updates require a "tombstone" deletion from one entry and an insertion into another.

3. **Materialized view maintenance:** With RAMP transactions, base data and views can be updated atomically. The maintenance of a view depends on its specification, but RAMP transactions provide appropriate concurrency control primitives for ensuring that changes are delivered to the materialized view partition.

RA Semantics and System Model

- ordered sequences of reads and writes to arbitrary sets of items, or **transactions**;
- each write creates a **version** of an item and we identify versions of items by a **timestamp** taken from a totally ordered set (e.g., natural numbers) that is unique across all versions of each item;
- **timestamps** induce a total order on versions of each item, and we denote version i of item x as x_i
- items have an initial version ⊥ that is located at the start of each order of versions for each item and is produced by an initial transaction T⊥
- each transaction ends in a commit or an abort operation; we call a transaction that commits a committed transaction and a transaction that aborts a aborted transaction

RA Semantics and System Model

- **Fractured Reads**

A transaction T_j exhibits the fractured reads phenomenon if transaction T_i writes versions x_a and y_b (in any order, where x and y may or may not be distinct items), T_j reads version x_a and version y_c, and c < b

- **Read Atomic**

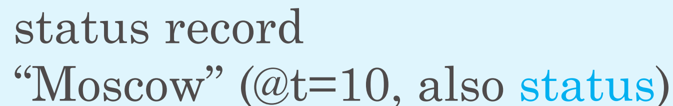
A system provides Read Atomic isolation (RA) if it prevents fractured reads phenomena and also prevents transactions from reading uncommitted, aborted, or intermediate versions

- under RA isolation, readers only observe the final output of a given transaction that has been accepted by the database

RAMP Transactions Idea

- RA is **invariant confluent**: if two read/write histories each independently do not have fractured reads, composing them will not change the values returned by any read operations
- Desired property: see all updates, or see none
- **w(status="talking"); w(loc="Moscow"); commit;**
- **RAMP:** multi- versioning with intention metadata

status record
"talking" (@t=10, also loc) OK

status record


!!! RAMP writers attach metadata to each write and use limited multi-versioning to prevent readers from stalling

RA Compared to Other Isolation Models

- RA is stronger than ***Read Committed*** as ***Read Committed*** does not prevent fractured reads. History 5.1 does not respect RA isolation. After T_1 commits, both T_2 and T_3could both commit but, to prevent fractured reads, T_4 and T_5 must abort

- ***Lost Updates*** phenomena informally occur when two transactions simultaneously attempt to make conditional modifications to the same data item(s). History 5.2 exhibits the Lost Updates phenomenon but is valid under RA. That is, T_1 and T_2 can both commit under RA isolation

$$
\begin{array}{ll}
T_1 & w(x_1); w(y_1) \\
T_2 & r(x_\perp); r(y_\perp) \\
T_3 & r(x_1); r(y_1) \\
T_4 & r(x_\perp); r(y_1) \\
T_5 & r(x_1); r(y_\perp)
\end{array}
\quad (5.1)
$$

$$
\begin{array}{ll}
T_1 & r(x_\perp); w(x_1) \\
T_2 & r(x_\perp); w(x_2)
\end{array}
\quad (5.2)
$$

RA Compared to Other Isolation Models

- History 5.3 exhibits the ***WriteSkew*** phenomenon, but is valid under RA. That is, T_1 and T_2 can both commit under RA isolation

$$\begin{array}{ll} T_1 & r(y_\perp); w(x_1) \\ T_2 & r(x_\perp); w(y_2) \end{array} \qquad (5.3)$$

Missing Transaction Updates

A transaction T_j misses the effects of a transaction T_i if T_i writes x_i and commits and another transaction T_j reads another version x_k such that k < I; i.e., T_j reads a version of x that is older than the version that was committed by T_i

No-Depend-Misses

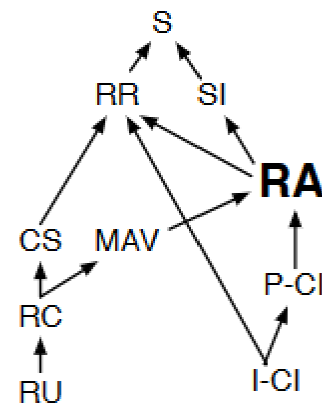
If transaction T_j depends on transaction T_i, T_j does not miss the effects of T_i

RA Compared to Other Isolation Models

- T1, T2, and T3 can all commit under RA isolation. Thus, fractured reads prevention is similar to No-Depend-Misses but only applies to immediate read dependencies (rather than all transitive dependencies).

$$\begin{array}{ll} T_1 & w(x_1); w(y_1) \\ T_2 & r(y_1); w(z_2) \\ T_3 & r(x_\perp); r(z_2) \end{array} \quad (5.4)$$

RA is stronger than Read Committed, Monotonic Atomic View, and Cut Isolation, weaker than Snapshot Isolation, Repeatable Read, and Serializability, and incomparable to Cursor Stability.



RA and Serializability

- for read-only and write-only transactions, if each reading transaction only reads a subset of the items that another write-only transaction wrote, then RA isolation is equivalent to **one-copy serializable isolation**
- by restricting the set of operations accessible to a user (e.g., RSIW read-only and write-only transactions), one can often achieve more scalable implementations without necessarily violating existing abstractions (e.g., one-copy serializable isolation)

A history is **one-copy serializable** if it is view equivalent to a serial execution of the transactions over a single logical copy of the database

System Model and Scalability

- Coordination-free execution ensures that one client's transactions cannot cause another client's to block and that, if a client can contact the partition responsible for each item in its transaction, the transaction will eventually commit
- "strong" isolation models like serializability and Snapshot Isolation require coordination and thus limit scalability. Locking is an example of a non-coordination-free implementation mechanism
- Partition independence ensures that, in order to execute a transaction, a client only contacts partitions for data items that its transaction directly accesses. Thus, a partition failure only affects transactions that access items contained on the partition.

RAMP Algorithms

1. RAMP-Small transactions require constant space (a timestamp per write) and two round trip time delays (RTTs) for reads and writes

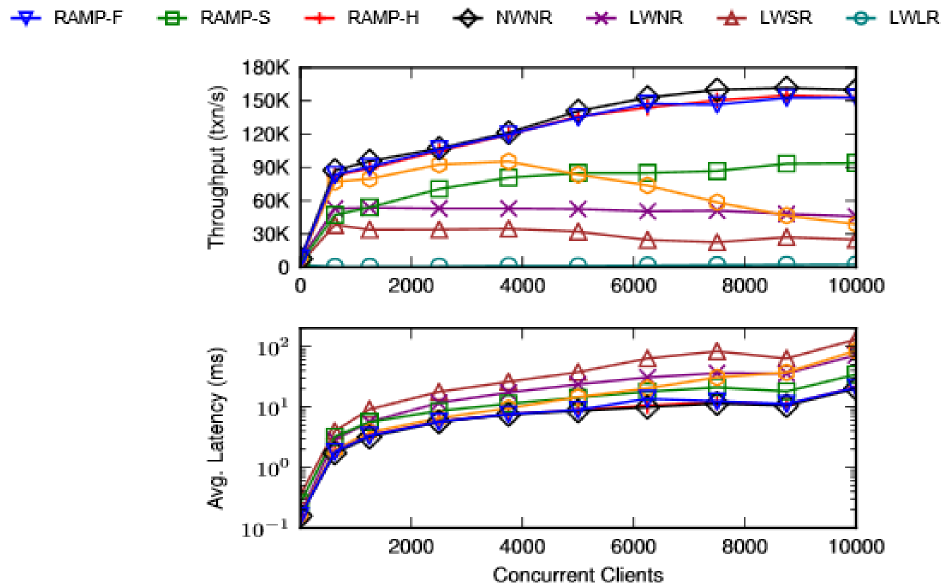
2. RAMP-Fast transactions require metadata size that is linear in the number of writes in the transaction but only require one RTT for reads in the common case and two in the worst case

3. RAMP-Hybrid transactions employ Bloom filters to provide an intermediate solution.

- Traditional techniques like locking couple atomic visibility and mutual exclusion; RAMP transactions provide the benefits of the former without incurring the scalability, availability, or latency penalties of the latter.

Experimental Evaluation

- several concurrency control algorithms in a partitioned, multi-versioned, main-memory database prototype
- our prototype is in Java and employs a custom RPC system for serialization
- evaluate each algorithm using the YCSB benchmark (Yahoo! Cloud Serving Benchmark- an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs)



Throughput and latency under varying client load

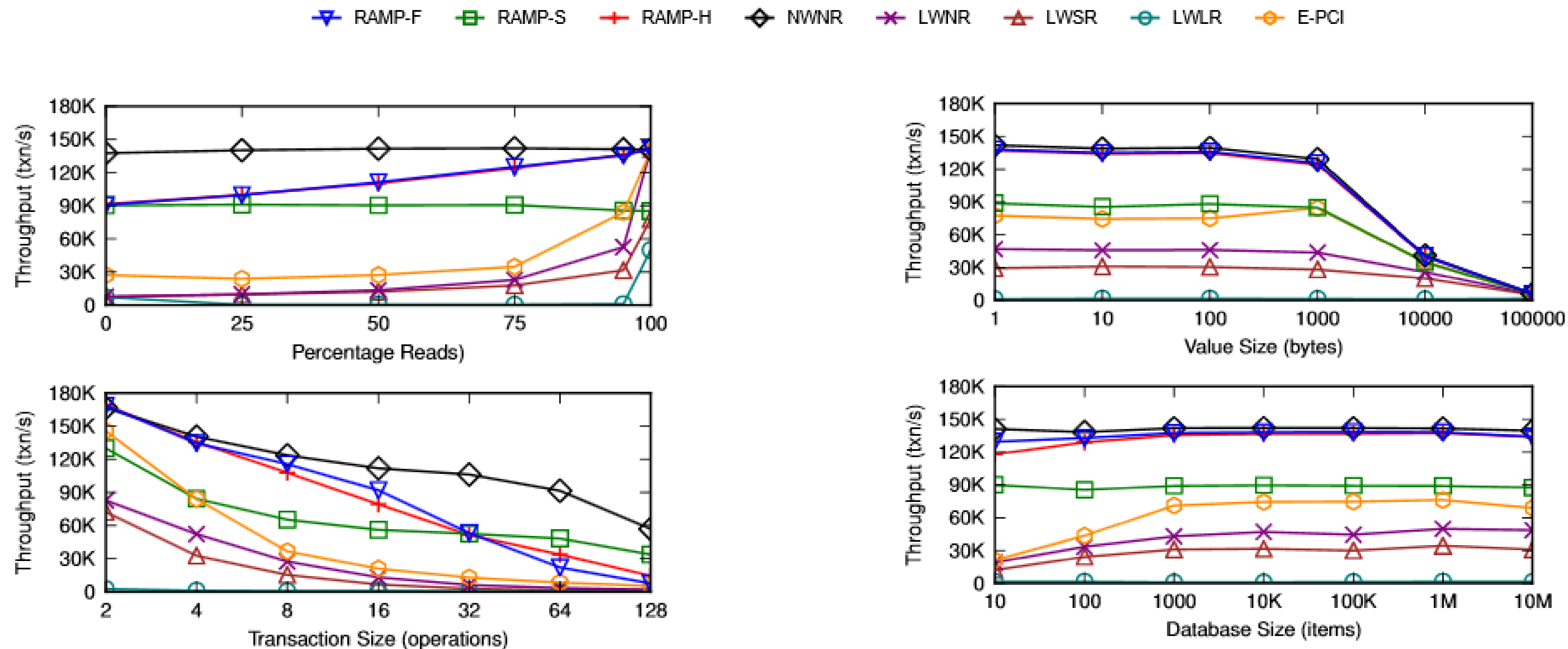
1. performance compared to baseline (BEST CASE!):
as a baseline, we do not employ any concurrency control (denoted NWNR, for no write and no read locks); reads and writes take one RTT and are executed in parallel

2. performance compared to existing techniques:

- lock-based strategies

LWLR- long write locks and long read locks, providing Repeatable Read isolation
LWSR- long write locks with short read locks, providing Read Committed isolation
LWNR-long write locks with no read locks, providing Read Uncommitted isolation

Experimental Evaluation



Algorithm performance across varying workload conditions.
RAMP-F and RAMP-H exhibit similar performance to NWNR baseline, while RAMP-S's 2 RTT reads incur a greater performance penalty across almost all configurations. RAMP transactions consistently out-perform RA isolated alternatives

Experimental Results: Scalability

- linear scalability of RAMP transactions to 100 servers
- with 100 servers, RAMP-F achieves slightly under 7.1 million operations per second, or 1.79 million transactions per second on a set of 100 servers
- RAMP-F throughput was always within 10% of NWNR

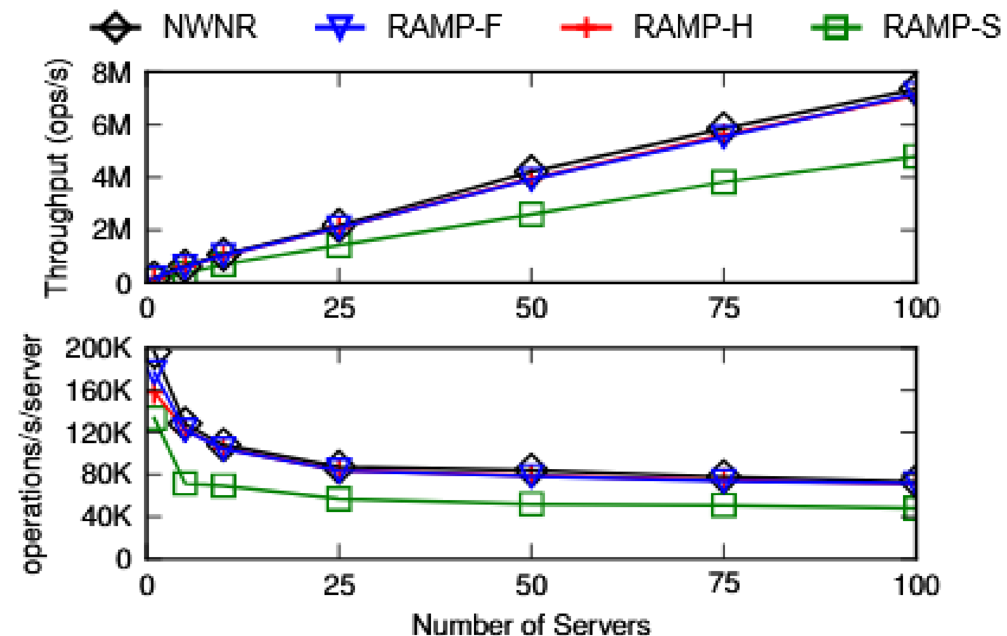


Figure 5.6: RAMP transactions scale linearly to over 7 million operations/s with comparable performance to NWNR baseline.

Conclusion

- We identified a new isolation level—**Read Atomic isolation—**that provides atomic visibility and matches the requirements of a large class of real-world applications
- We subsequently achieved RA isolation via scalable **RAMP transactions**
- **RAMP transactions** provide correct semantics for applications requiring secondary indexing, foreign key constraints, and materialized view maintenance while maintaining scalability and performance
- The choice of coordination-free and partition independent algorithms allowed us to achieve **near-baseline performance** across a variety of workload configurations and scale linearly to 100 servers
- While RAMP transactions are not appropriate for all applications, the many applications for which they are appropriate will benefit significantly

Thank you for attention!