Squall: Fine-Grained Live Reconfiguration for Partitioned Main Memory Databases

AARON J. ELMORE, VAIBHAV ARORA, REBECCA TAFT, ANDY PAVLO, DIVY AGRAWAL, AMR EL ABBADI

Higher OLTP Throughput

Demand for High-throughput transactional systems (OLTP) especially due to web-based services

- Cost per GB for RAM is dropping.
- Network memory is faster than local disk.

Let's use Main-Memory

Scaling-out via Partitioning

Growth in scale of the data

Data Partitioning enables managing scale via Scaling-Out.

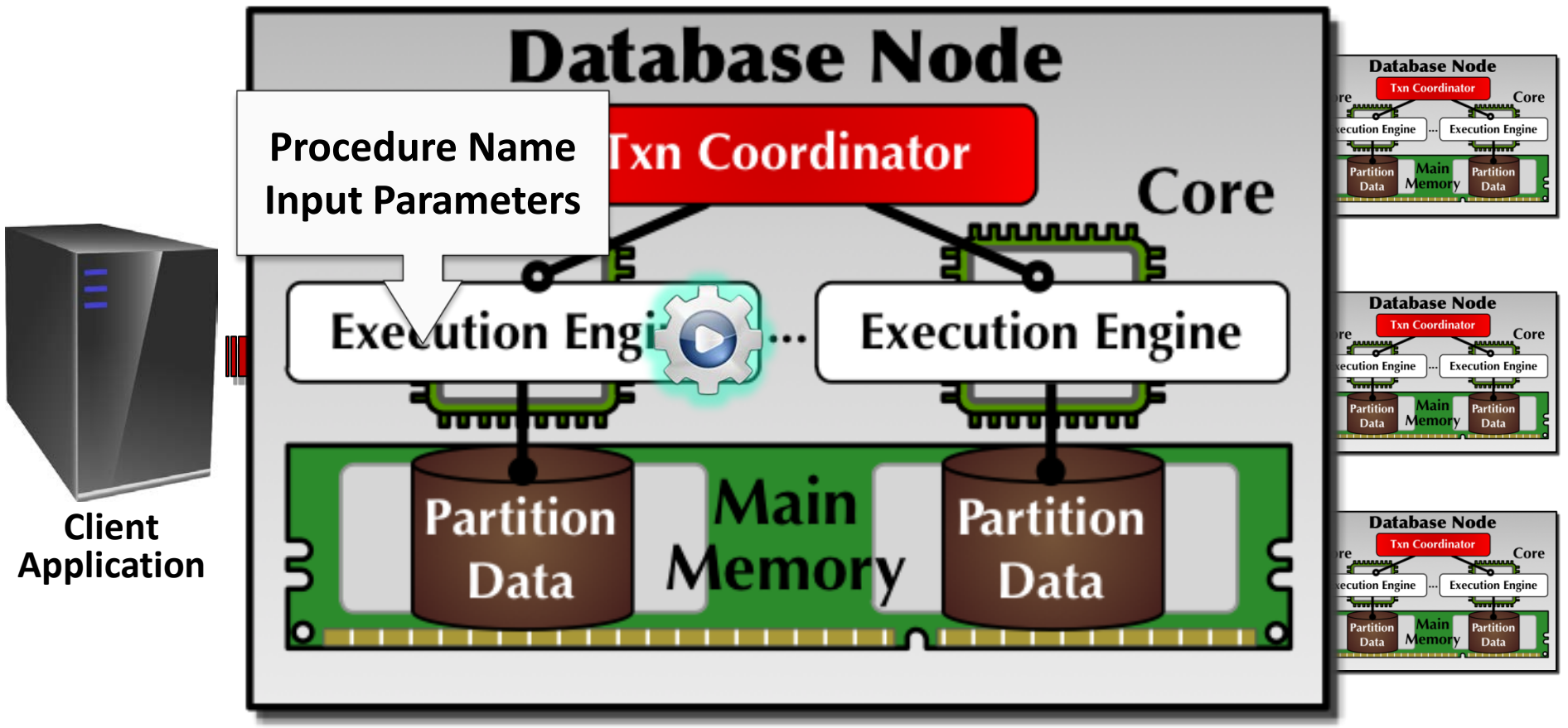
Approaches for main-memory DBMS*

Highly concurrent, latch-free data structures – Hekaton, Silo

Partitioned data with single-threaded executors – Hstore, VoltDB

*Excuse the generalization

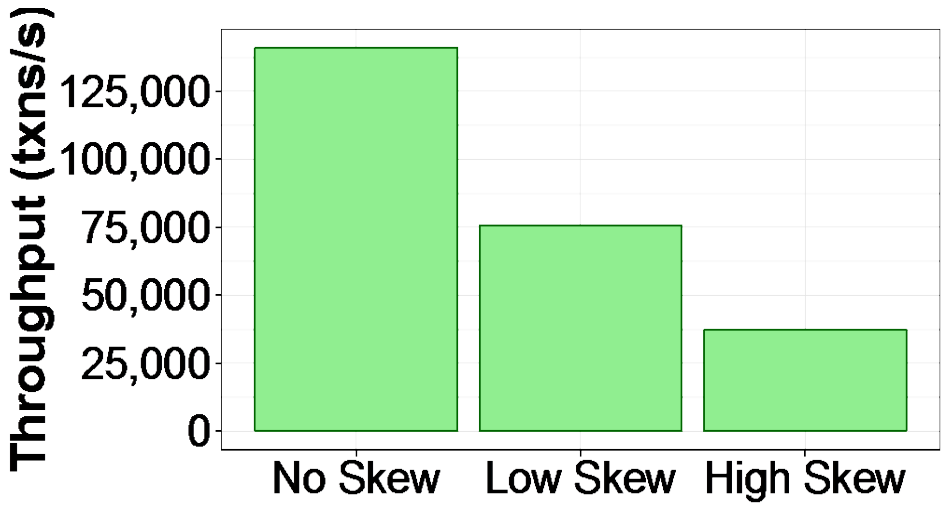
H-Store


Database Node
Txn Coordinator
Core
Procedure Name
Input Parameters
Execution Engine
Execution Engine
Partition Data
Main Memory
Partition Data
Client Application
Database Node
Txn Coordinator
Core
Execution Engine
Execution Engine
Partition Data
Main Memory
Partition Data
Database Node
Txn Coordinator
Core
Execution Engine
Execution Engine
Partition Data
Main Memory
Partition Data
Database Node
Txn Coordinator
Core
Execution Engine
Execution Engine
Partition Data
Main Memory
Partition Data

The Problem: Workload Skew

High skew increases latency by 10X and decreases throughput by 4X

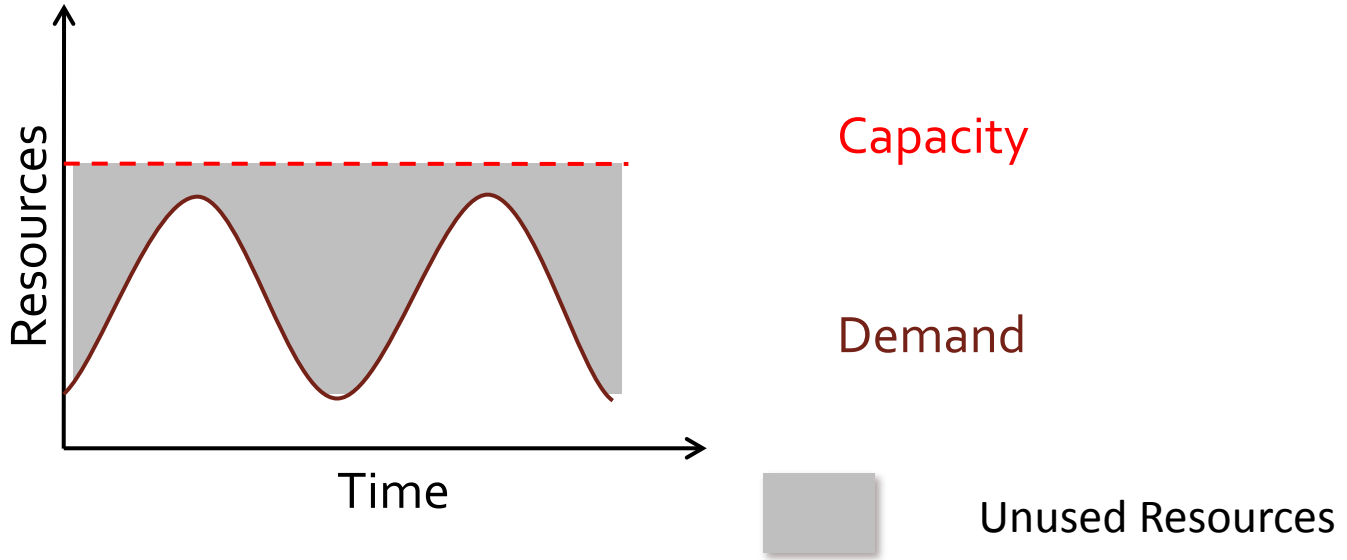
Partitioned shared-nothing systems are especially susceptible



The Problem: Workload Skew

Possible solutions:

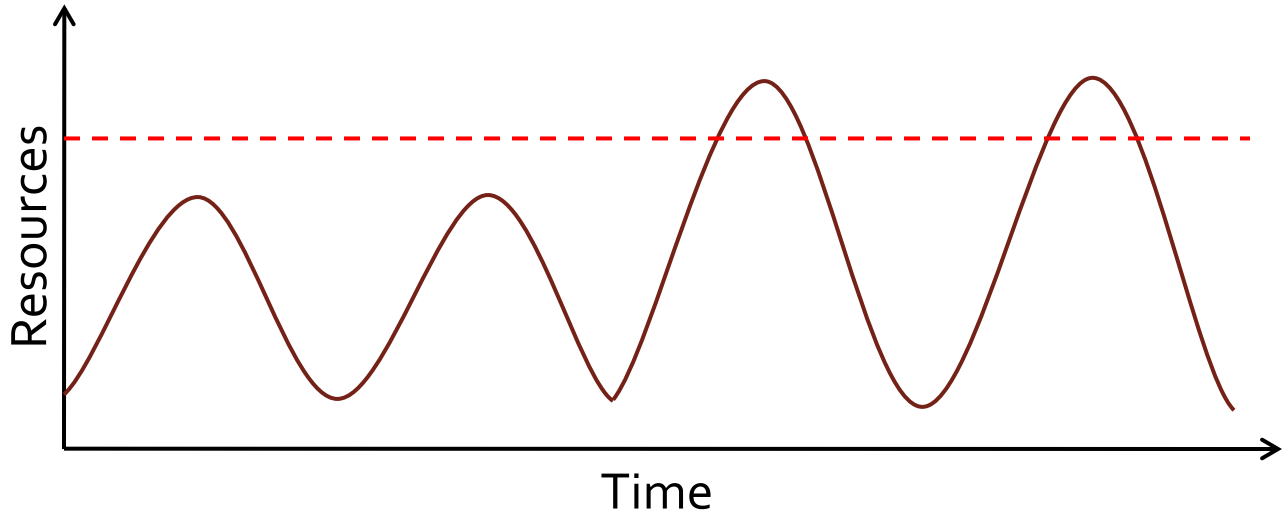
Provision resources for peak load (Very expensive and brittle!)



The Problem: Workload Skew

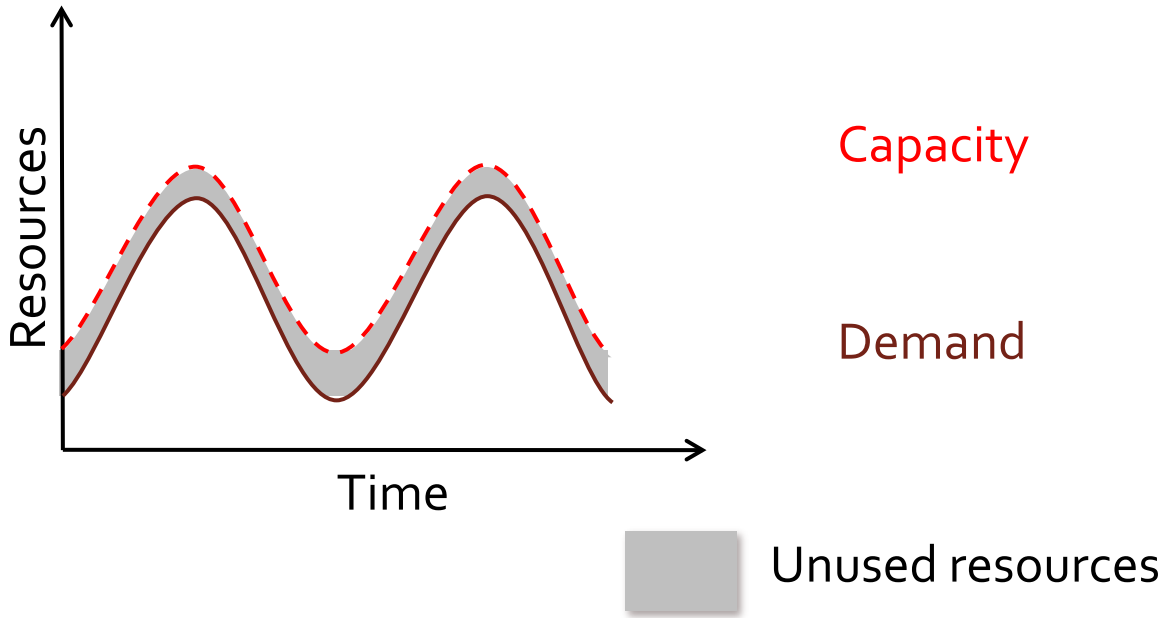
Possible solutions:

Limit load on system (Poor performance!)



Need Elasticity

The Promise of Elasticity



Slide Credits: Berkeley RAD Lab

What we need…

Enable system to elastically scale in or out to dynamically adapt to changes in load

Reconfiguration

Change the partition plan

Add nodes

Remove nodes

Problem Statement

Need to migrate tuples between partitions to reflect the updated **partition plan**.

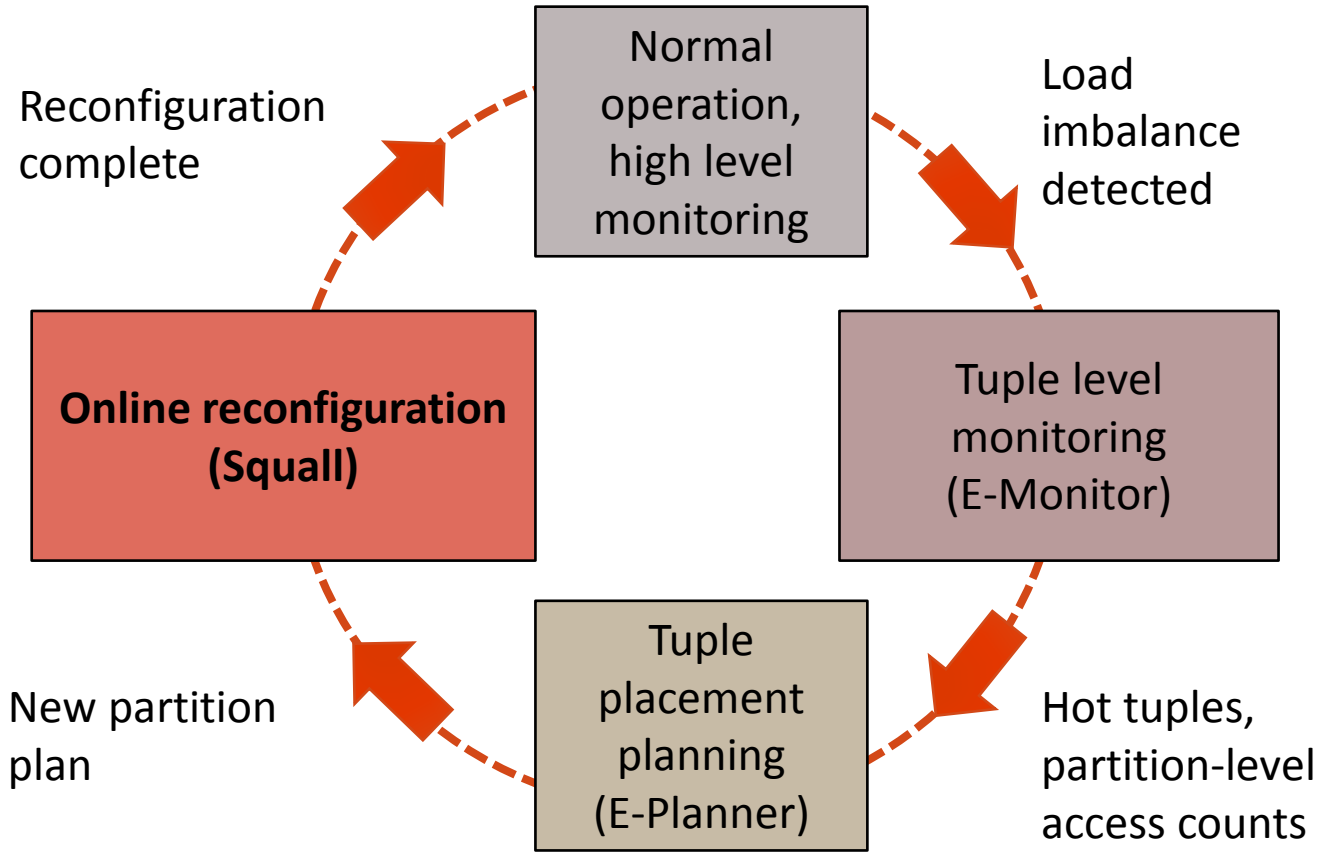
Partition	Warehouse
Partition 1	[0,2)
Partition 2	[2,4)
Partition 3	[4,6)

Partition	Warehouse
Partition 1	[0,1)
Partition 2	[2,3)
Partition 3	[1, 2),[3,6)

Would like to do this without bringing the system offline:

- **Live Reconfiguration**

E-Store


Normal operation, high level monitoring
Load imbalance detected
Tuple level monitoring (E-Monitor)
Hot tuples, partition-level access counts
Tuple placement planning (E-Planner)
New partition plan
Online reconfiguration (Squall)
Reconfiguration complete

Live Migrations Solutions are Not Suitable

Predicated on disk based solutions with traditional concurrency and recovery.

Zephyr: Relies on concurrency (2PL) and disk pages.

ProRea: Relies on concurrency (SI and OCC) and disk pages.

Albatross: Relies on replication and shared disk storage. Also introduces strain on source.

Not Your Parents' Migration

Single threaded execution model

- Either doing work or migration

More than a **single** source and destination (and the destination is not cold)

- Want lightweight coordination

Presence of distributed transactions
and replication

Squall

Given plan from E-Planner, Squall physically moves the data while the system is live

Pull based mechanism – Destination pulls from source

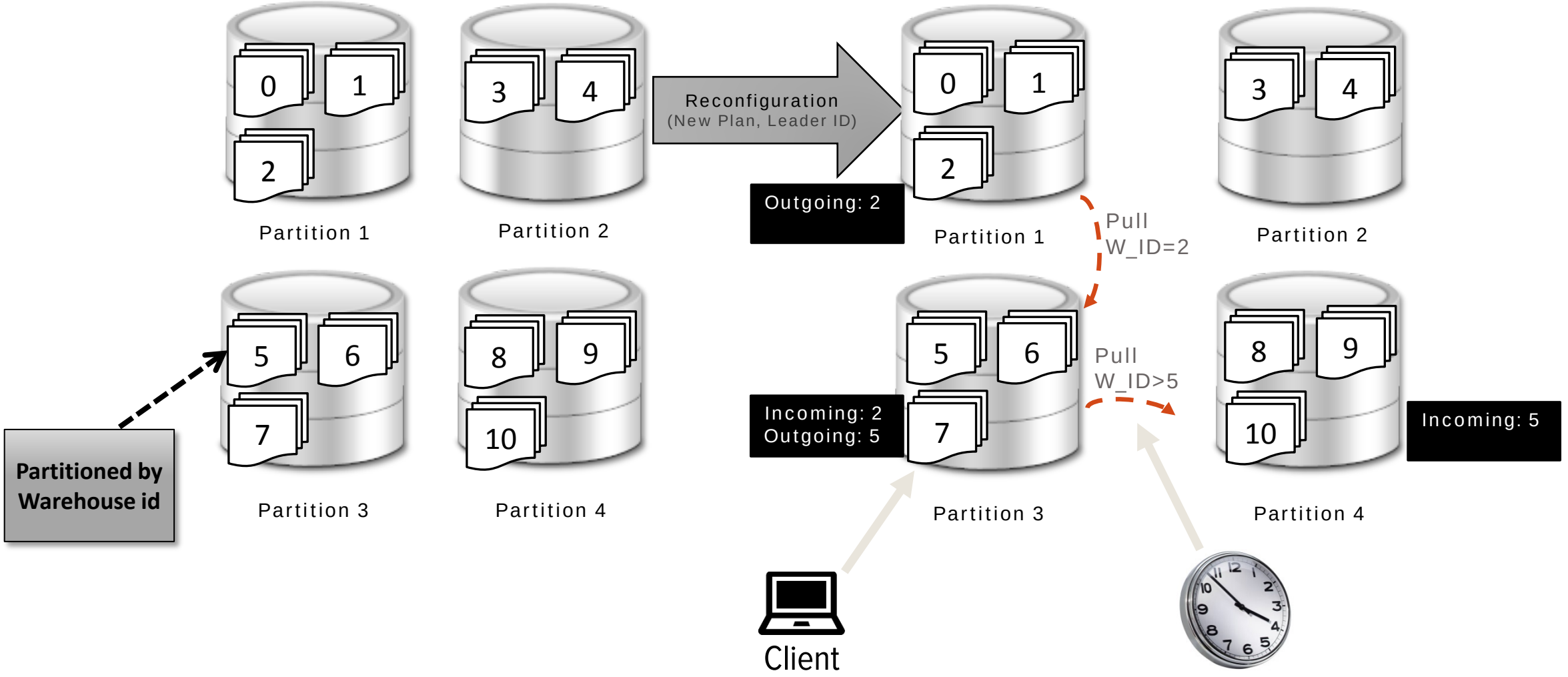
Conforms to H-Store single-threaded execution model

- **While data is moving, transactions are blocked – but only on partitions moving the data**

To avoid performance degradation, Squall moves **small chunks of data at a time**, interleaved with regular transaction execution

Squall Steps

1. Initialization and Identify migrating data
2. Live **reactive pulls** for required data
3. Periodic **lazy/async pulls** for large chunks



Chunk Data for Asynchronous Pulls

Why Chunk?

Unknown amount of data when not partitioned by clustered index.

Customers by W_ID in TPC-C

Time spent extracting, is time not spent on TXNS.

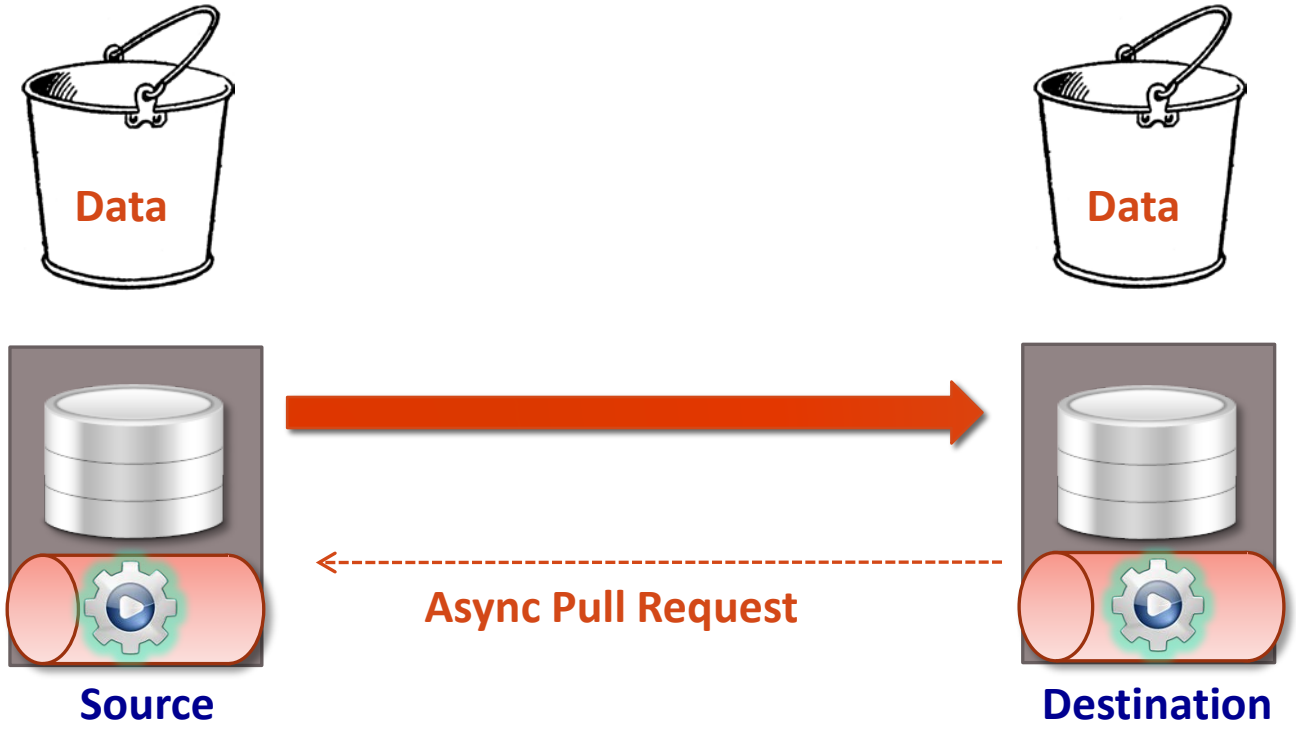
Async Pulls

Periodically pull chunks of cold data

These pulls are answered lazily – Start at lower priority than transactions. Priority increases with time.

Execution is interwoven with extracting and sending data (dirty the range!)

Chunking Async Pulls



Keys to Performance

Properly **size** reconfiguration granules and **space** them apart.

Split large reconfigurations to limit demands on a single partition.

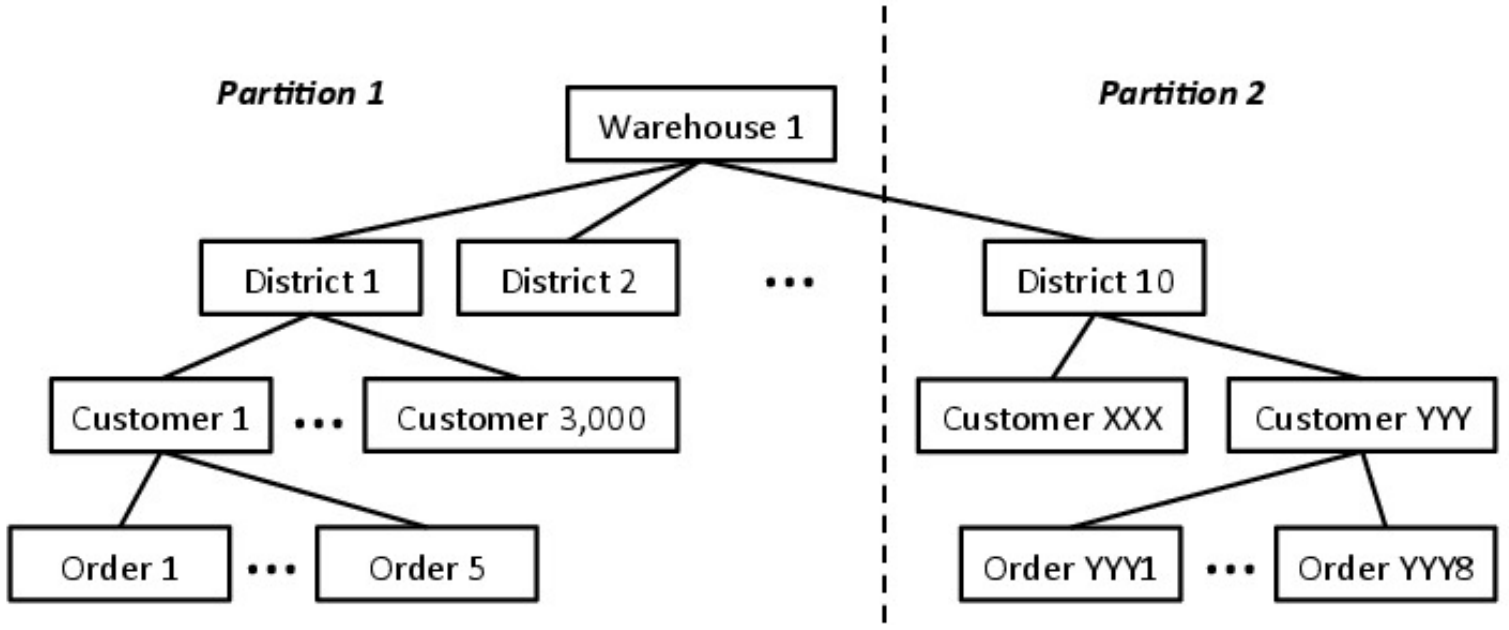
Redirect or pull only if needed.

Tune what gets pulled.

Sometimes pull a little extra.

Optimization: Splitting Reconfigurations

1. Split by pairs of source and destination - Avoids contention to a single partition
   - Example: partition 1 is migrating W_ID 2,3 to partitions 3 and 7, execute as two reconfigurations.
2. Split large objects and migrate one piece at a time



Evaluation

Workloads

YCSB

TPC-C

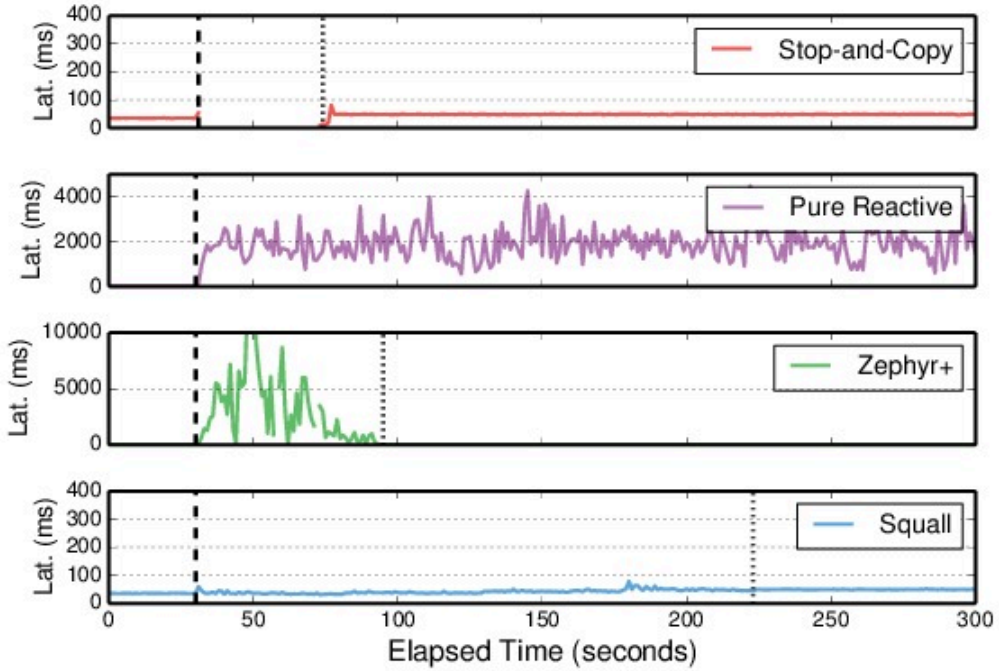
Baselines

Stop & Copy

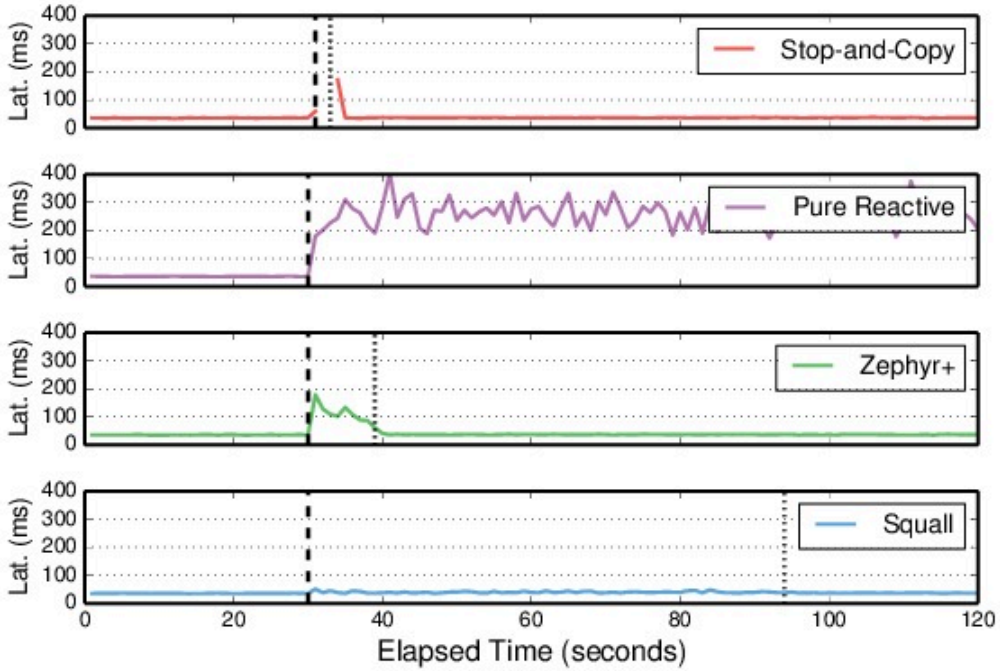
Purely Reactive – Only Demand based pulling

Zephyr+ - Purely Reactive + Asynchronous Chunking with Pull Prefetching (Semantically equivalent to Zephyr)

YCSB Latency

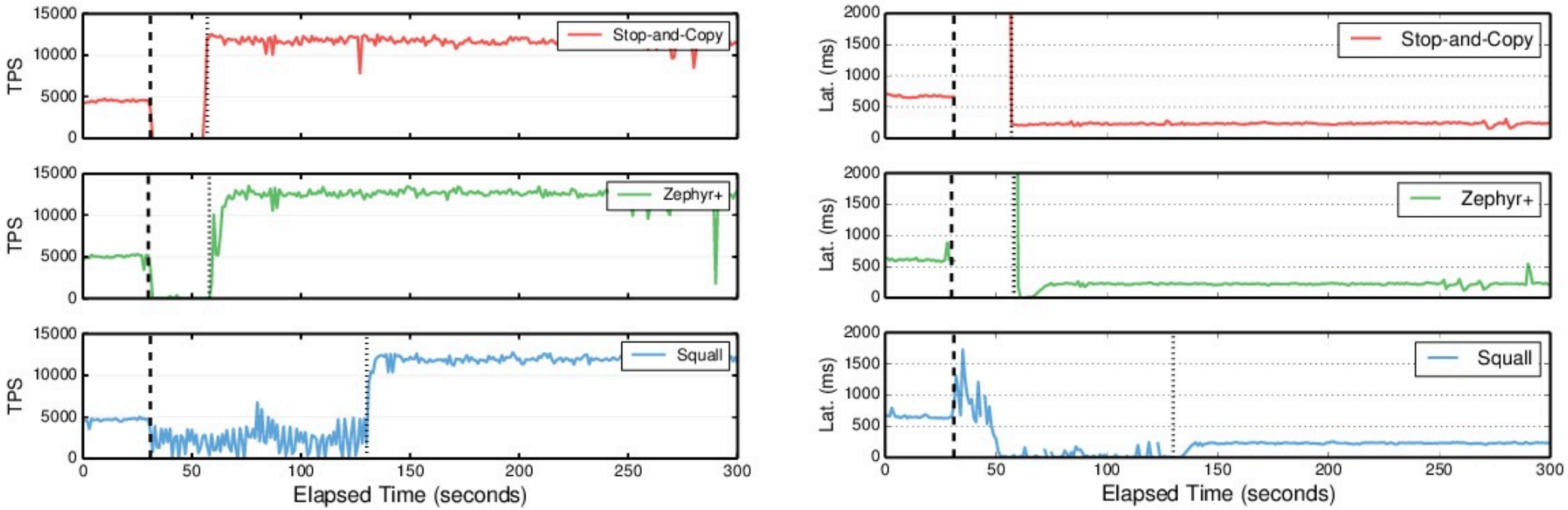


YCSB cluster consolidation 4 to 3 nodes



YCSB data shuffle 10% pairwise

Results Highlight

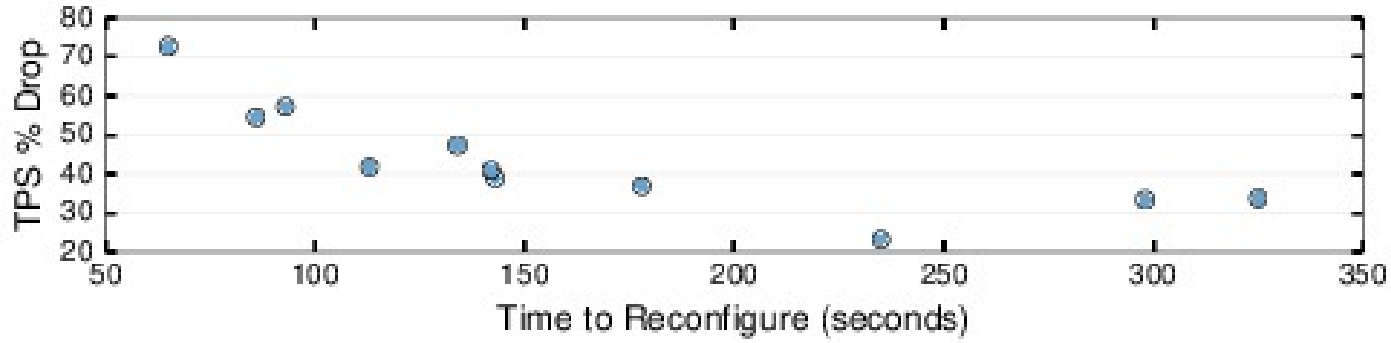


TPC-C load balancing hotspot warehouses

All about trade-offs

Trading off time to complete migration and performance degradation.

Future work to consider automating this trade-off based on service level objectives.



I Fell Asleep… What Happened?

Partitioned Single Threaded Main Memory Environment -> Susceptible to Hotspots.

Elastic data Management is a solution -> Squall provides a mechanism for executing a fine grained live reconfiguration

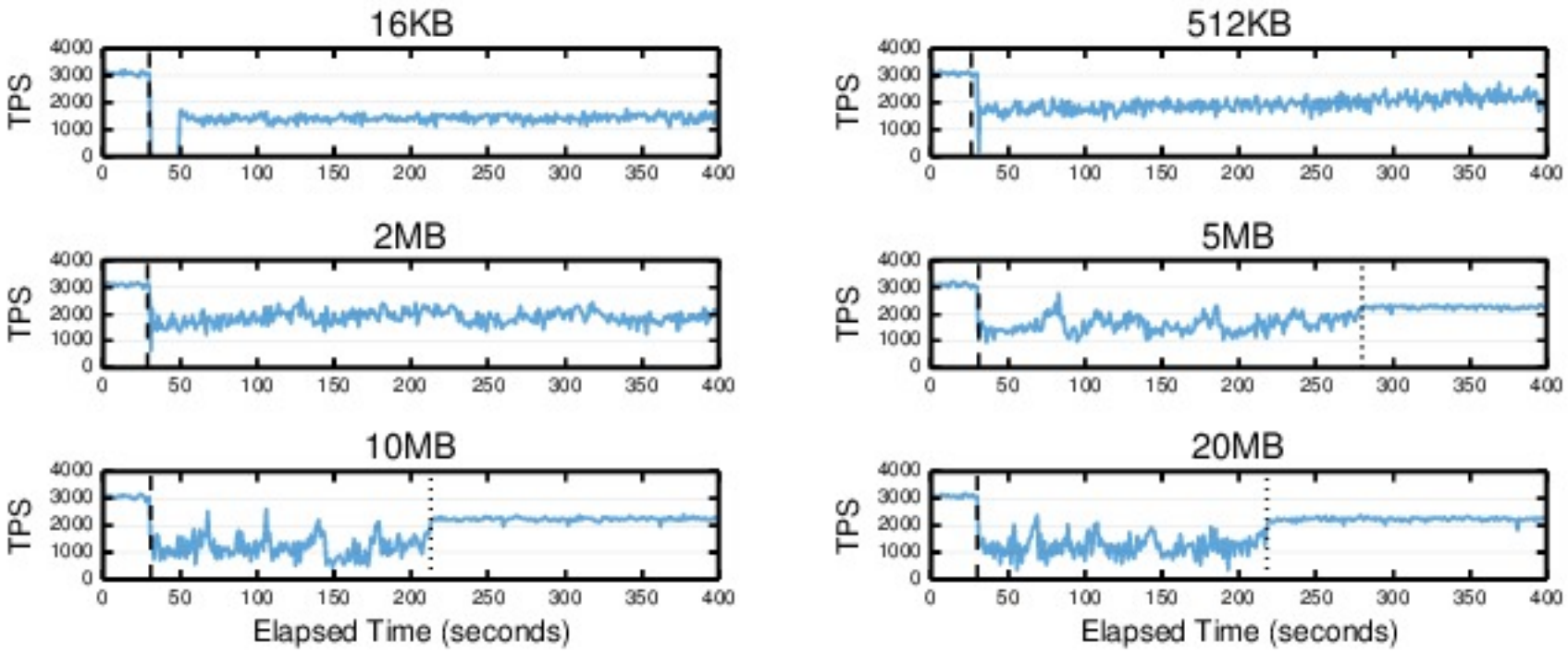
Questions?

Tuning Optimizations

Sizing Chunks

Static analysis to set chunk sizes, future work to dynamically set sizing and scheduling.

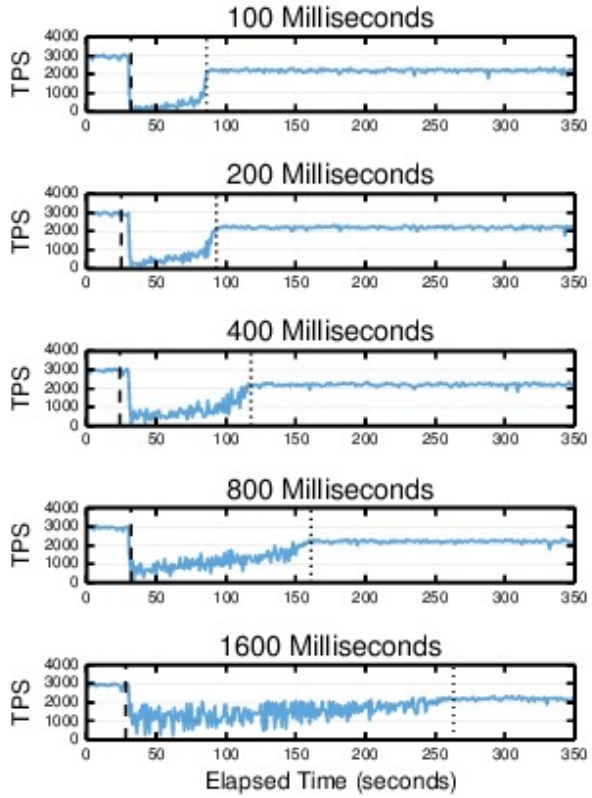
Impact of chunk sizes on a 10% reconfiguration during a YCSB workload.



Spacing Async Pulls

Delay at destination between new async pull requests.

Impact on chunk sizes on a 10% reconfiguration during a YCSB workload with 8mb chunk size.



Effect of Splitting into Sub-Plans

Set a cap on sub-plan splits, and split on pairs and ability to decompose migrating objects