Building an Elastic Main-Memory Database: E-Store

AARON J. ELMORE

AELMORE@CS.UCHICAGO.EDU

Collaboration Between Many

Rebecca Taft, Vaibhav Arora, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andy Pavlo, Amr El Abbadi, Divy Agrawal, Michael Stonebraker

E-Store @ VLDB 2015, Squall @ SIGMOD 2015, E-Store++ @ ????

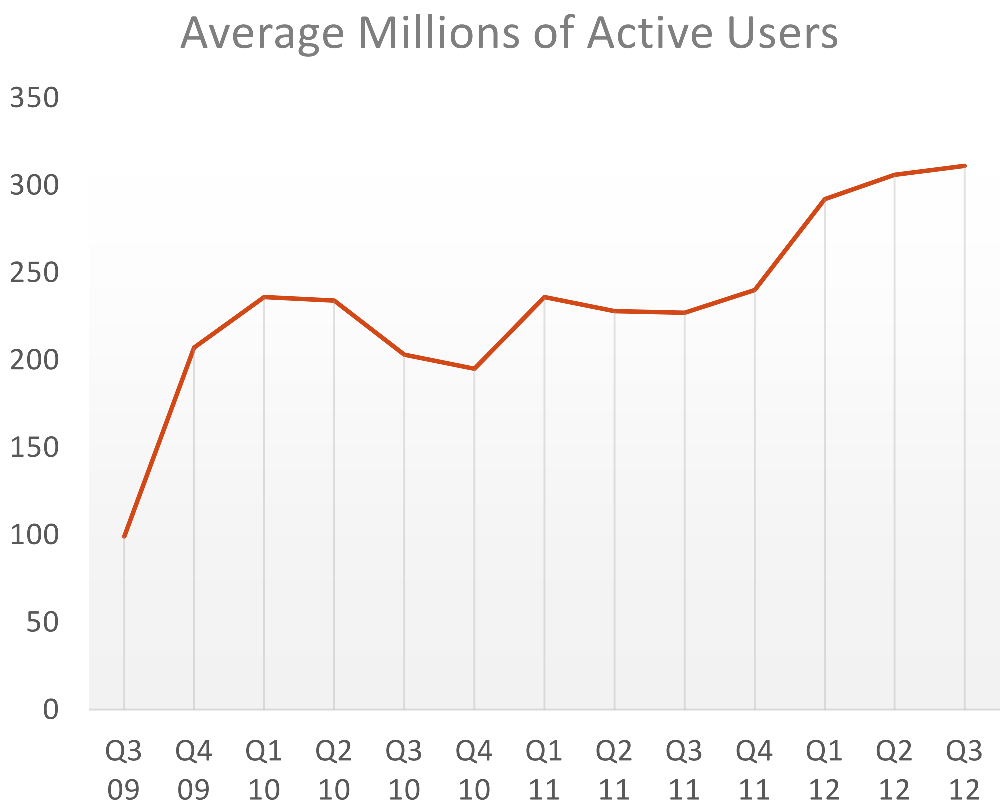
Databases are Great

Developer ease via ACID

Turing Award winning great

But they are Rigid and Complex

Growth…



Rapid growth of some web services led to design of new “web-scale” databases…

Rise of NoSQL

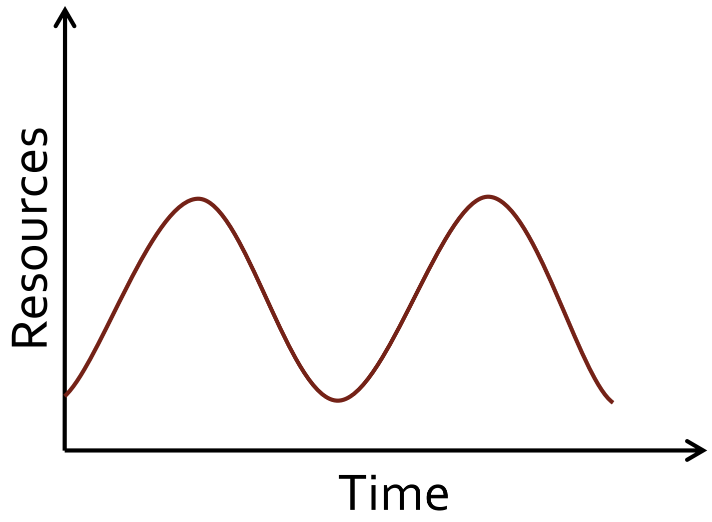
Scaling is needed

Chisel away at functionality

- No transactions
- No secondary indexes
- Minimal recovery
- Mixed Consistency

Not always suitable…

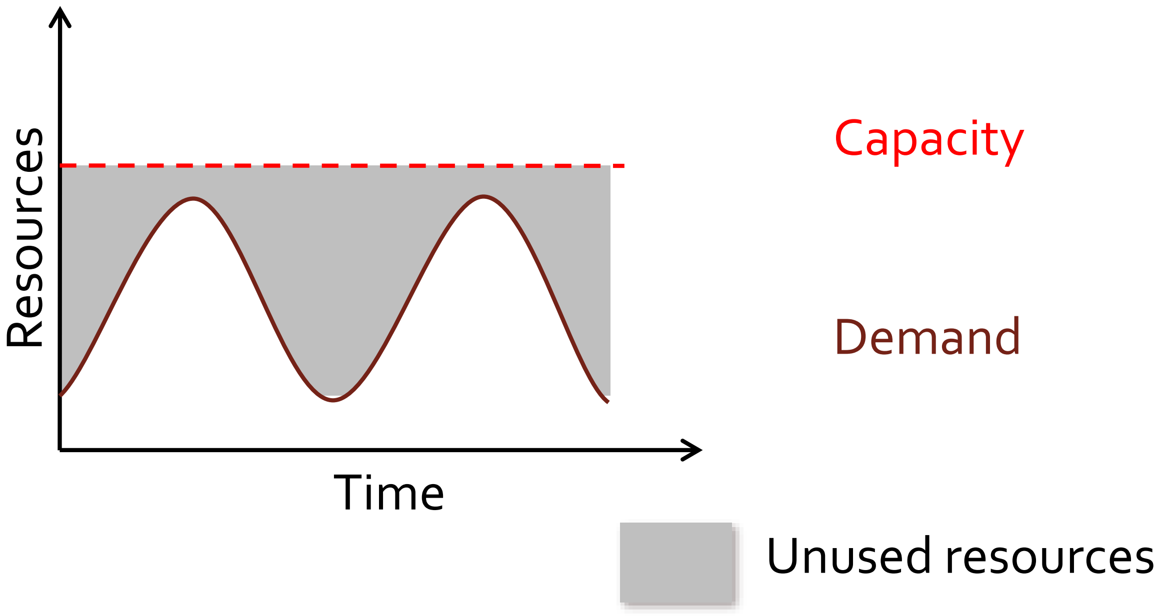
Workloads Fluctuate



Demand

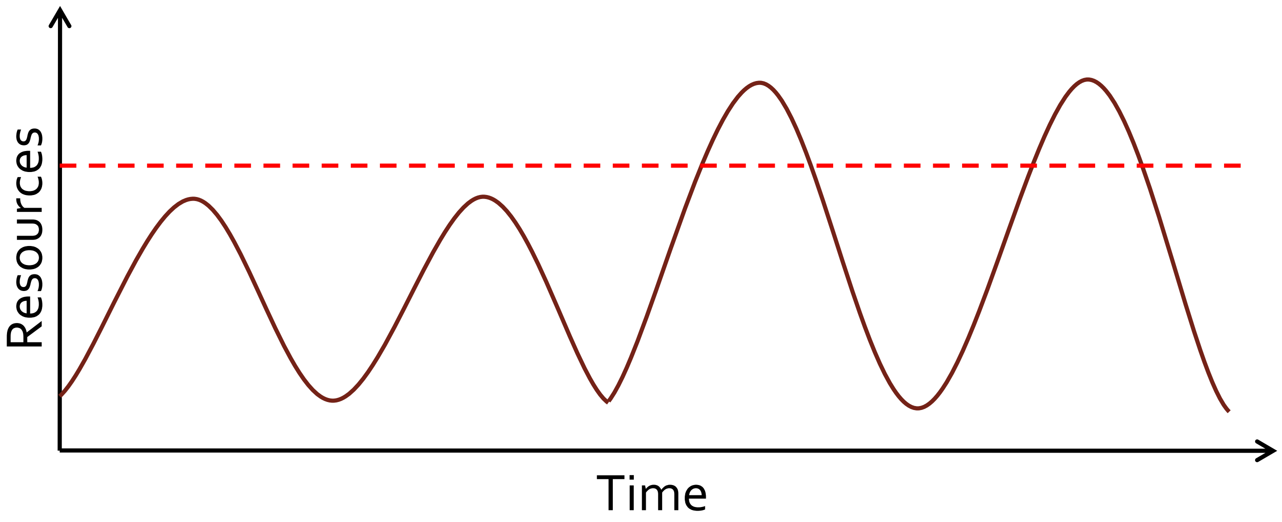
Slide Credits: Berkeley RAD Lab

Peak Provisioning



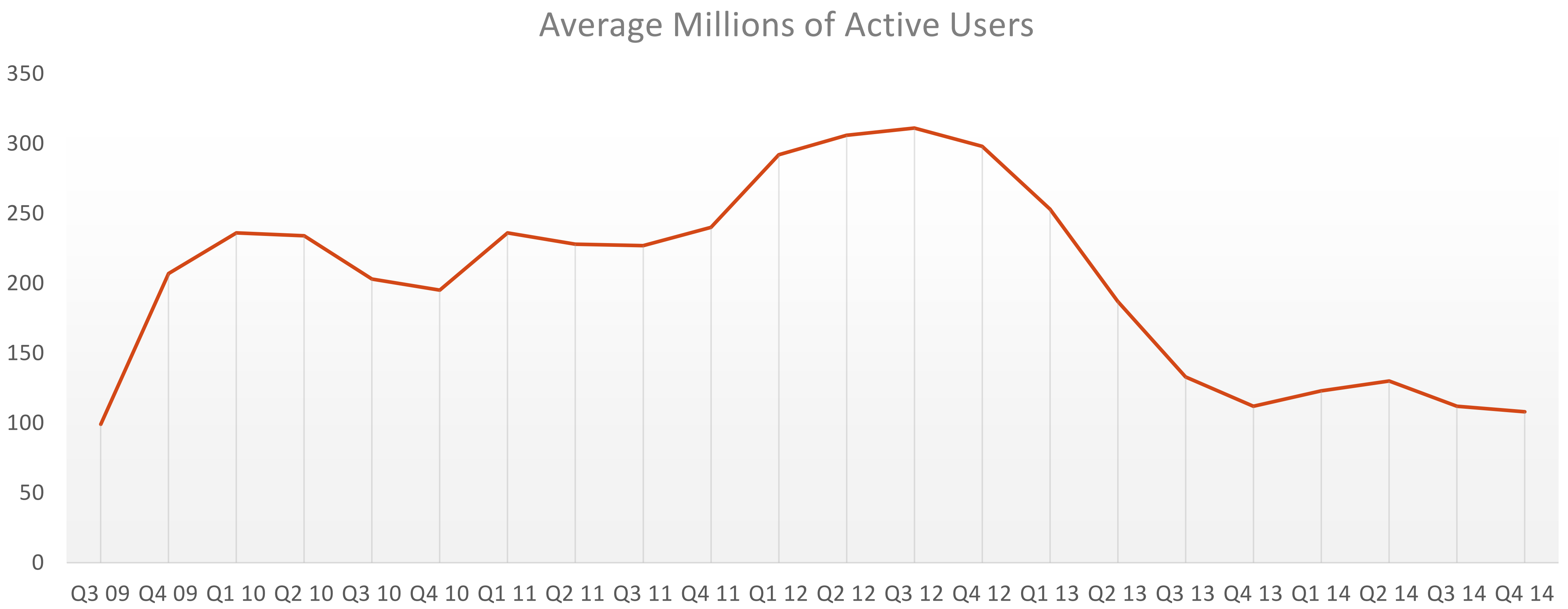
Slide Credits: Berkeley RAD Lab

Peak Provisioning isn’t Perfect



Slide Credits: Berkeley RAD Lab

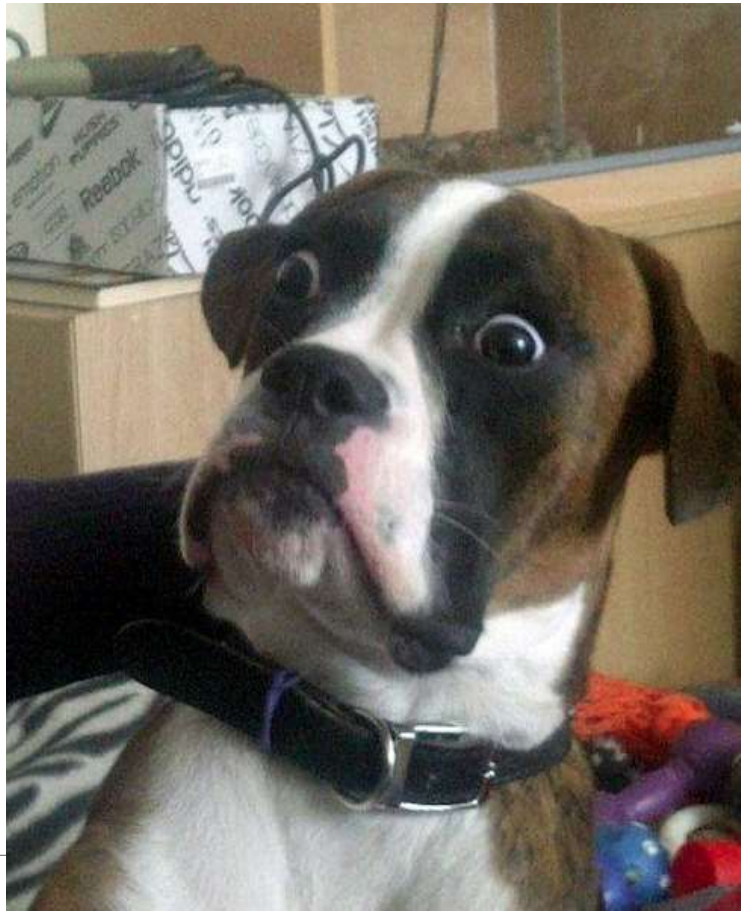
Growth is not always sustained



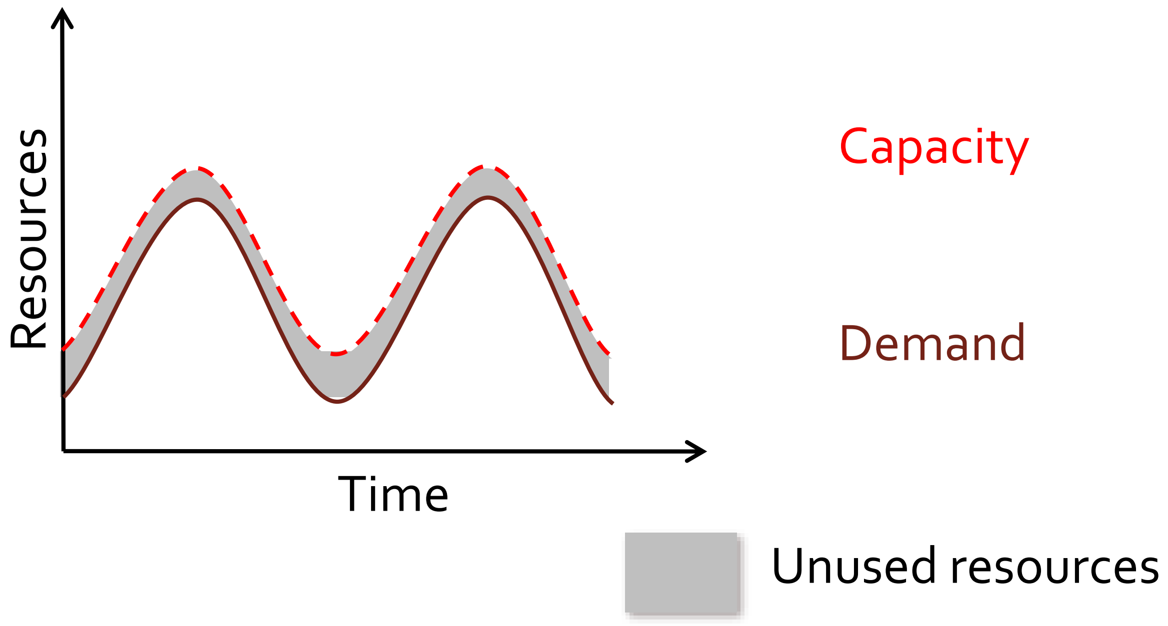
http://www.statista.com/statistics/273569/monthly-active-users-of-zynga-games/

Need Elasticity

ELASTICITY > SCALABILITY



The Promise of Elasticity



Slide Credits: Berkeley RAD Lab

Primary use-cases for elasticity

Database-as-a-Service with elastic placement of non-correlated tenants, often low utilization per tenant.

High-throughput transactional systems (OLTP)

No Need to Weaken the Database!

High Throughput = Main Memory

Cost per GB for RAM is dropping.

Network memory is faster than local disk.

Much faster than disk based DBs.

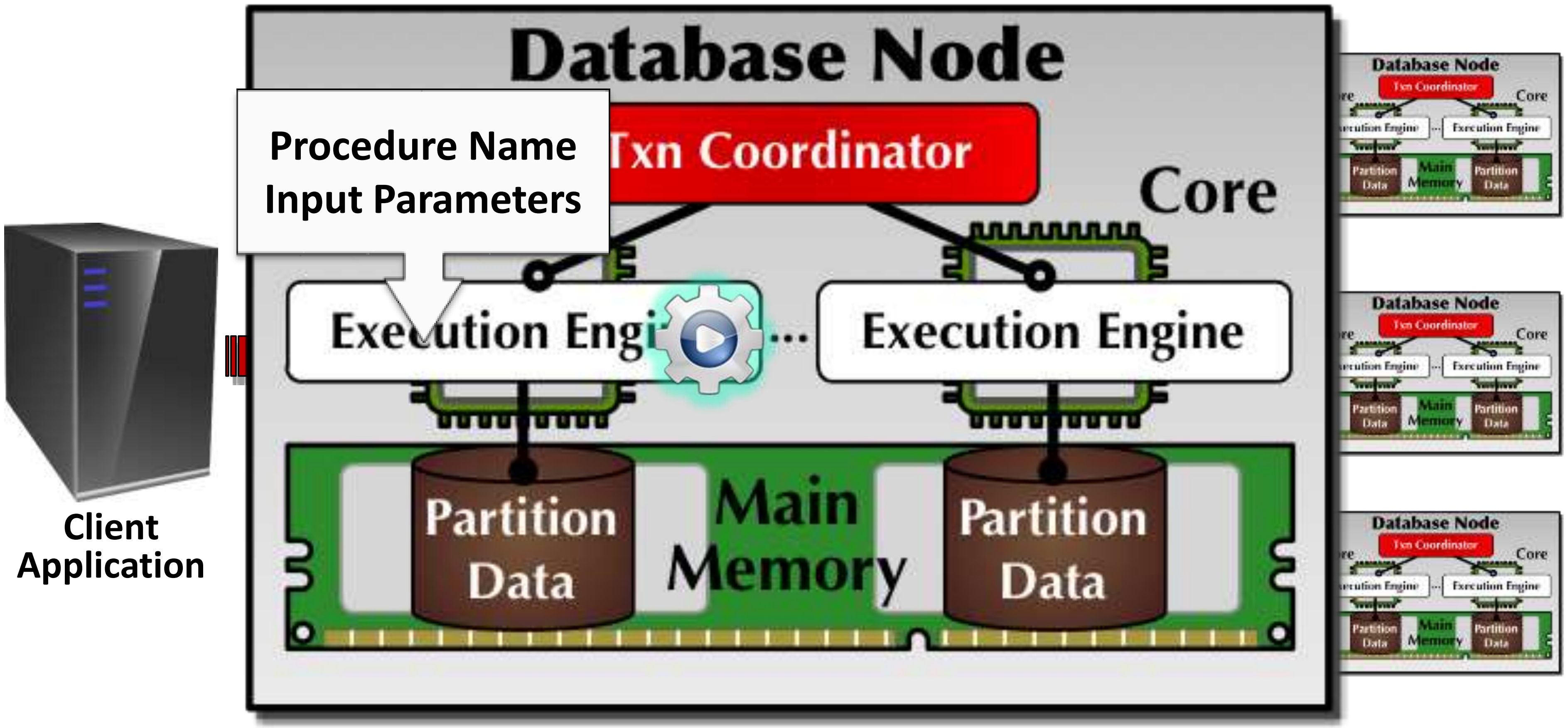
Approaches for “NewSQL” main-memory*

Highly concurrent, latch-free data structures

Partitioning into single-threaded executors

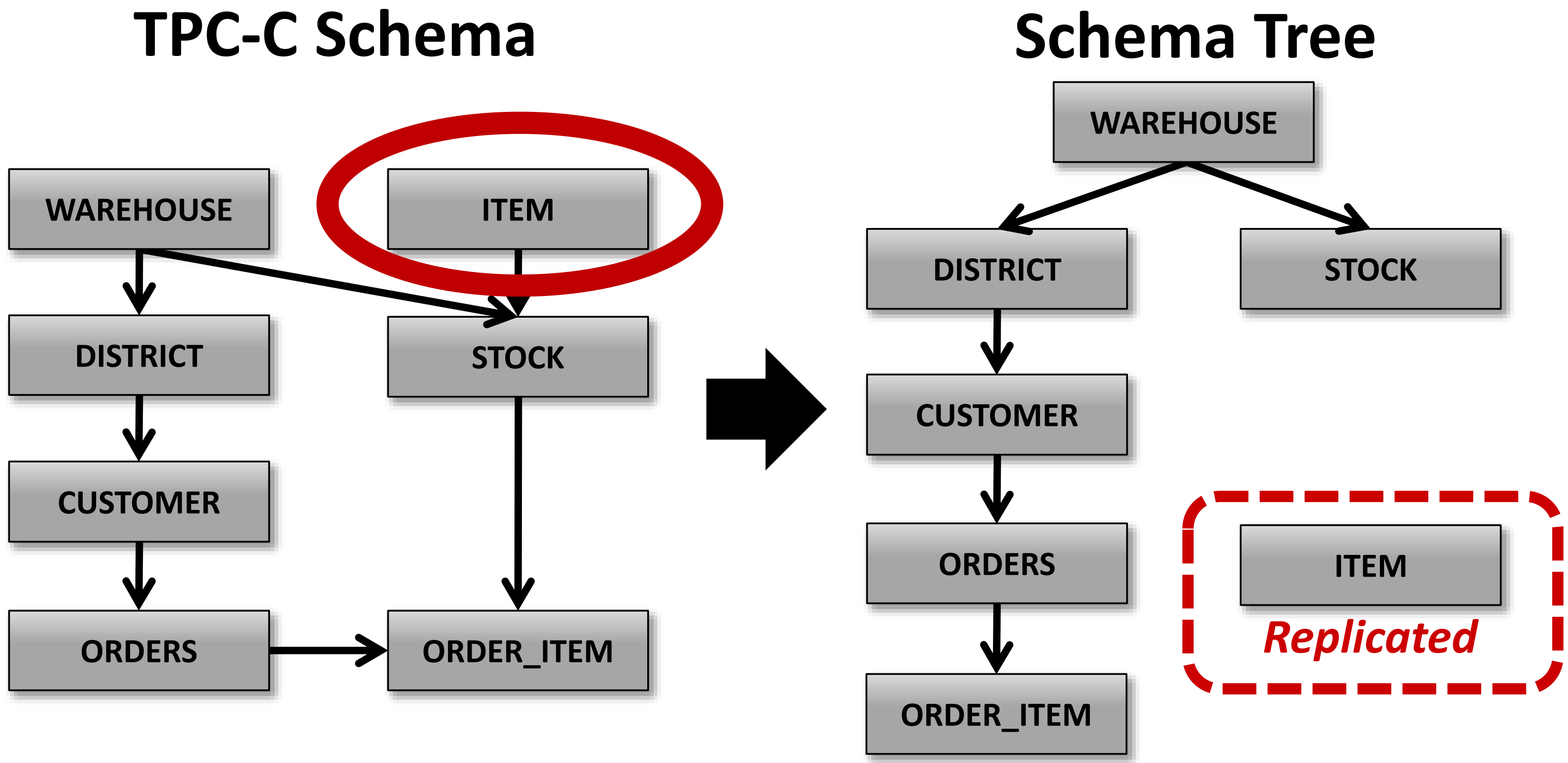
*Excuse the generalization

H-Store

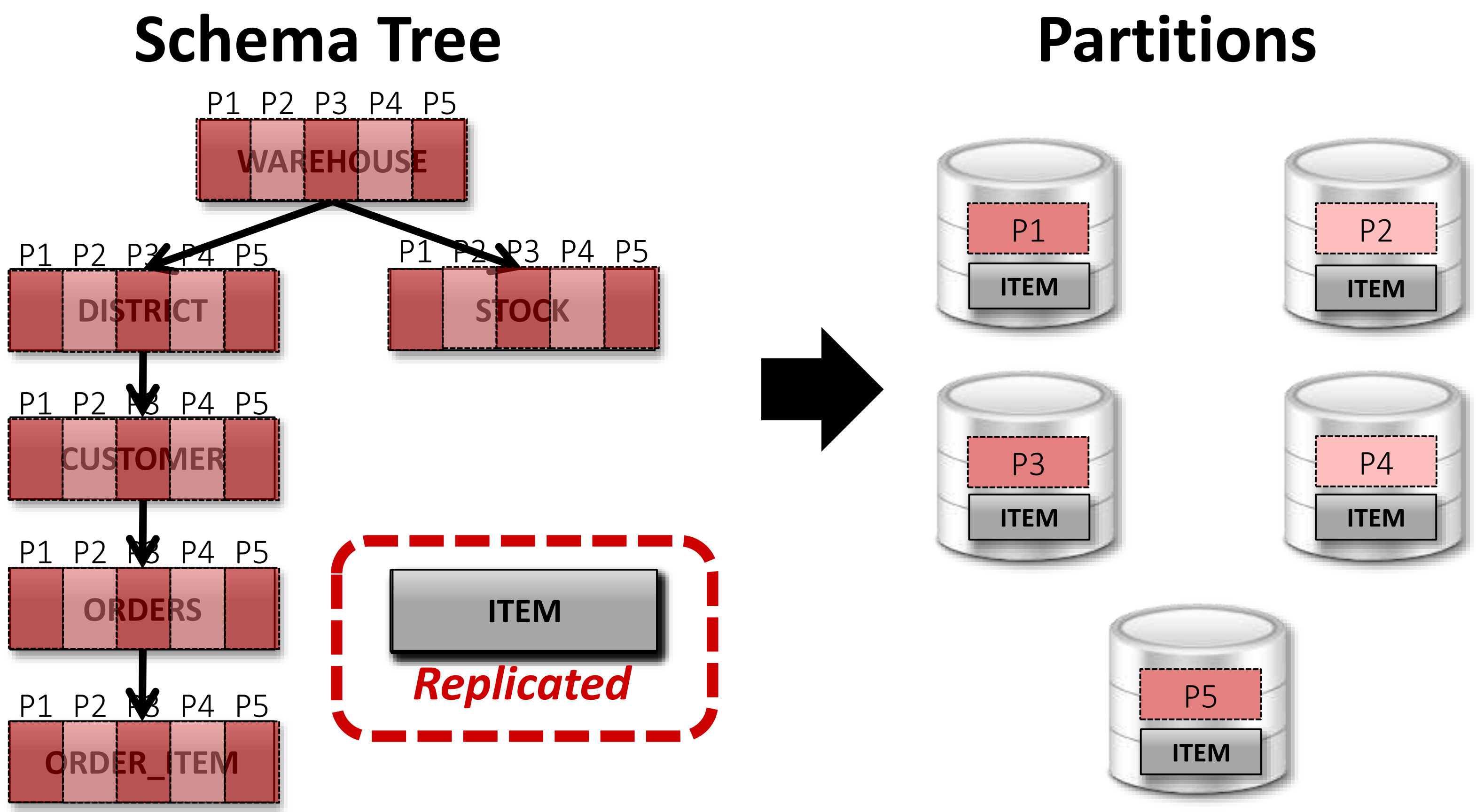


Database Node
Txn Coordinator
Core
Procedure Name
Input Parameters
Execution Engine
...
Execution Engine
Partition Data
Main Memory
Partition Data
Client Application
Database Node
Core
Execution Engine
Partition Data
Main Memory
Partition Data
Database Node
Core
Execution Engine
Partition Data
Main Memory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

Database Partitioning



Database Partitioning



The Problem: Workload Skew

Many OLTP applications suffer from variable load and high skew:

Extreme Skew: 40-60% of NYSE trading volume is on 40 individual stocks

Time Variation: Load "follows the sun"

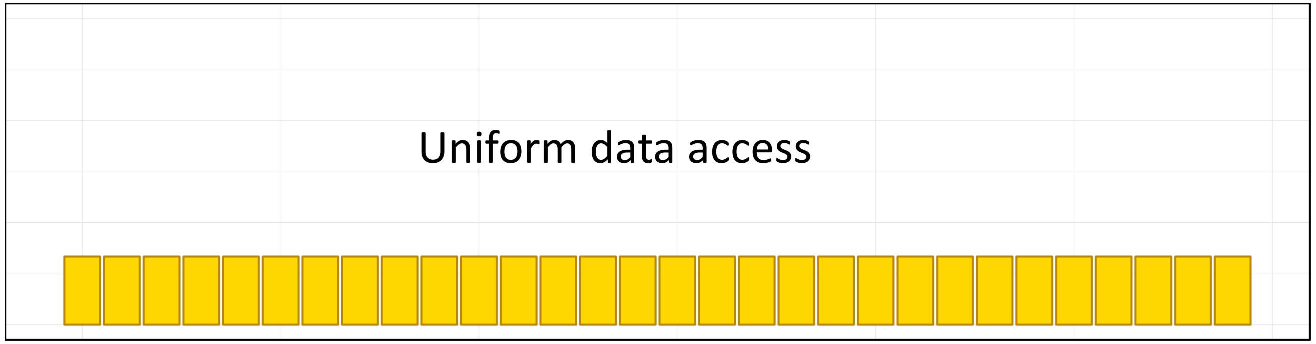
Seasonal Variation: Ski resorts have high load in the winter months

Load Spikes: First and last 10 minutes of trading day have 10X the average volume

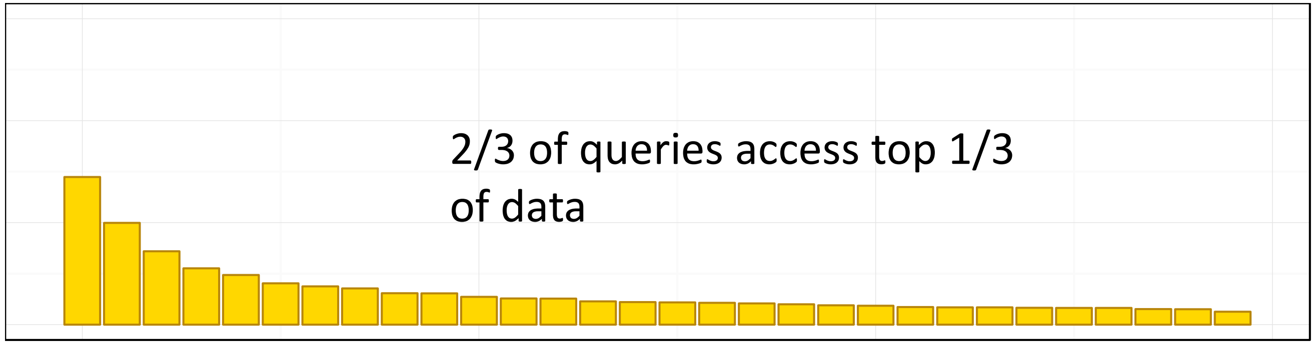
Hockey Stick Effect: A new application goes "viral"

The Problem: Workload Skew

No Skew



Low Skew

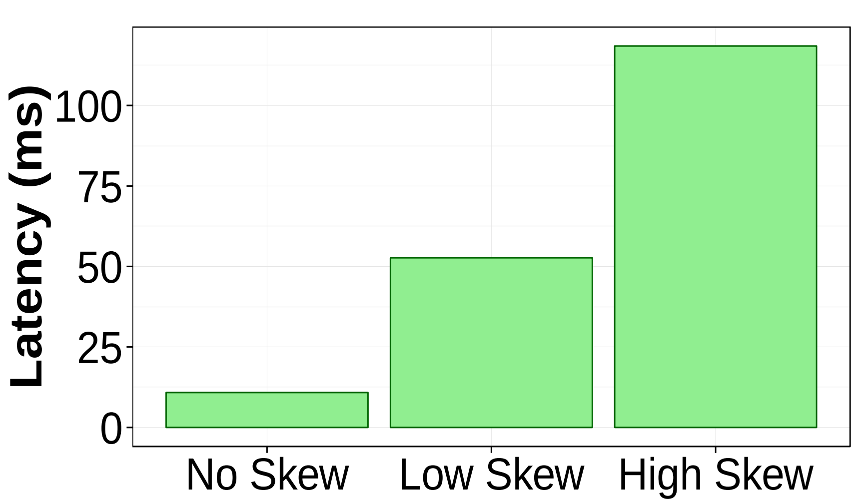


High Skew

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!)
- Limit load on system (Poor performance!)
- Enable system to elastically scale in or out to dynamically adapt to changes in load

Elastic Scaling

Key	Value
1	kjfhlaks
2	ewoiej
3	fsakjfkf
4	fkjwei

Key	Value
5	wekjxmn
6	fmeixtm
7	ewjkdm
8	weiuqw

Key	Value
9	mznjku
10	ewrenx
11	qieucm
12	roiwrio

Key	Value
1	kjfhlaks
2	ewoiej
3	fsakjfkf
4	fkjwei

Key	Value
5	wekjxmn
6	fmeixtm
7	ewjkdm
8	weiuqw

Key	Value
9	mznjku
10	ewrenx

Key	Value
11	qieucm
12	roiwrio

Load Balancing

Key	Value
1	kjfhlaks
2	ewoiej
3	fsakjfkf
4	fkjwei

Key	Value
5	wekjxmn
6	fmeixtm
7	ewjkdm
8	weiuqw

Key	Value
9	mznjku
10	ewrenx
11	qieucm
12	roiwrio

Key	Value
1	kjfhlaks
2	ewoiej
3	fsakjfkf
4	fkjwei
10	ewrenx
11	qieucm

Key	Value
5	wekjxmn
6	fmeixtm
7	ewjkdm
8	weiuqw
12	roiwrio

Key	Value
9	mznjku

Two-Tiered Partitioning

What if only a few specific tuples are very hot? Deal with them separately!

Two tiers:

1. Individual hot tuples, mapped explicitly to partitions
2. Large blocks of colder tuples, hash- or range-partitioned at coarse granularity

Possible implementations:

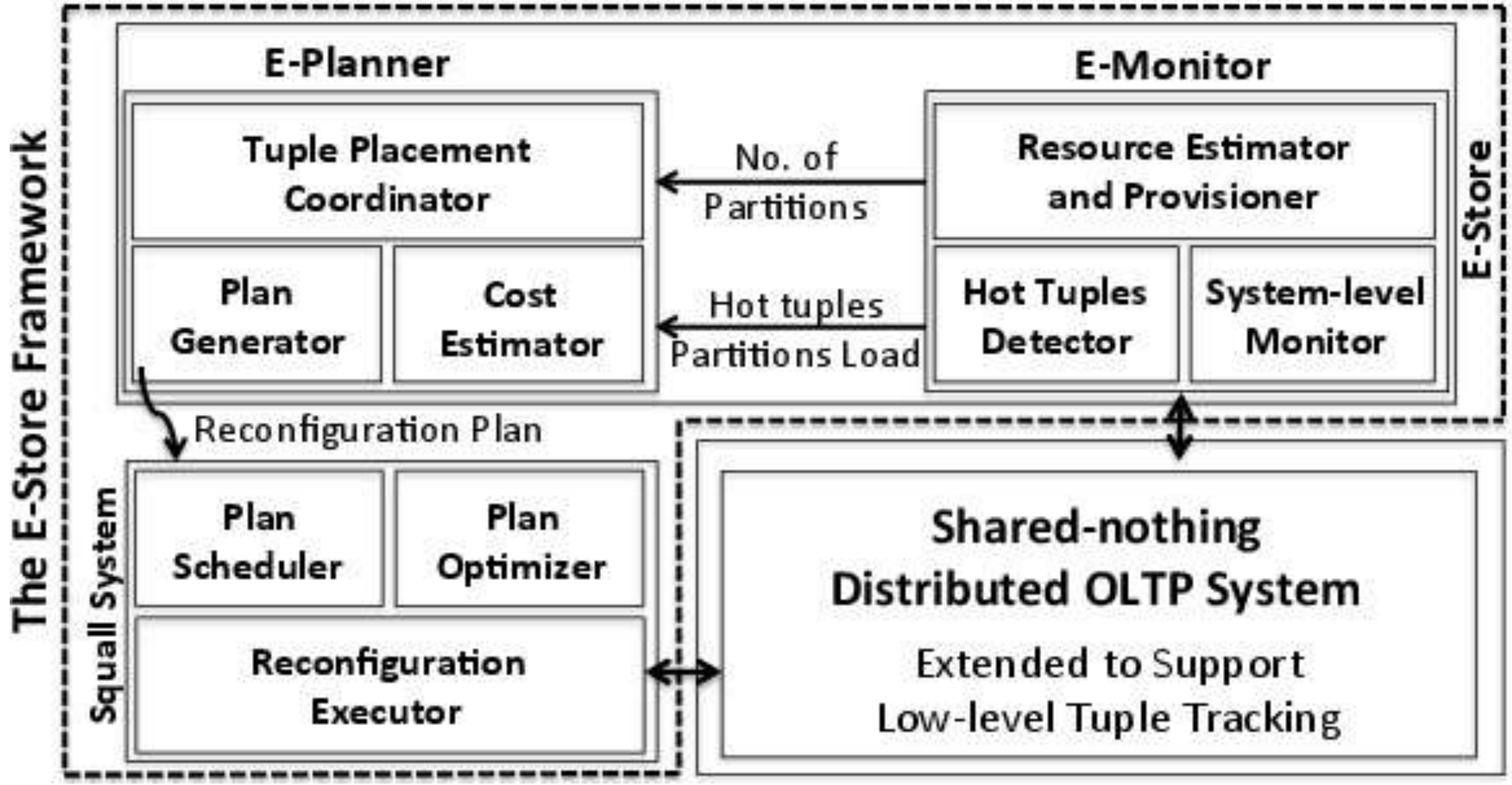
- Fine-grained range partitioning
- Consistent hashing with virtual nodes
- Lookup table combined with any standard partitioning scheme

Existing systems are “one-tiered” and partition data only at course granularity

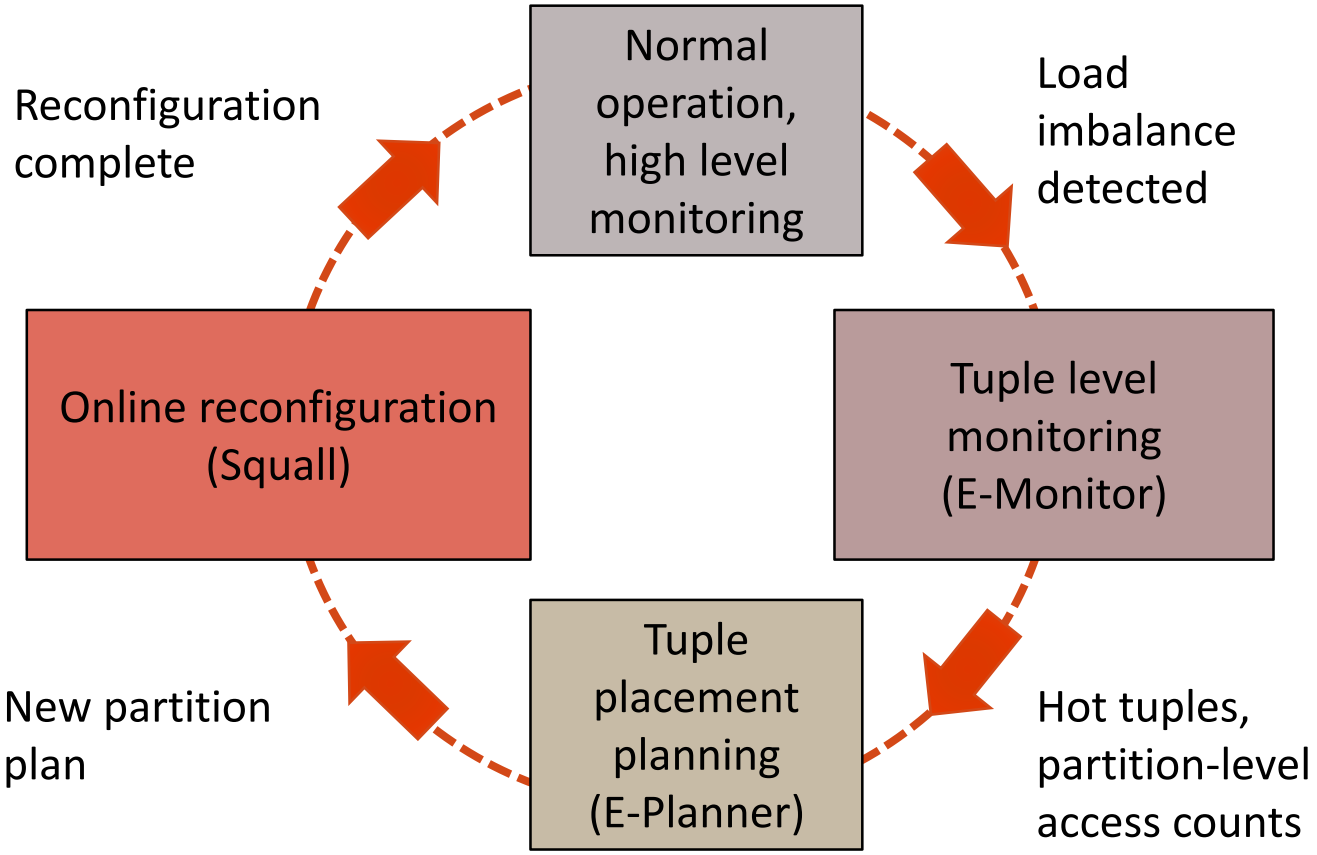
- Unable to handle cases of extreme skew

E-Store

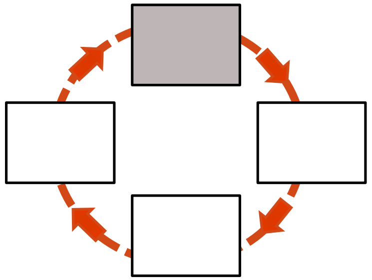
End-to-end system which extends H-Store (a distributed, shared-nothing, main memory DBMS) with automatic, adaptive, two-tiered elastic partitioning



E-Store



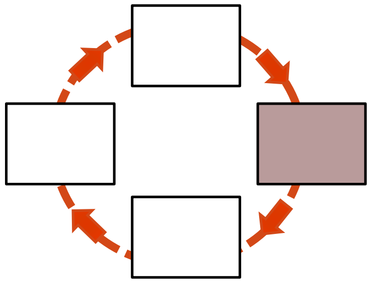
Normal operation, high level monitoring
Load imbalance detected
Tuple level monitoring (E-Monitor)
Hot tuples, partition-level access counts
Tuple placement planning (E-Planner)
New partition plan
Online reconfiguration (Squall)
Reconfiguration complete



E-Monitor: High-Level Monitoring

High level system statistics collected every ~1 minute

- CPU indicates system load, used to determine whether to add or remove nodes, or re-shuffle the data
- Accurate in H-Store since partition executors are pinned to specific cores
- Cheap to collect
- When a load imbalance (or overload/underload) is detected, detailed monitoring is triggered



E-Monitor: Tuple-Level Monitoring

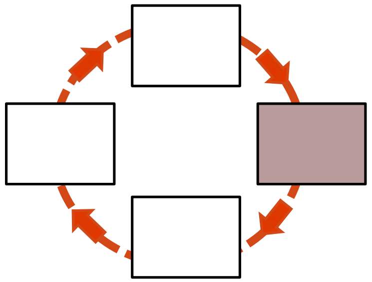
Tuple-level statistics collected in case of load imbalance

- Finds the top 1% of tuples accessed per partition (read or written) during a 10 second window
- Finds total access count per block of cold tuples

Can be used to determine workload distribution, using tuple access count as a proxy for system load

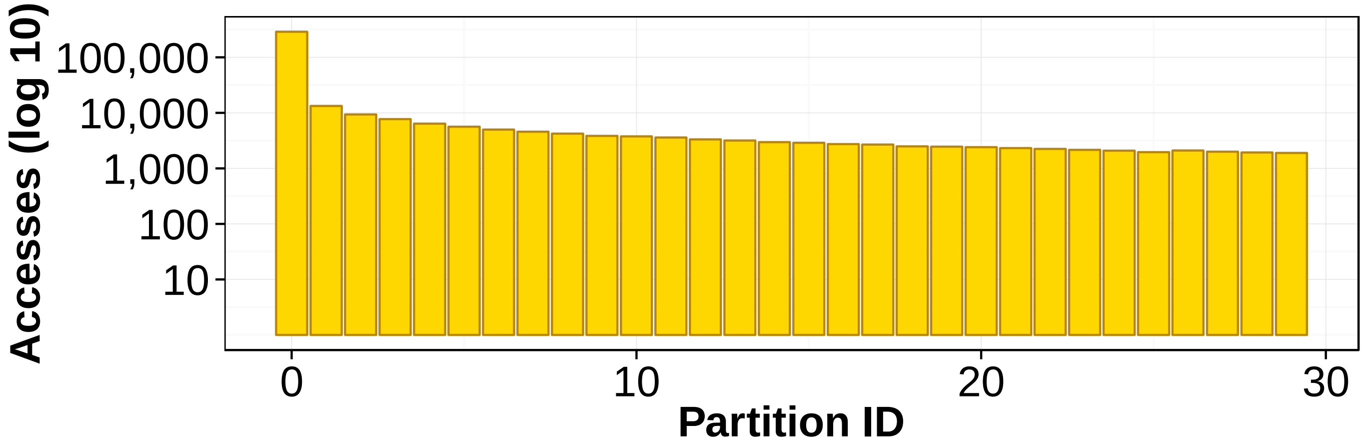
- Reasonable assumption for main-memory DBMS w/ OLTP workload

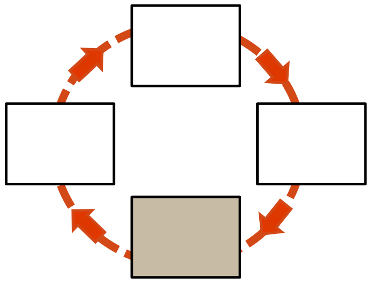
Minor performance degradation during collection



E-Monitor: Tuple-Level Monitoring

Sample output





E-Planner

Given current partitioning of data, system statistics and hot tuples/partitions from E-Monitor, E-Planner determines:

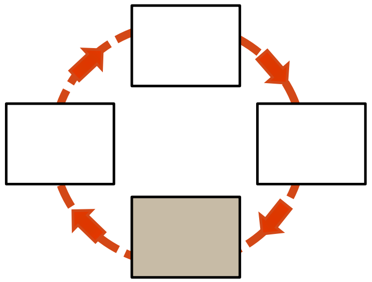
- Whether to add or remove nodes
- How to balance load

Optimization problem: minimize data movement (migration is not free) while balancing system load.

We tested five different data placement algorithms:

- One-tiered bin packing (ILP – computationally intensive!)
- Two-tiered bin packing (ILP – computationally intensive!)
- First Fit (global repartitioning to balance load)
- Greedy (only move hot tuples)
- Greedy Extended (move hot tuples first, then cold blocks until load is balanced)

E-Planner: Greedy Extended Algorithm



Hot tuples	Accesses
0	20,000
1	12,000
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…

```
Current YCSB partition
plan

"usertable": {

  0: [0-100000)
  1: [100000-200000)
  2: [200000-300000)

}
```

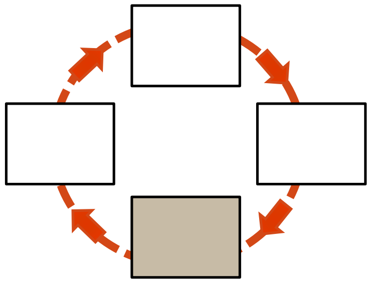
?

```
New YCSB partition
plan

"usertable": {

  0: [1000-100000)
  1: [1-2),[100000-200000)
  2: [200000-300000),[0-1),
     [2-1000)

}
```



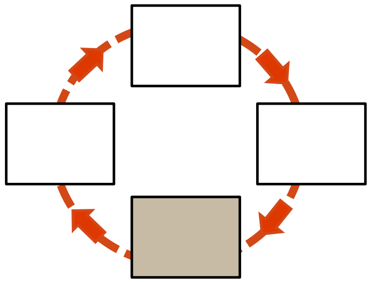
E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[0-100000)	77,000
1	[100000-200000)	23,000
2	[200000-300000)	5,000

Target cost per partition: 35,000

Hot tuples	Accesses
0	20,000
1	12,000
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…



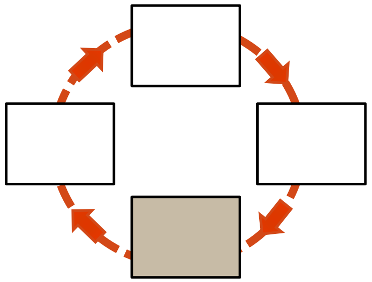
E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[0-100000)	77,000
1	[100000-200000)	23,000
2	[200000-300000)	5,000

Target cost per partition: 35,000

Hot tuples	Accesses
0	20,000
1	12,000
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…



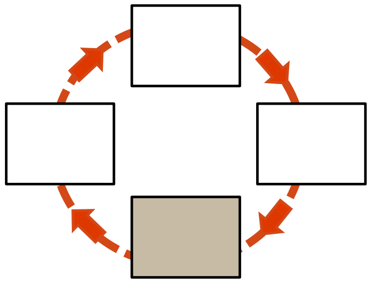
E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[1-100000)	57,000
1	[100000-200000)	23,000
2	[200000-300000),[0-1)	25,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
1	12,000
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…



E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[1-100000)	57,000
1	[100000-200000)	23,000
2	[200000-300000),[0-1)	25,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
1	12,000
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…

E-Planner: Greedy Extended Algorithm

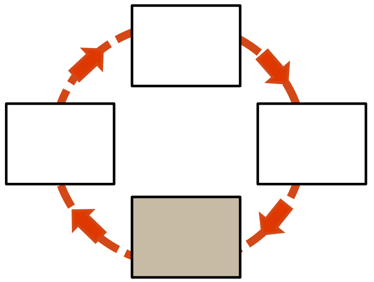
Partition	Keys	Total Cost (tuple accesses)
0	[1-100000)	57,000
1	[100000-200000)	23,000
2	[200000-300000),[0-1)	25,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
1	12,000
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…

E-Planner: Greedy Extended Algorithm



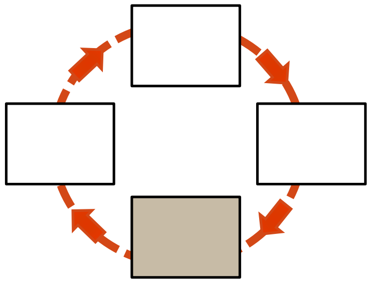
Partition	Keys	Total Cost (tuple accesses)
0	[2-100000)	45,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1)	25,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…

E-Planner: Greedy Extended Algorithm

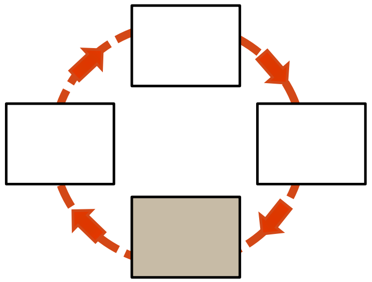


Partition	Keys	Total Cost (tuple accesses)
0	[2-100000)	45,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1)	25,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…



E-Planner: Greedy Extended Algorithm

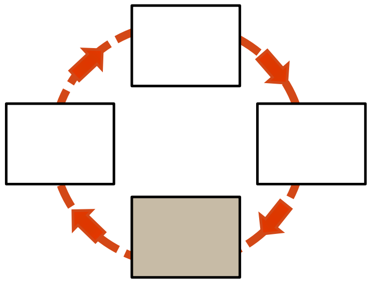
Partition	Keys	Total Cost (tuple accesses)
0	[2-100000)	45,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1)	25,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
2	5,000

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…

E-Planner: Greedy Extended Algorithm

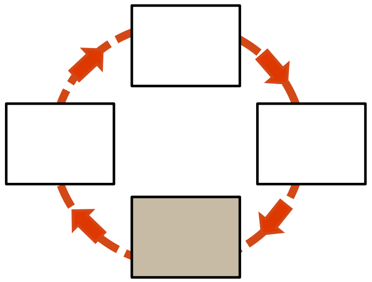


Partition	Keys	Total Cost (tuple accesses)
0	[3-100000)	40,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1), [2-3)	30,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
~~2~~	~~5,000~~

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…



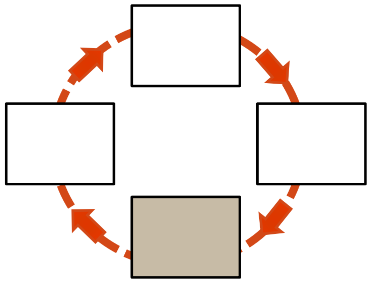
E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[3-100000)	40,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1), [2-3)	30,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
~~2~~	~~5,000~~

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…



E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[3-100000)	40,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1), [2-3)	30,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
~~2~~	~~5,000~~

Range	Accesses
3-1000	5,000
1000-2000	3,000
2000-3000	2,000
…	…

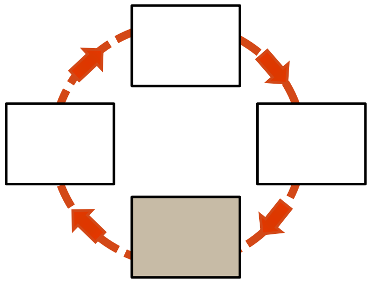
E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[1000-100000)	35,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1), [2-1000)	35,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
~~2~~	~~5,000~~

Range	Accesses
~~3-1000~~	~~5,000~~
1000-2000	3,000
2000-3000	2,000
…	…



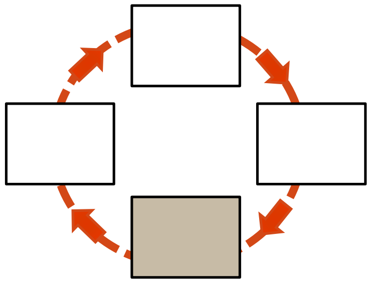
E-Planner: Greedy Extended Algorithm

Partition	Keys	Total Cost (tuple accesses)
0	[1000-100000)	35,000
1	[100000-200000), [1-2)	35,000
2	[200000-300000),[0-1), [2-1000)	35,000

Target cost per partition: 35,000

Hot tuples	Accesses
~~0~~	~~20,000~~
~~1~~	~~12,000~~
~~2~~	~~5,000~~

Range	Accesses
~~3-1000~~	~~5,000~~
1000-2000	3,000
2000-3000	2,000
…	…



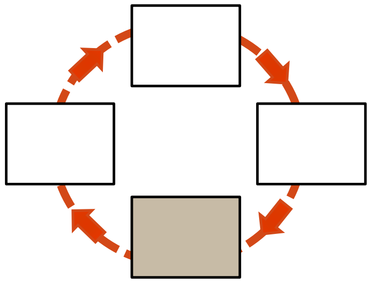
E-Planner: Other Heuristic algorithms

Greedy

- Like Greedy Extended, but the algorithm stops after all hot tuples have been moved
- If there **are not** many hot tuples (e.g. low skew), may not sufficiently balance the workload

First Fit

- First packs hot tuples onto partitions, filling one partition at a time
- Then packs blocks of cold tuples, filling the remaining partitions one at a time
- Results in a balanced workload, but **does not attempt to limit** the amount of data movement



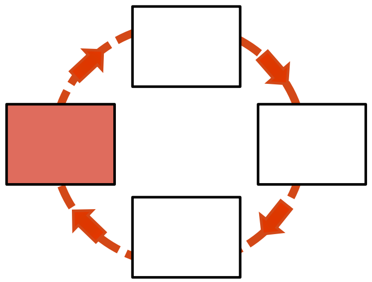
E-Planner: Optimal Algorithms

Two-Tiered Bin Packer

- Uses Integer Linear Programming (ILP) to optimally pack hot tuples and cold blocks onto partitions
- Constraints: each tuple/block must be assigned to exactly one partition, and each partition must have total load less than the average + 5%
- Optimization Goal: minimize the amount of data moved in order the satisfy the constraints

One-Tiered Bin Packer

- Like Two-Tiered Bin Packer, but can only pack blocks of tuples, not individual tuples
- Both are computationally intensive, but show one- and two-tiered approaches in best light



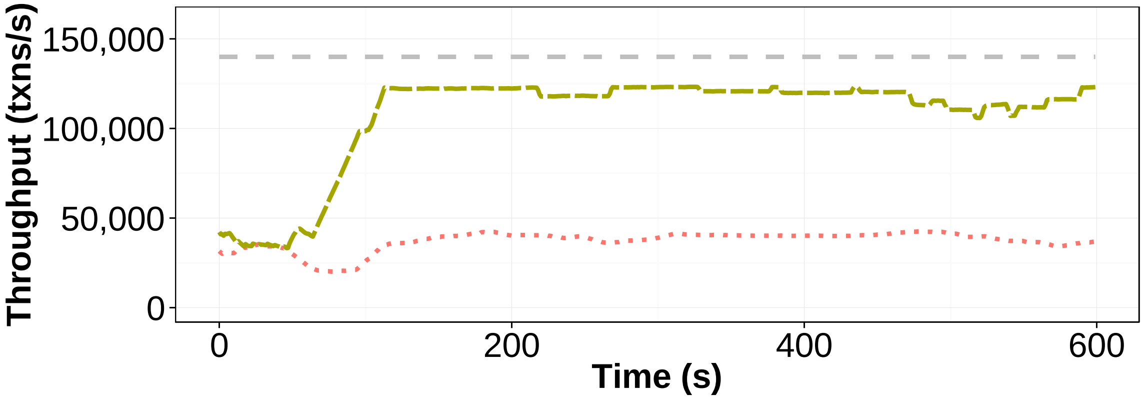
Squall

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

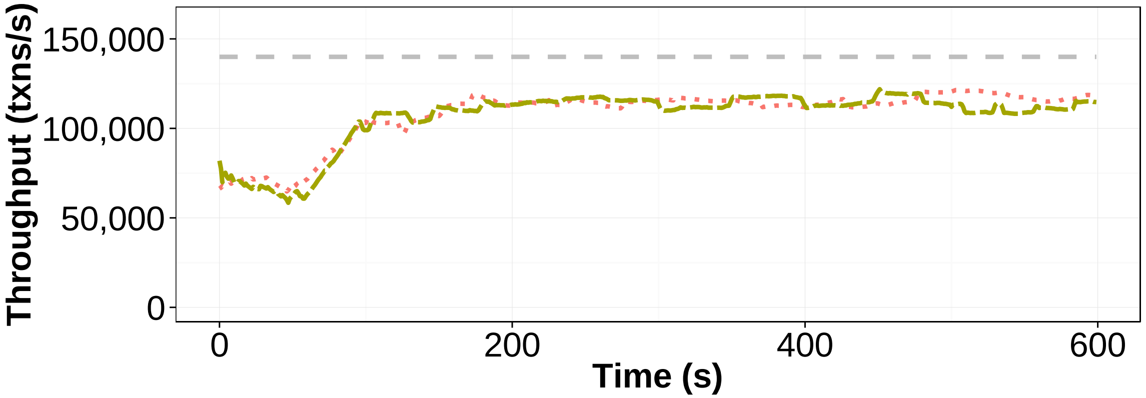
For immediate benefit, moves data from hottest partitions to coldest partitions first

More on this in a bit…

Results – Two-Tiered V. One-Tiered



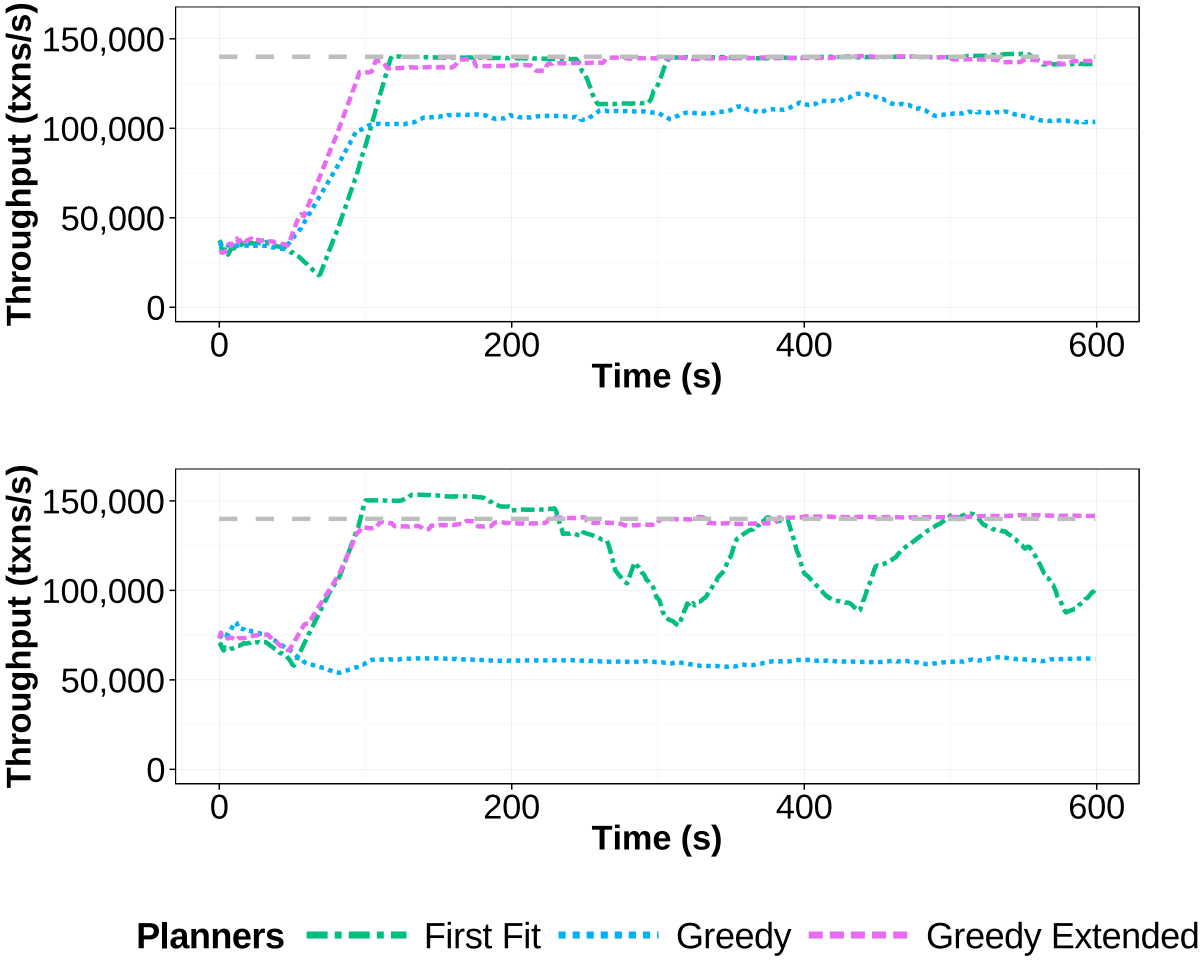
YCSB
High skew



YCSB
Low skew

Planners Bin Packer One Tiered Bin Packer Two Tiered

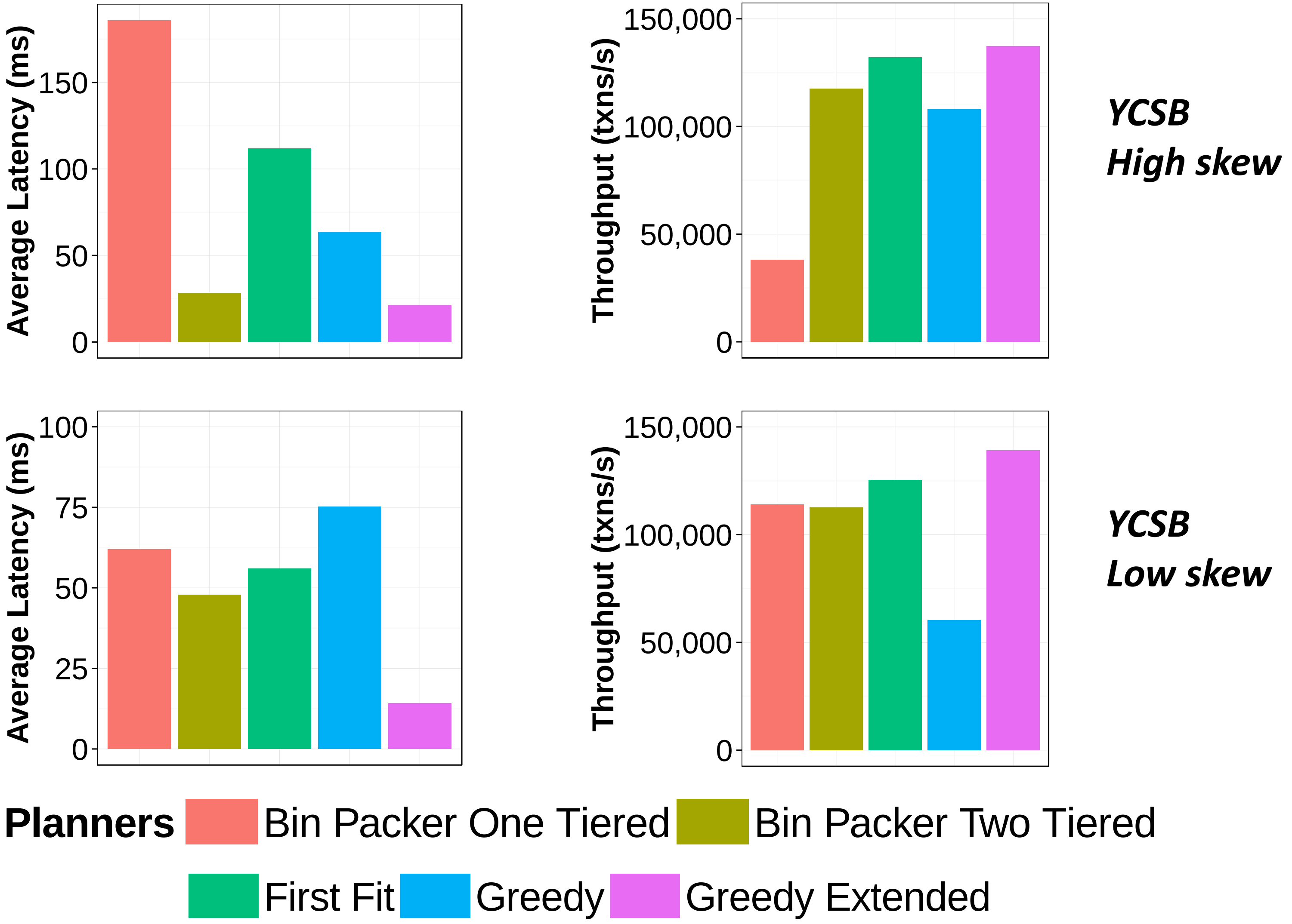
Results – Heuristic Planners



YCSB High skew

YCSB Low skew

Results


Average Latency (ms)
0
50
100
150
Throughput (txns/s)
0
50,000
100,000
150,000
YCSB
High skew
Average Latency (ms)
0
25
50
75
100
Throughput (txns/s)
0
50,000
100,000
150,000
YCSB
Low skew
Planners
Bin Packer One Tiered
Bin Packer Two Tiered
First Fit
Greedy
Greedy Extended

But What About…

Distributed Transactions???

Current E-Store does not take them into account when planning data movement

Ok when most transactions access a single partitioning key – tends to be the case for "tree schemas" such as YCSB, Voter, and TPC-C

E-Store++ will address the general case

- More later…

Squall

FINE-GRAINED LIVE RECONFIGURATION FOR PARTITIONED MAIN MEMORY DATABASES

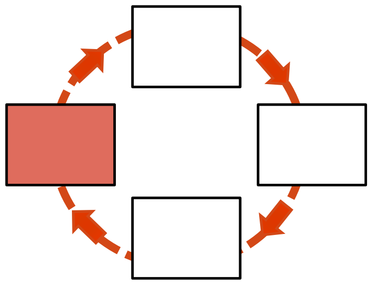
The Problem

Need to migrate tuples between partitions to reflect the updated partitioning.

Would like to do this without bringing the system offline:

- **Live Reconfiguration**

Similar to live migration of an entire database between servers.



Existing Solutions are Not Ideal

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.

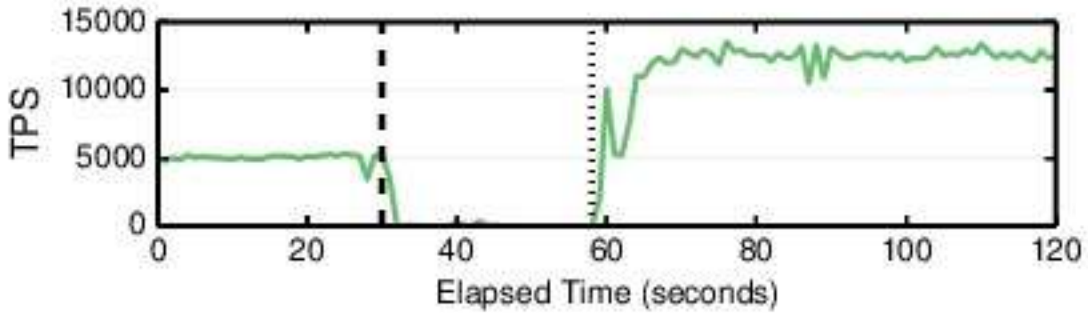
Slacker: Replication middleware based.

Not Your Parent's Migration

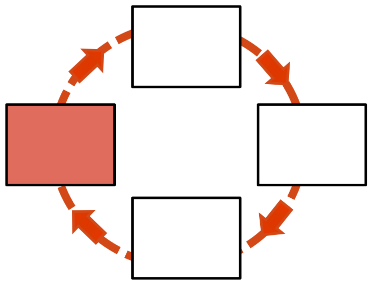
More than a **single** source and destination
- Want lightweight coordination

Single threaded execution model
- Either doing work or migration

Presence of distributed transactions and replication



Migrating 2 warehouses in TPC-C
In E-Store with a Zephyr like migration



Squall

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

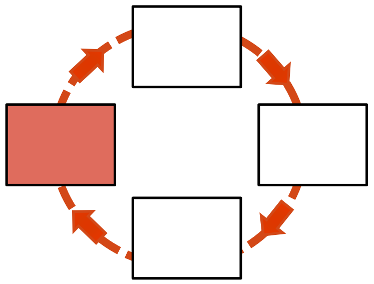
Conforms to H-Store single-threaded execution model

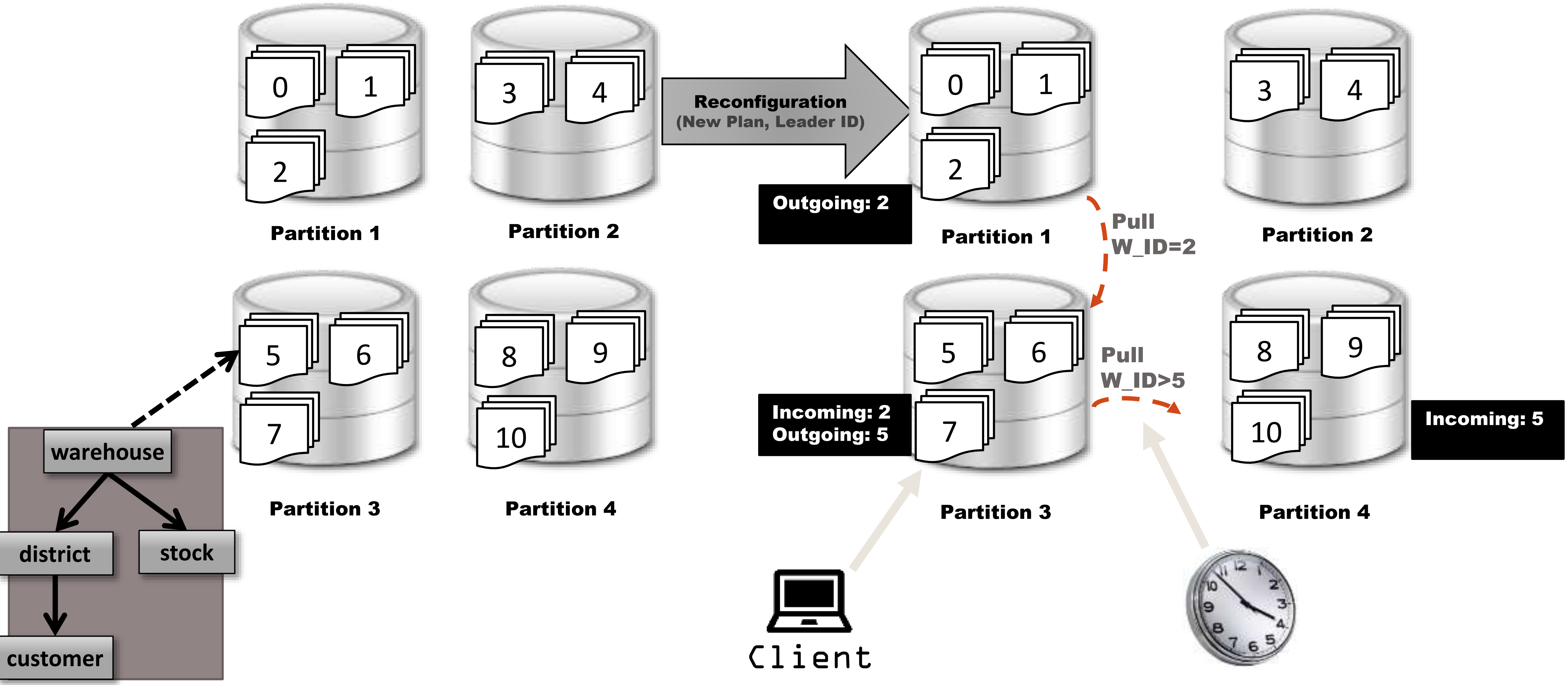
- While data is moving, transactions are blocked

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

Squall Steps

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





Keys to Performance

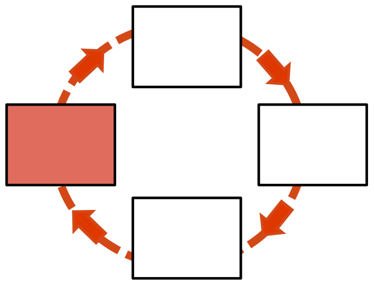
Redirect or pull only if needed.

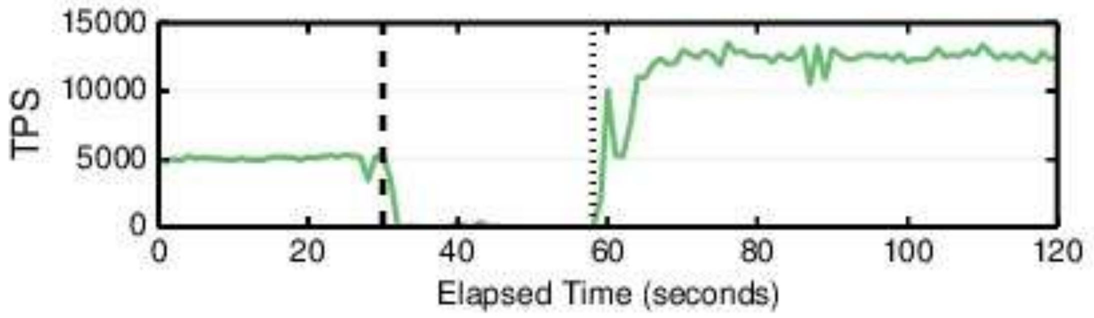
Properly **size** reconfiguration granule.

Split large reconfigurations to limit demands on single partition.

Tune what gets pulled.

Sometimes pull a little extra.

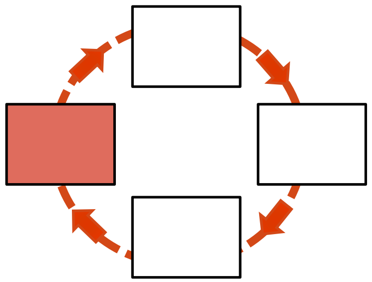




Migrating 2 warehouses in TPC-C
In E-Store with a Zephyr like migration

Redirect and Pull Only When Needed

Data Migration

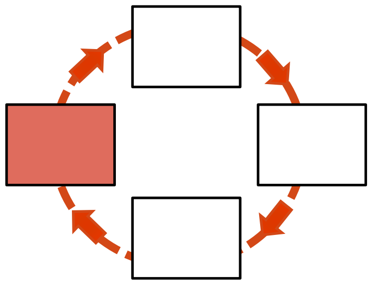


Query arrives, must be trapped to check if data is potentially moving. Check key map, then ranges list.

If either source or destination partition is local check their map, **keep local if possible**.

If neither partition is local, **forward to destination.**

If data is not moving, process transaction.

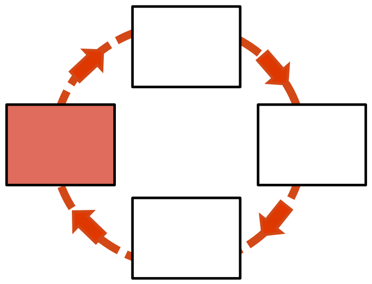


Trap for Data Movement

If txn requires incoming data, **block execution** and schedule data pull.

- Can only block dependent nodes in query plan
- Upon receipt mark and dirty tracking structures, and unblock.

If txn requires lost data, **restart** as distributed transaction or forward request.



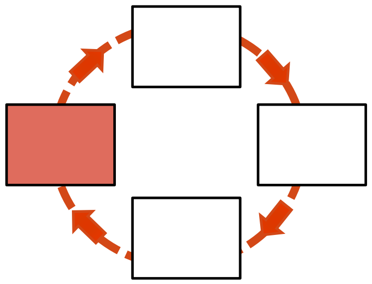
Data Pull Requests

Live data pulls are scheduled at destination as high priority transactions.

Current transaction finishes before extraction.

Timeout detection is needed.

Chunk Data for Asynchronous Pulls



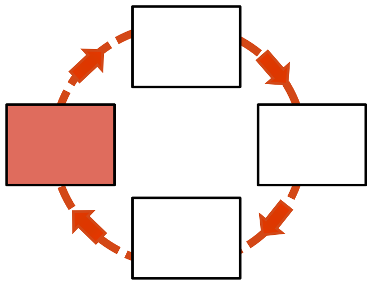
Why Chunk?

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.

Want a mechanism to support partial extraction while maintaining consistency.



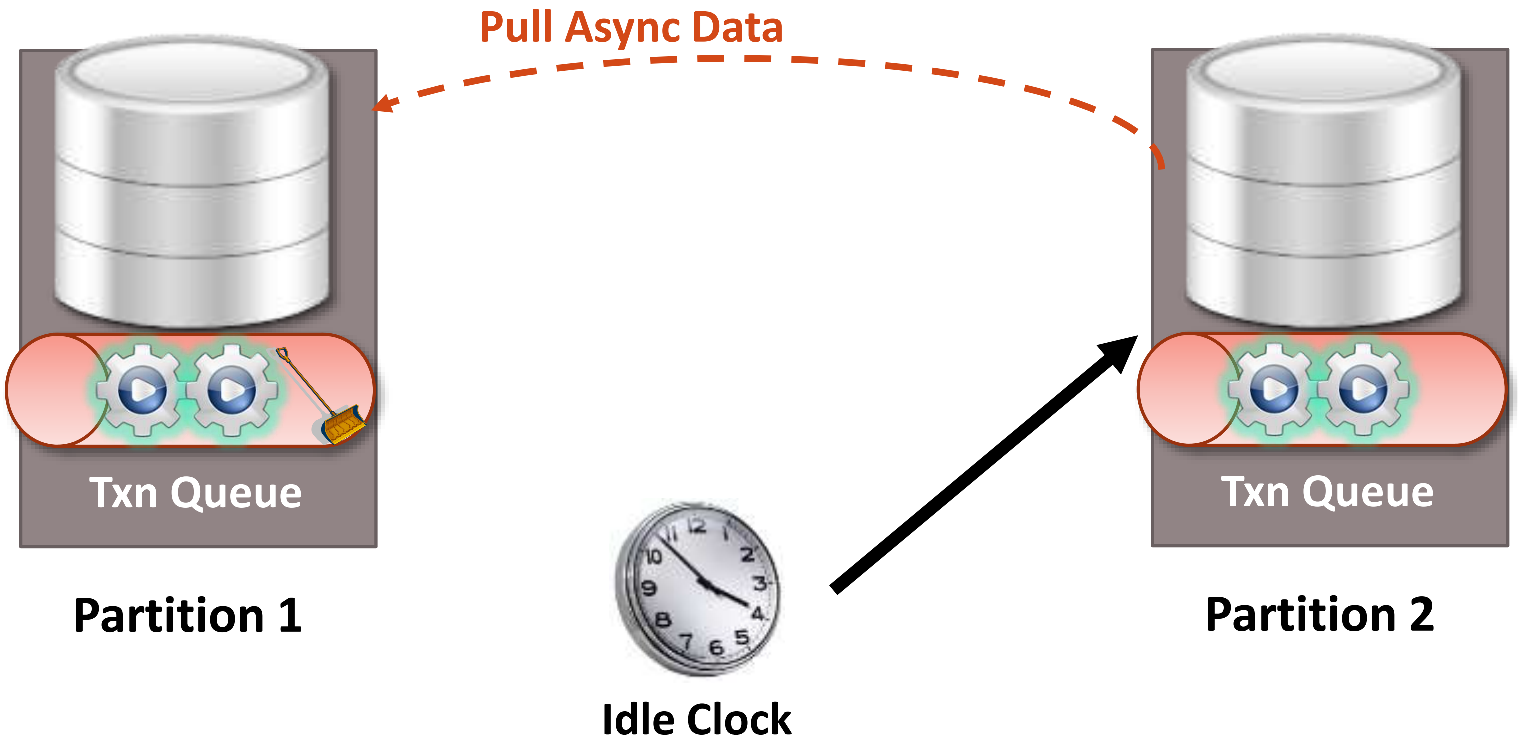
Async Pulls

Periodically pull chunks of cold data

These pulls are answered lazy

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

Mitigating Async Pulls



New Transactions Take Precedent

Extract up to Chunk Limit

Important to note data is partially migrated!

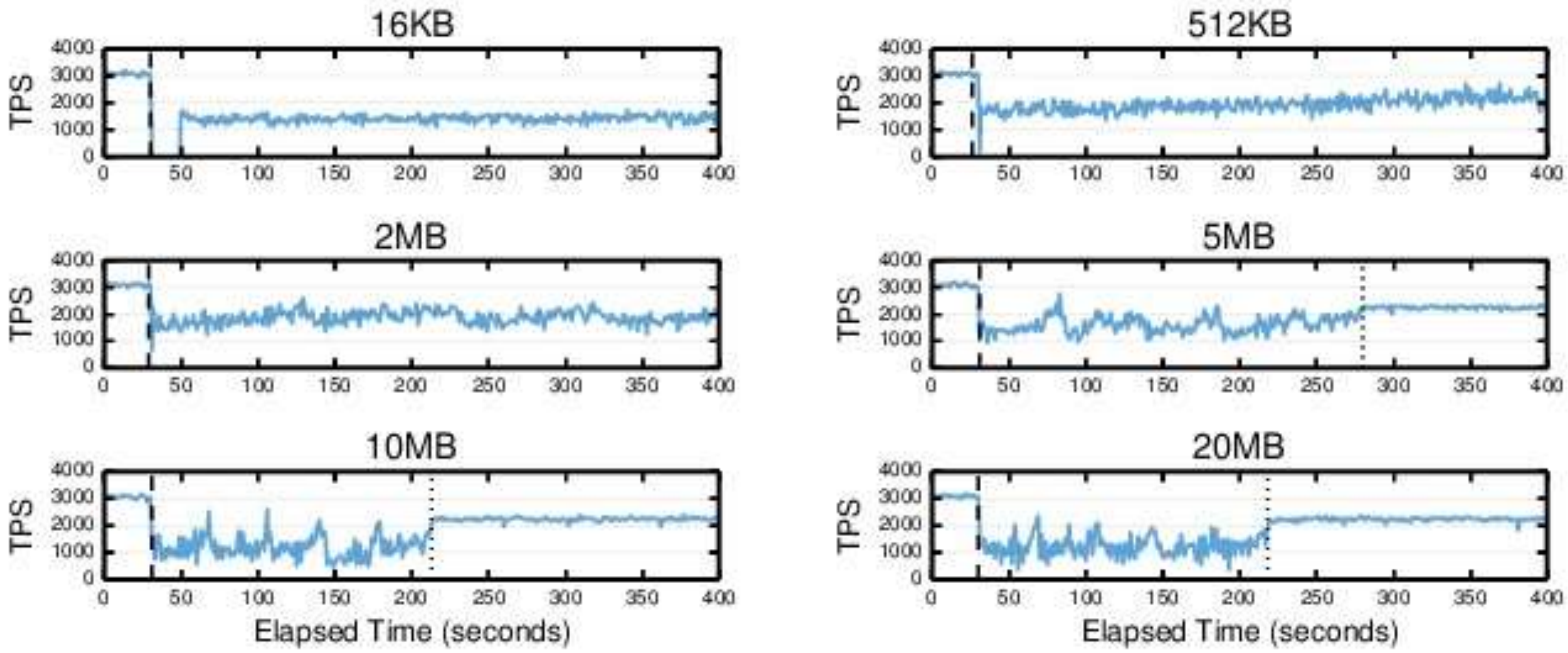
Repeat Until Complete

Repeat chunking until complete.
New transactions still take
precedent

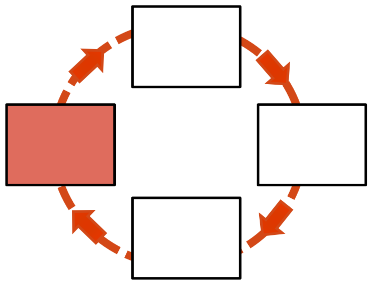
Sizing Chunks

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

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

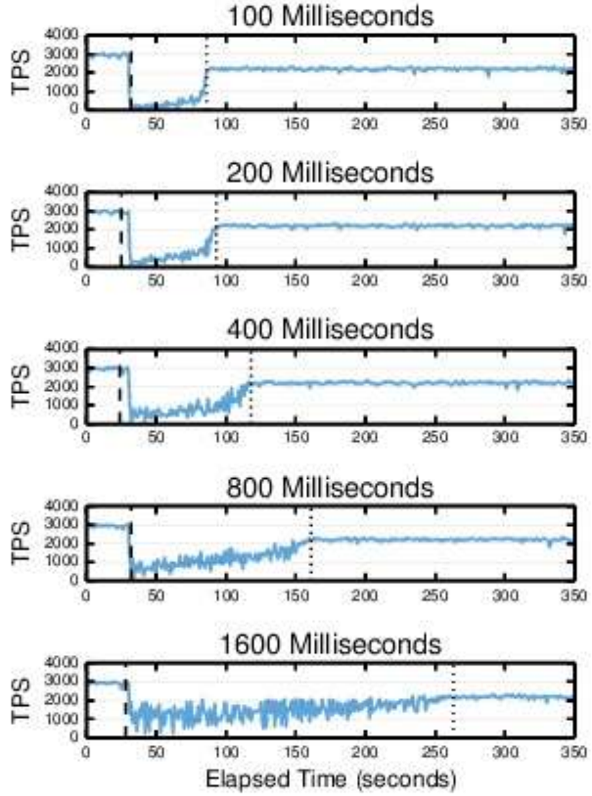


Space Async Pulls



Introduce delay at destination between new async pull requests.

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

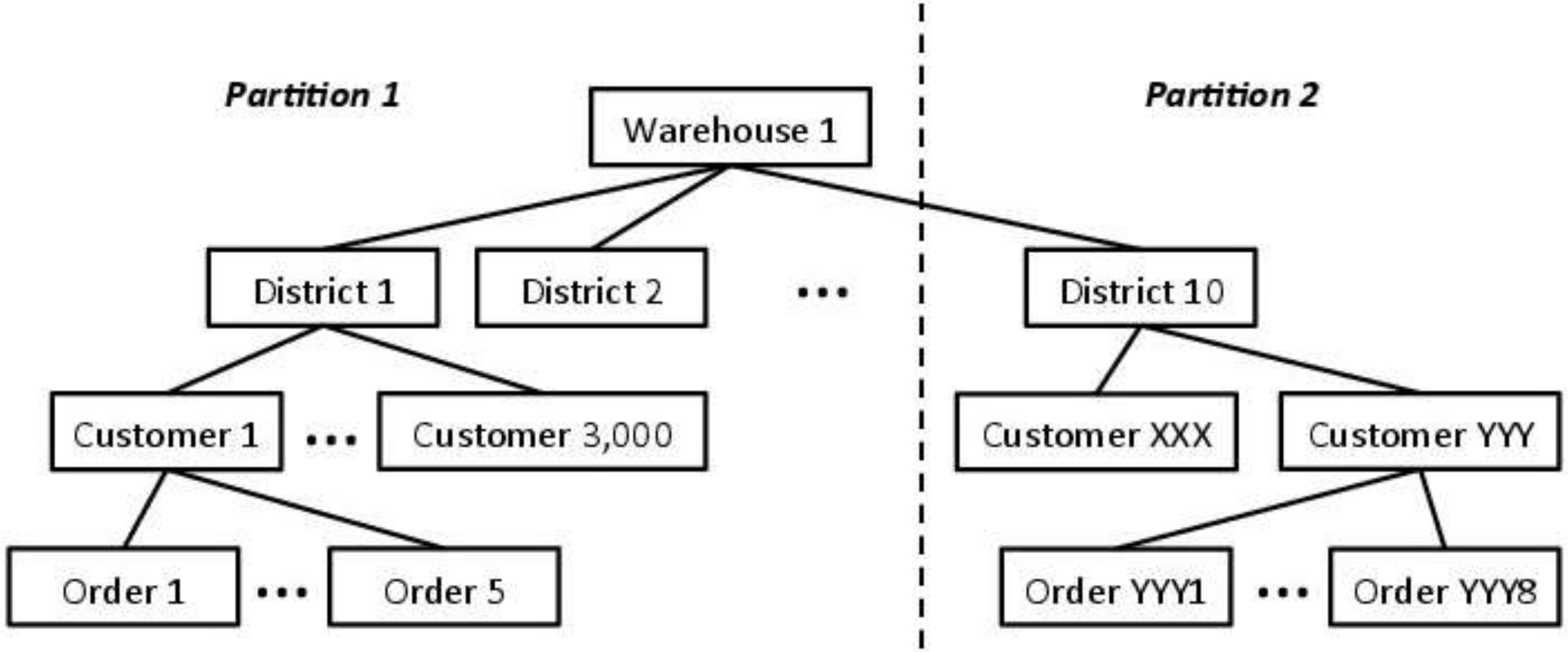


Splitting Reconfigurations

Split by pairs of source and destination

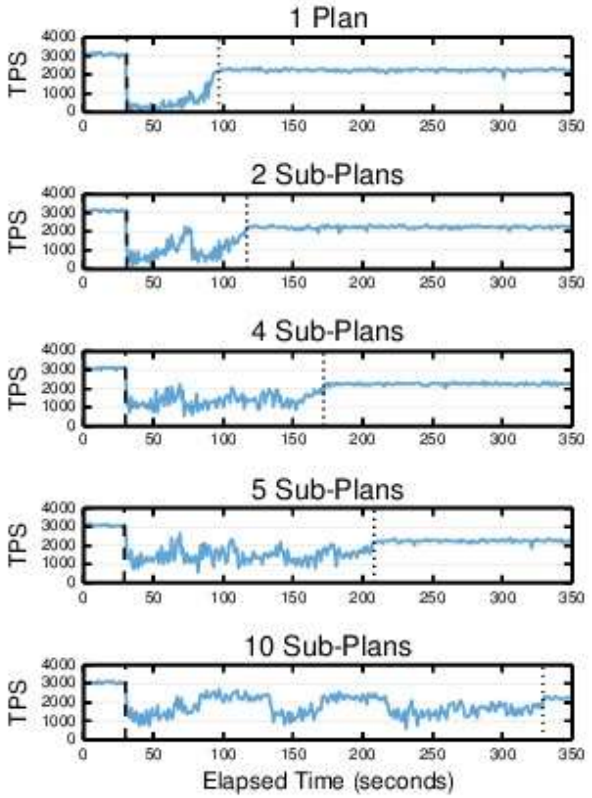
Example: partition 1 is migrating W_ID 2,3 to partitions 3 and 7, execute as two reconfigurations.

If migrating large objects, split them and use distributed transactions.



Splitting into Sub-Plans

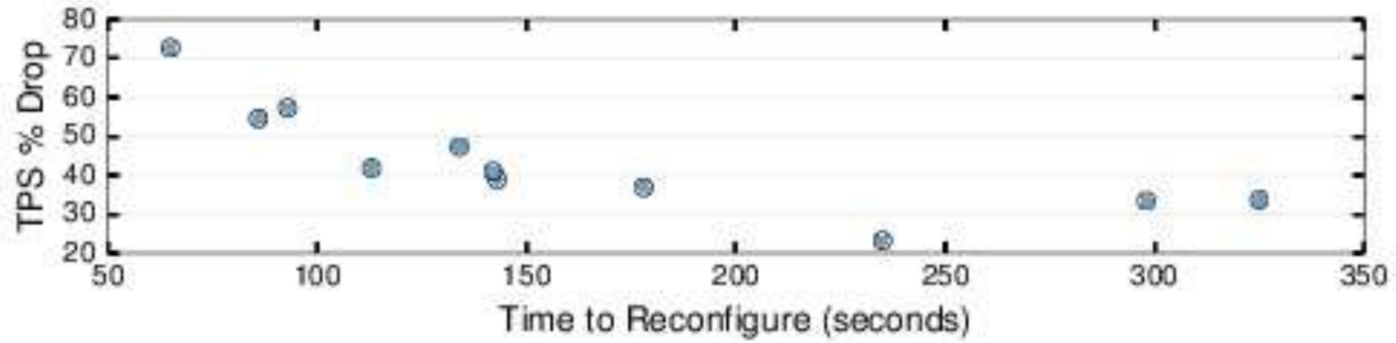
Set a cap on sub-plan splits, and split on pairs and ability to decompose migrating objects



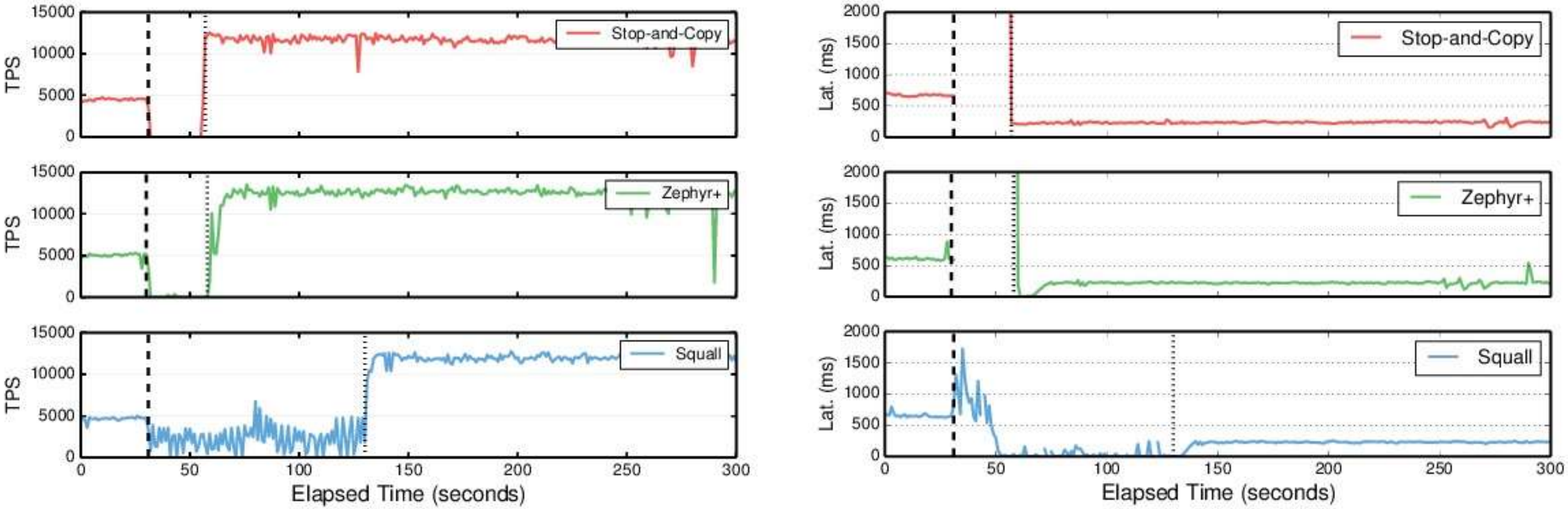
All about trade-offs

Trading off time to complete migration and performance degradation.

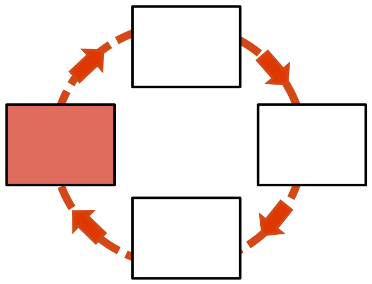
Future work to consider automating this trade-off based on service level objectives.



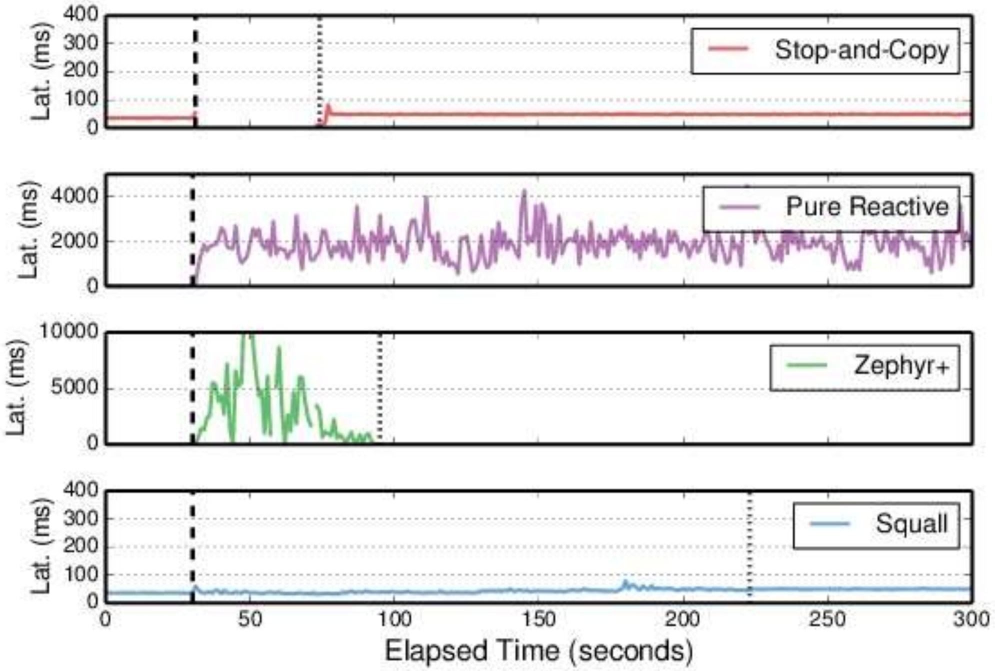
Results Highlight



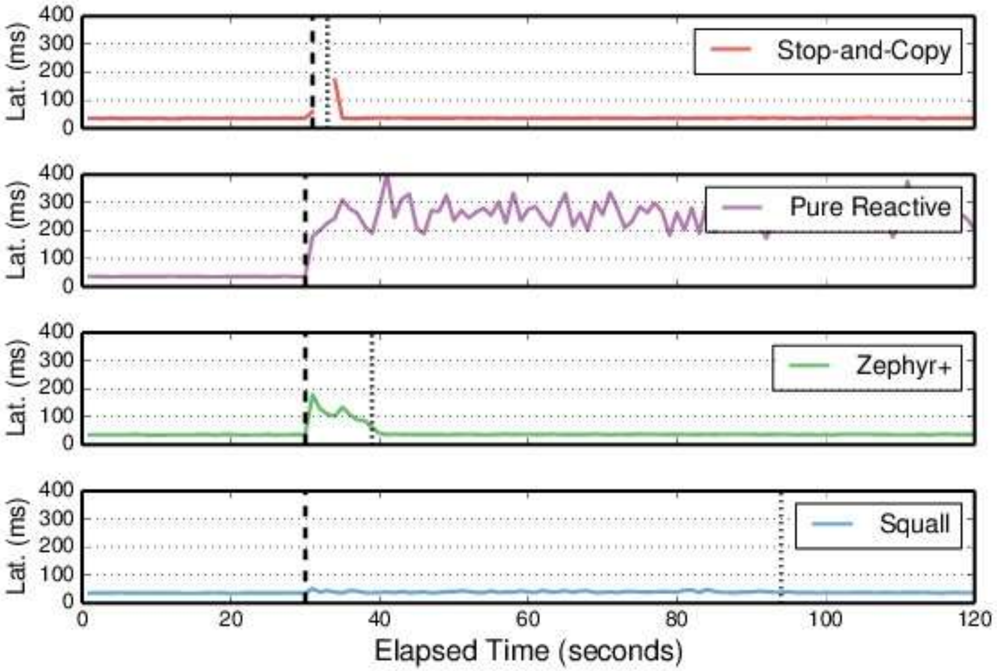
TPC-C load balancing hotspot warehouses



YCSB Latency



YCSB cluster consolidation 4 to 3 nodes



YCSB data shuffle 10% pairwise

E-Store++

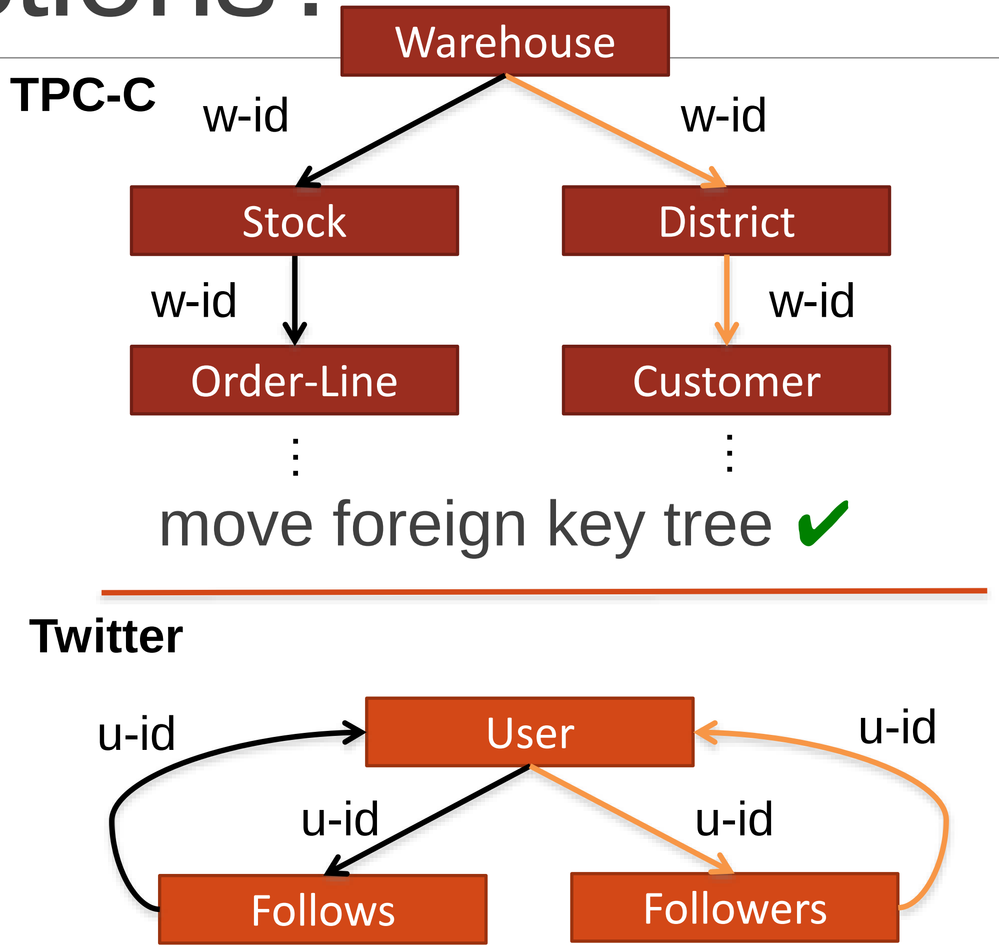
Multi-tuple transactions?

Works for **TPC-C**

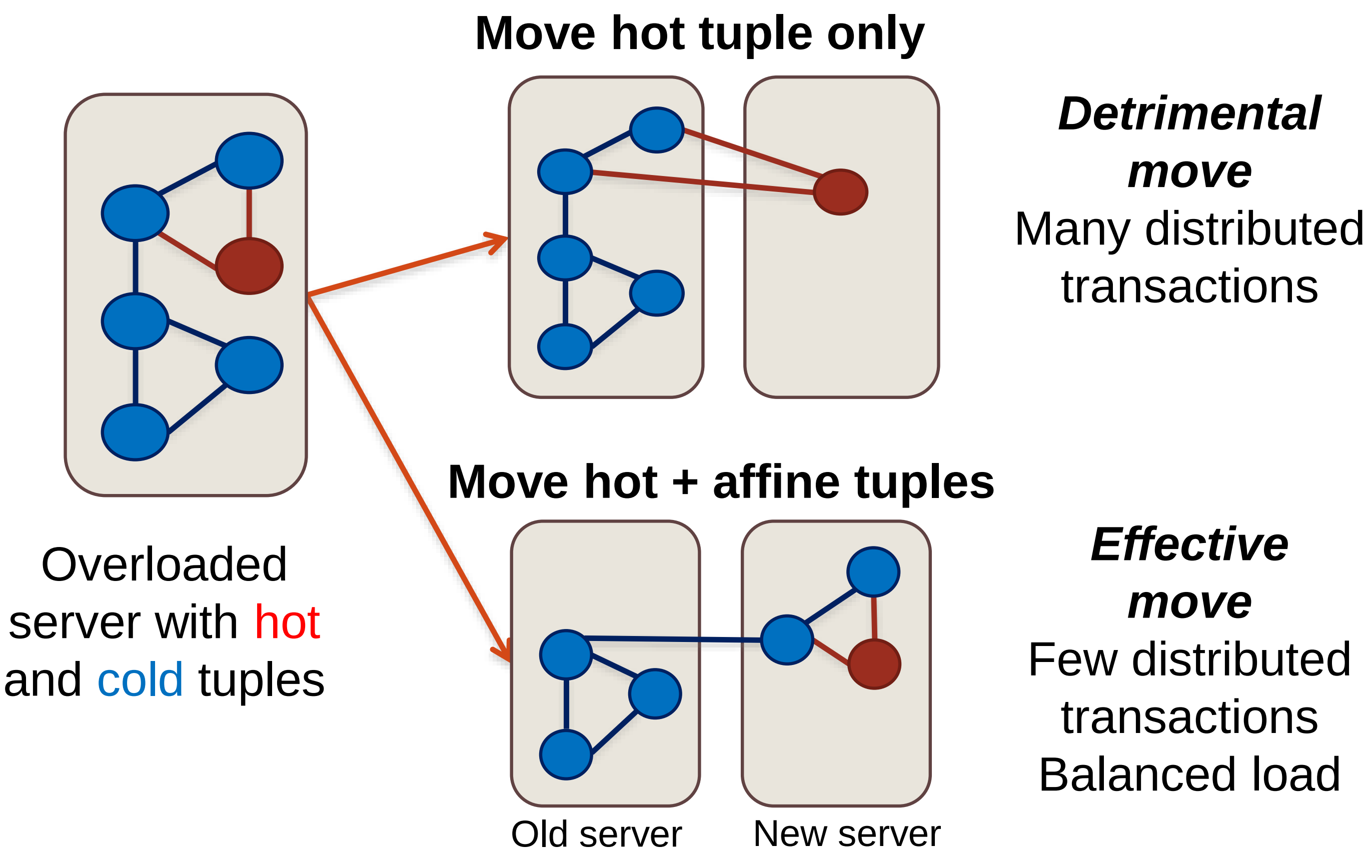
- Foreign keys form a **tree**
- Most accesses within tree
- Likelihood of cross-tree accesses is uniform

What if there is no tree?

- E.g. Twitter benchmark



Next: Consider Co-accesses



Challenges

More detailed online monitoring

- Log co-accesses among tuples

Online two-tiered graph partitioning

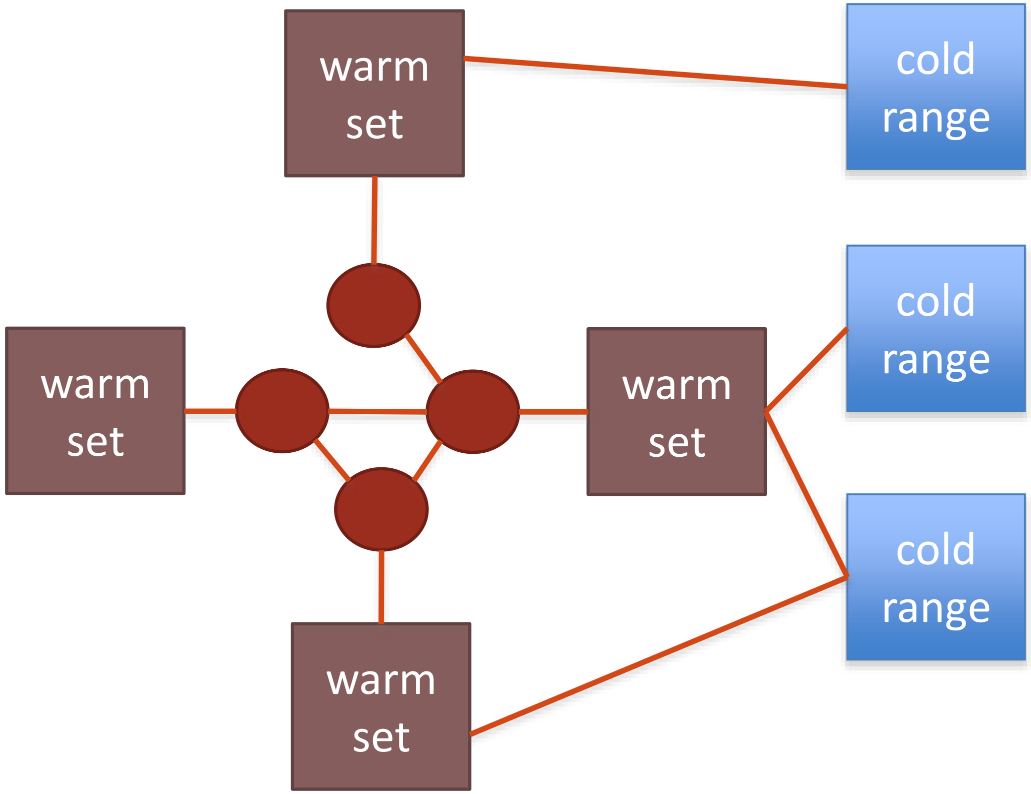
- Quickly load large access graph in memory
- Real-time partitioning
- Two tiered principle for scalability
  - hot tuples = fine granularity – cold tuples = coarse granularity

Two-Tiered Access Graph

Graph

- Hot tuples
- Warm sets: 1-hop cold tuples
- Cold ranges: rest

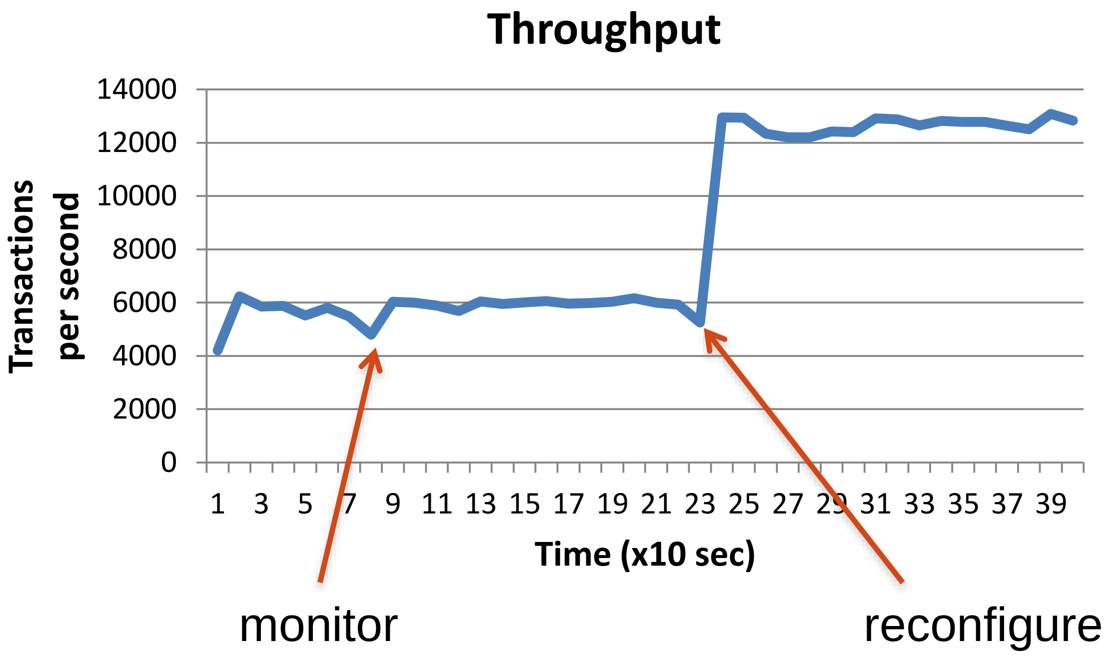
Smaller than 1-tiered



Initial Results

Many-to-many database (producer – parts – supplier)

2x throughput improvement



I Fell Asleep… What Happened

Skew happens, two-tiered partitioning for greedy load-balancing and elastic growth helps

If you have work or migrate, be careful to break up the migrations and don't be too needy on any one partition.

We are thinking hard about skewed workloads that aren't trivial to partition.

Questions?