E-Store: Fine-Grained Elastic Partitioning for Distributed Transaction Processing Systems

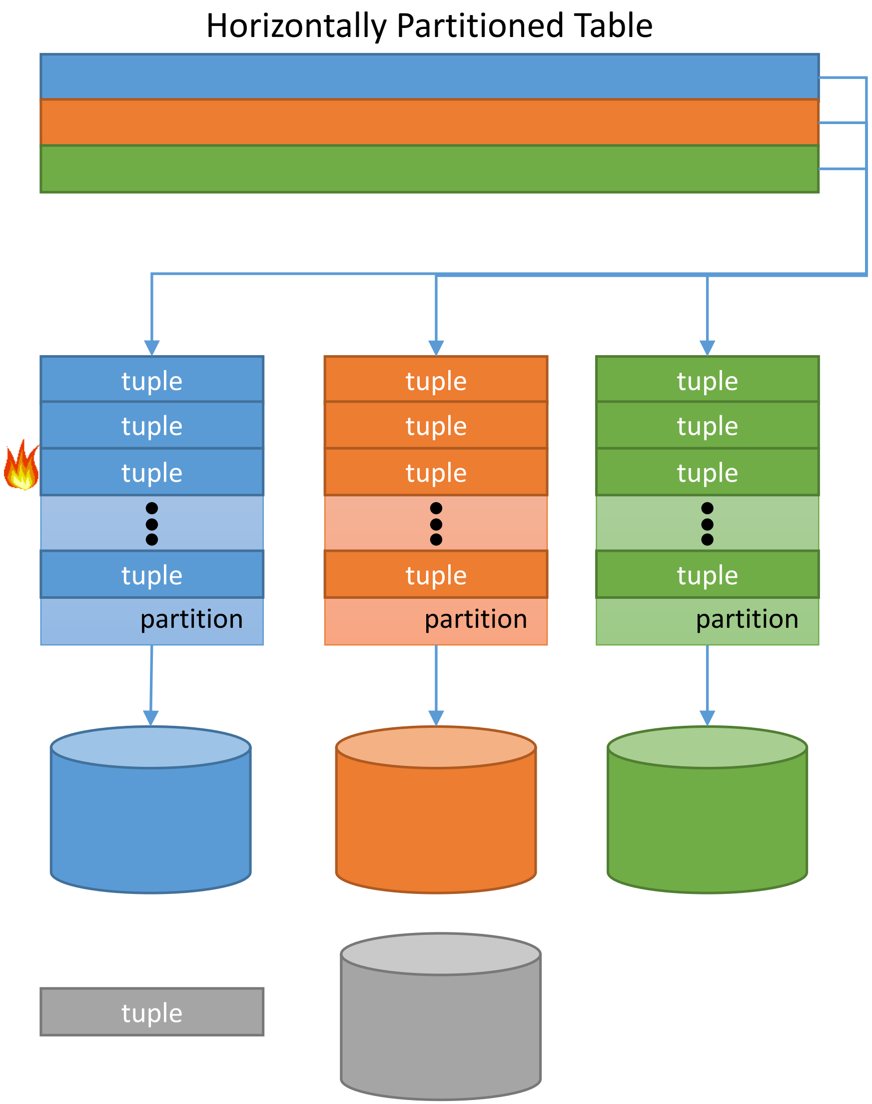
Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andrew Pavlo, Michael Stonebraker

Contributions of the Paper

- E-Store
  - Monitoring, Planning and Reconfiguration System for H-Store
  - Low-Overhead Monitoring System for OLTP Hotspots
  - Planning Engine
  - Migrates Hot Tuples on Demand
- Identifies critical design parameters for such a system
  - Monitoring System & Implementation
    - Time window of monitoring
    - How many tuples to migrate
  - Placement algorithms
    - Optimal vs. Approximate

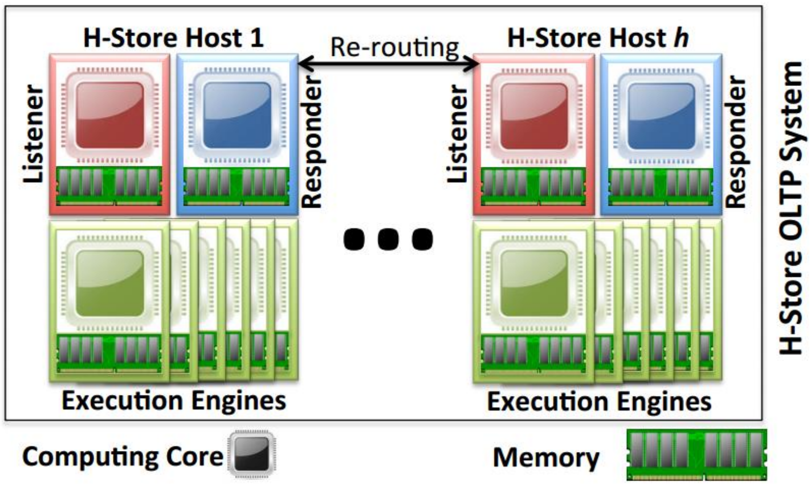
Motivation

- Skewed OLTP Workloads
  - Hot Spots
  - Time-Varying Skew
  - Load Spikes
  - Hockey Stick Effect
- Existing re-balancers work at partition-level
- Dynamic Monitoring and Movement of Hot Tuples



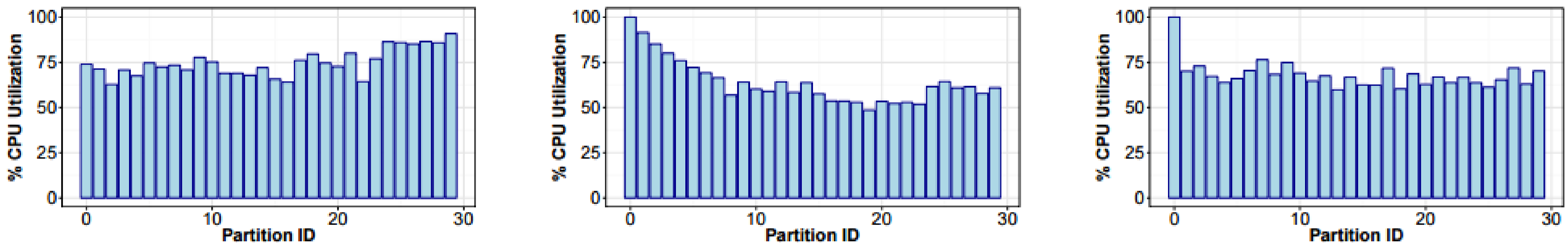
Background

- H-Store
  - In Memory DB
- DB Partitions are assigned an execution engines
  - Runs on each core
  - Optimized for stored procs
- Transaction refers to stored procs in this paper



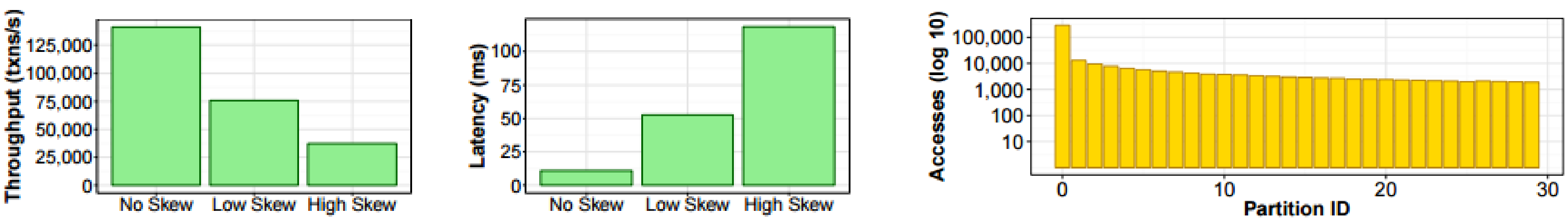
Study on Effect of Skew

- YCSB: 60M tuples, 1KB each, 30 partitions on 5 nodes
- No Skew (uniform) ●Low Skew (zipf) ●High Skew (zipf 40% + hotspots 60%)



(a) No Skew (b) Low Skew (c) High Skew

Figure 2: Partition CPU utilization for the YCSB workload with varying amounts of skew. The database is split across five nodes, each with six partitions.

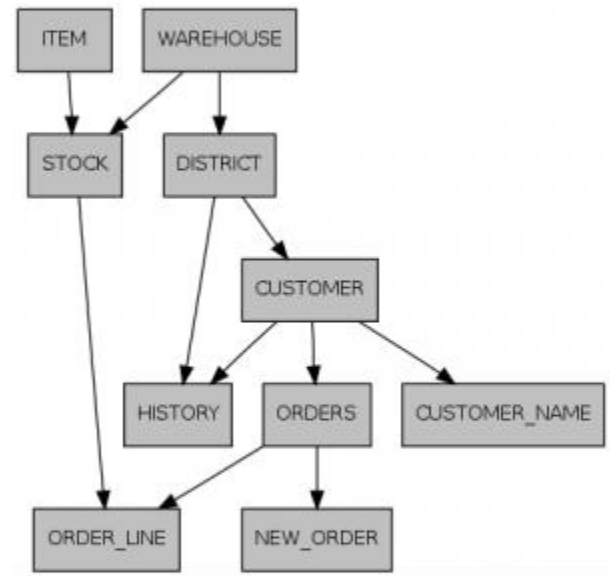


(a) Throughput (b) Latency (c) Tuple Accesses per Partition with High Skew

Figure 3: Latency and throughput measurements for different YCSB workloads with varying amounts of skew. In Fig. 3c, we show the total tuple accesses per partition over a 10 second window for the **high skew** workload.

Transactions

- Assumes that DB is in a tree-schema linked by FKs
- Co-location tuple allocation strategy
  - Partition root tuples and co-locate descendants



E-Store Architecture

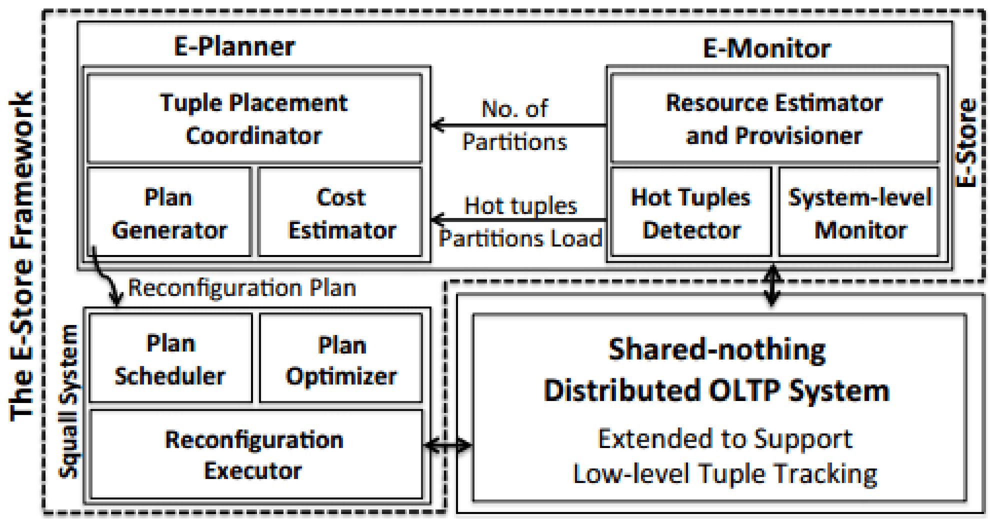


Figure 4: The E-Store Architecture.

- E-Monitor
  - Find Hot Tuples
- E-Planner
  - Find arrangement for Hot Tuples
- Squall
  - Migrate Hot Tuples

Data Migration

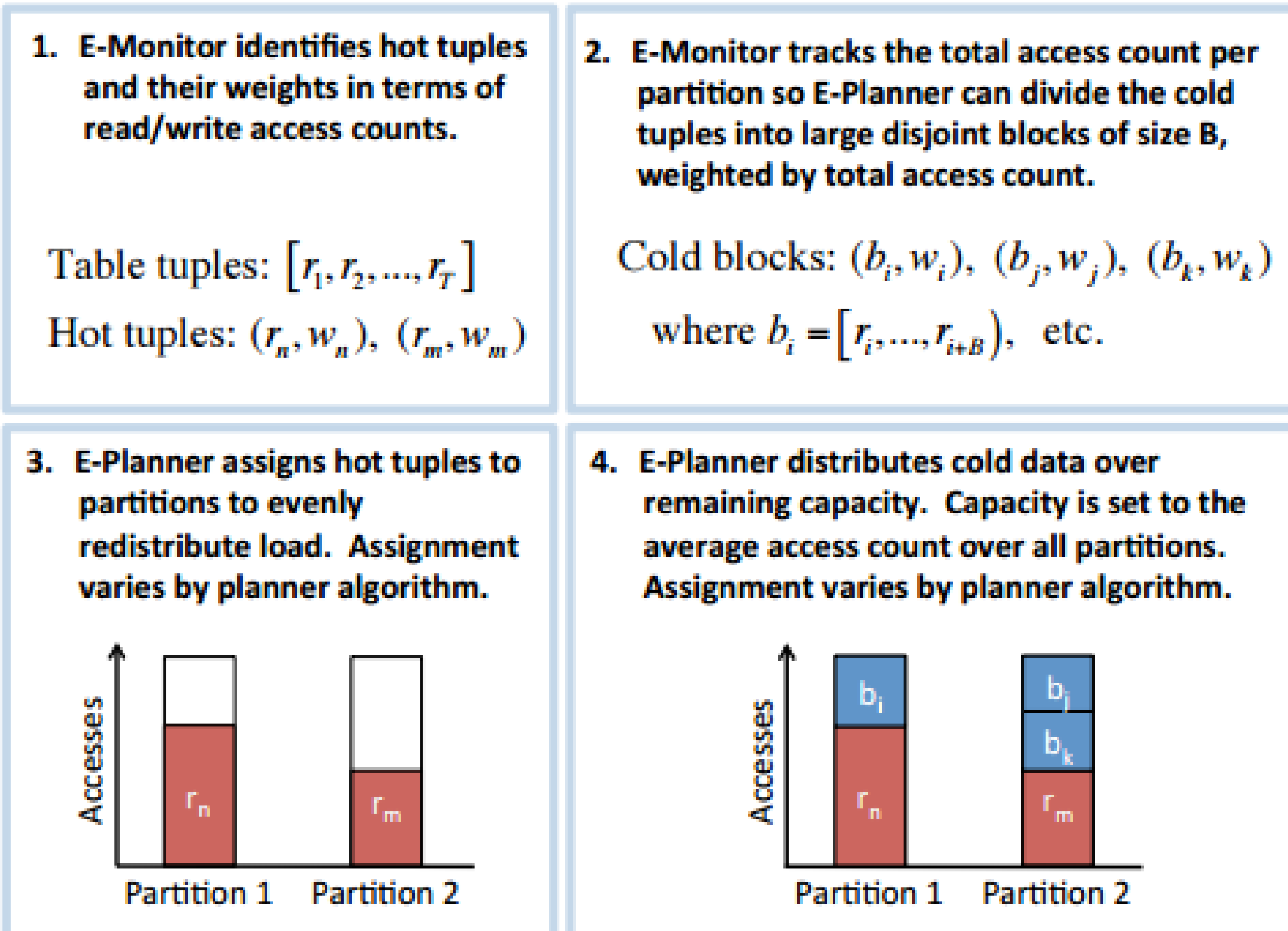


Figure 5: The steps of E-Store's migration process.

Two-Tiered Partitioning

- Single Level
  - Hash/Range partitioning on a Set of Keys
  - **Disadvantage**: Cannot handle hot tuples at fine granularity
- Two-Level
  - First Level: Root Level keys partitioned into B-size blocks
  - Voter & YCSB: B = 100,000; TPC-C: B = 1
  - Consider k top tuples at the second level; k = 1%
  - **Advantage**: Hot tuples and Cold Ranges are considered.

Adaptive Partitioning Monitoring

- Two Level Monitoring
- 1: Collecting System Level Metrics
  - CPU Utilization moving average over 60 seconds.
- 2: Tuple Level Metrics
  - Engaged when there is a significant change in level 1
  - Node selects top-k tuples in a partition
  - List sent to E-Monitor for each time window, W
  - E-Monitor assembles global top-k list of hot tuples.
  - DBA should tune time window based on transaction rate and access pattern distribution.

Re-provisioning: Optimal Placement

- Generate new partitioning scheme when hot tuple list changes
  - Select hot tuples and promote to individual placement
  - Select cold tuples and demote to block allocation scheme
- Scaling currently done 1 node at a time
- Memory not considered in placement
  - future work

Bin Packing

- **Two Tier Bin Packing**
  - Place tuples and blocks such that transmission overhead is minimized:

$$\sum_{i=1}^{n}\sum_{j=1}^{c}(x_{i,j} \times t_{i,j}) + \sum_{k=1}^{d}\sum_{j=1}^{c}(y_{k,j} \times t_{k,j} \times B)$$

  - Given Constraints:

$$\sum_{j=1}^{c} x_{i,j} = 1 \qquad \sum_{j=1}^{c} y_{k,j} = 1 \qquad L(p_j) = \sum_{i=1}^{n}(x_{i,j} \times L(r_i)) + \sum_{k=1}^{d}(y_{k,j} \times L(b_k)) \geq A - \varepsilon$$

- **Single-Tier:**
  - Only arrange blocks, not tuples

Re-provisioning: Approximate Placement

- Greedy
  - Assign tuples to nodes via locally optimal choices
  - Select hottest tuple and assign to least loaded machine
- Greedy Extended
  - Execute greedy and then balance cold blocks if cluster is still overloaded
- First-Fit
  - Assign hottest tuples in numeric order to individual nodes until they are at capacity
  - Assign cold blocks in reverse order

Evaluation - Setup

- 10 linux nodes
  - Intel Xeon Quad Core @ 2.67 Ghz
  - 32 GB RAM
- 10 Gbps switch
- H-Store
  - Command Logging
  - Transaction Commits written out to 7200 RPM HDD

Evaluation - Benchmarks

- Voter
  - Phone-based election app
- YCSB
  - No/Low/High Skew setup
- TPC-C
  - No Skew
  - Low Skew: Zipf access distribution
  - High Skew: 40% zipf, 60% to three warehouses on P0

Parameter Sensitivity Analysis

- Performance Impact of Monitoring:

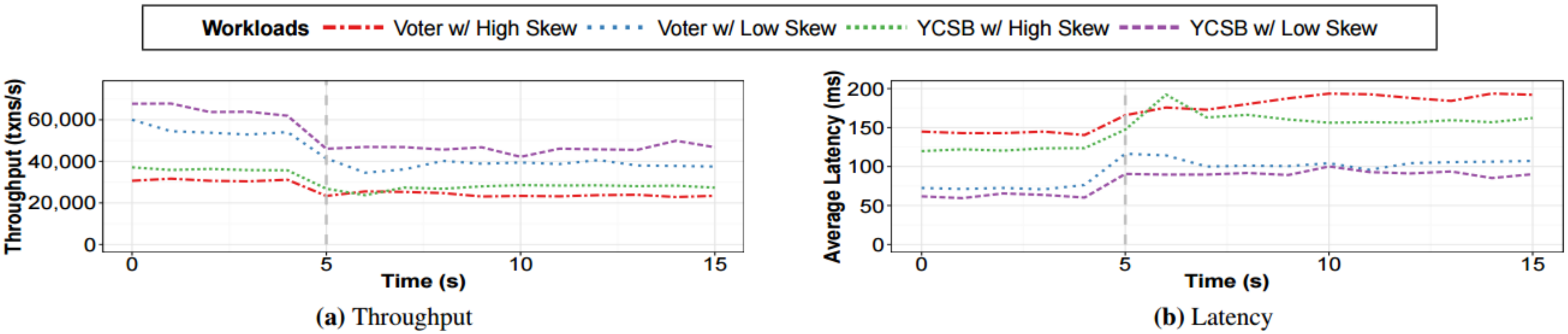
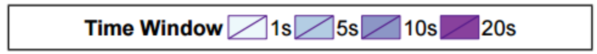


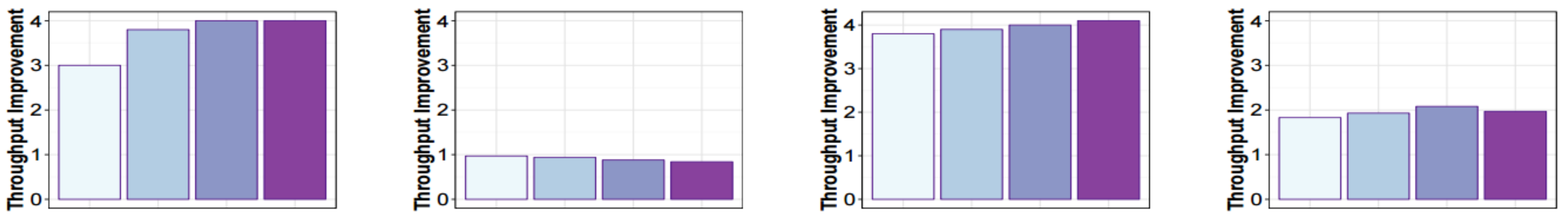
Figure 6: The impact of tuple-level monitoring on throughput and latency. Dashed lines at 5 seconds indicate the start of tuple-level monitoring.

- Throughput Hit
  - ~33% for low-skew, ~25% for high-skew
- Latency Increase
  - 45% for low-skew, 28% for high-skew

Parameter Sensitivity Analysis

- Time Window, W

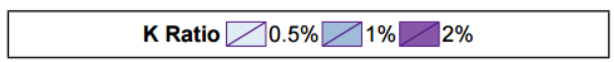


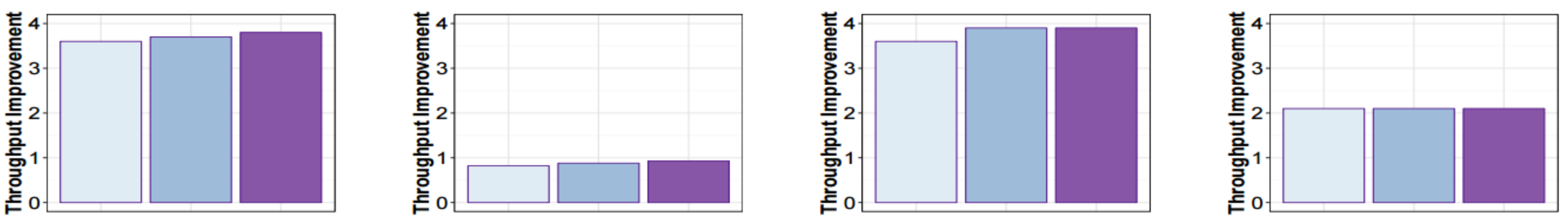


(a) Greedy High Skew (b) Greedy Low Skew (c) Greedy Extended High Skew (d) Greedy Extended Low Skew

Figure 7: Throughput improvement ratio for YCSB after reconfiguration with Greedy and Greedy Extended planners with different time windows.

- Top-k ratio:





(a) Greedy High Skew (b) Greedy Low Skew (c) Greedy Extended High Skew (d) Greedy Extended Low Skew

Figure 8: Throughput improvement ratio for YCSB after reconfiguration with Greedy and Greedy Extended planners with different top-k ratios.

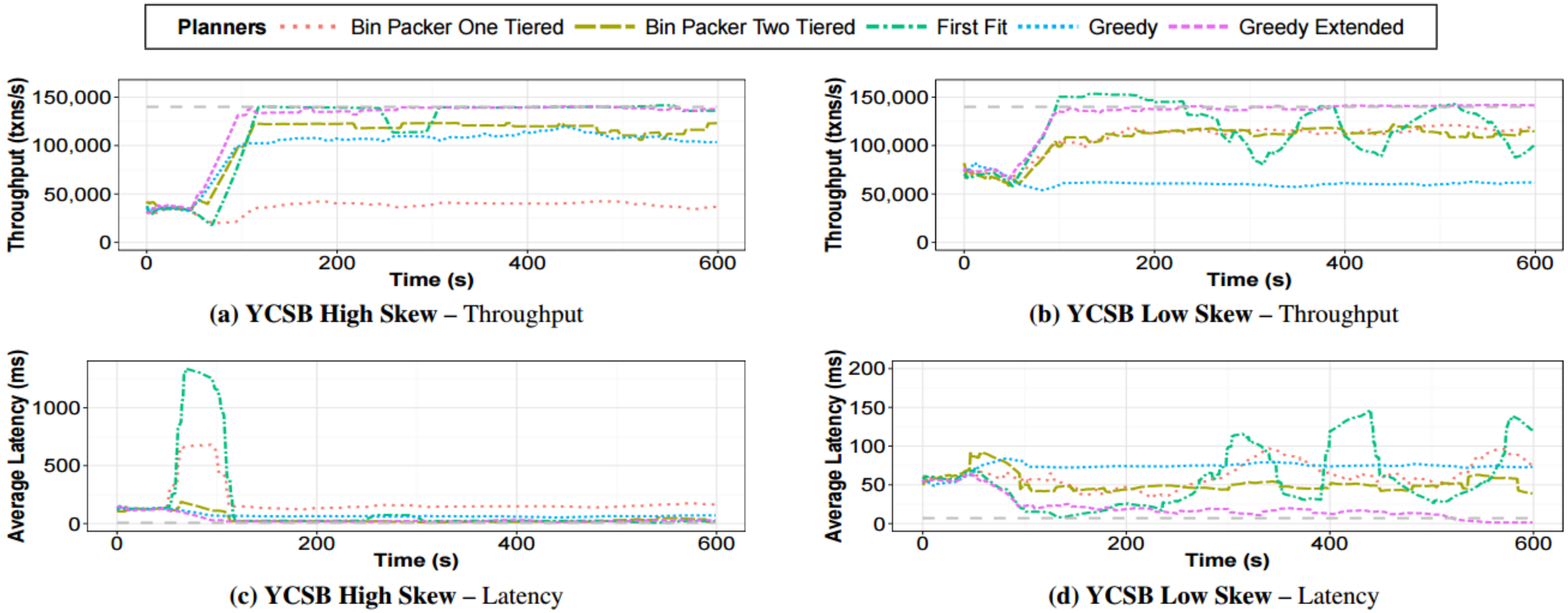
- Selected Parameters: W = 10 sec; k = 1%

Planning Execution Time

Planner	Low skew	High skew
One-tier bin packer	> 20 hrs	> 20 hrs
Two-tier bin packer	> 20 hrs	> 20 hrs
Greedy	835 ms	103 ms
Greedy Extended	872 ms	88 ms
First Fit	861 ms	104 ms

Table 1: Execution time of all planner algorithms on YCSB.

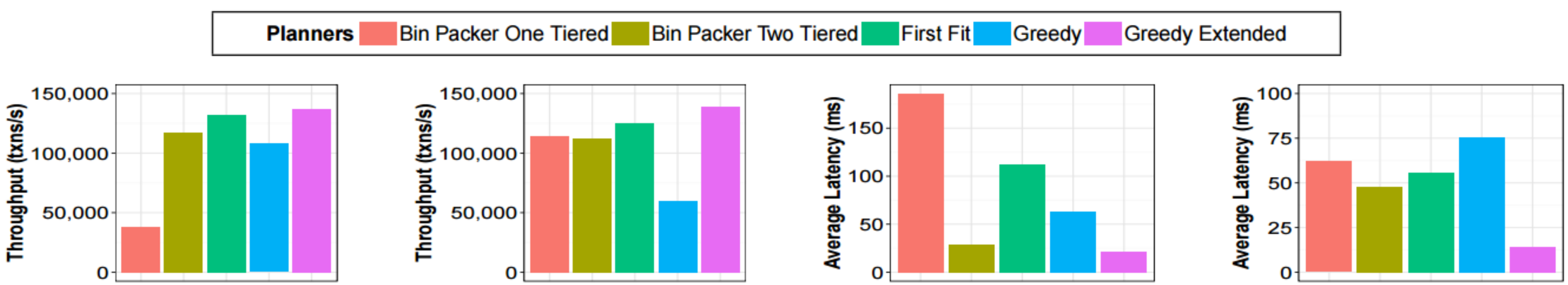
Placement Algorithm - YCSB



(a) **YCSB High Skew** – Throughput (b) **YCSB Low Skew** – Throughput

(c) **YCSB High Skew** – Latency (d) **YCSB Low Skew** – Latency

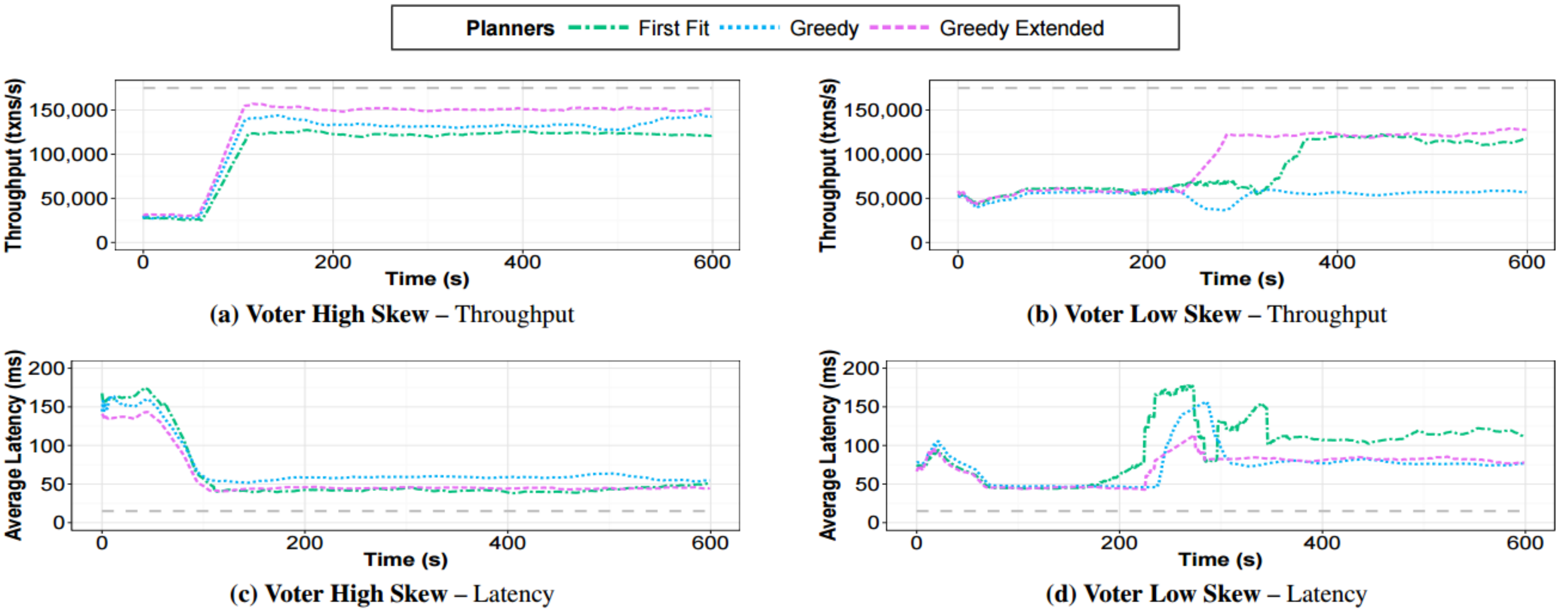
Figure 9: Comparison of all our tuple placement methods with different types of skew on YCSB.



(a) **YCSB High Skew** – Throughput (b) **YCSB Low Skew** – Throughput (c) **YCSB High Skew** – Latency (d) **YCSB Low Skew** – Latency

Figure 10: YCSB throughput and latency from Fig. 9 averaged from the start of reconfiguration at 30 seconds to the end of the run.

Placement Algorithm - Voter



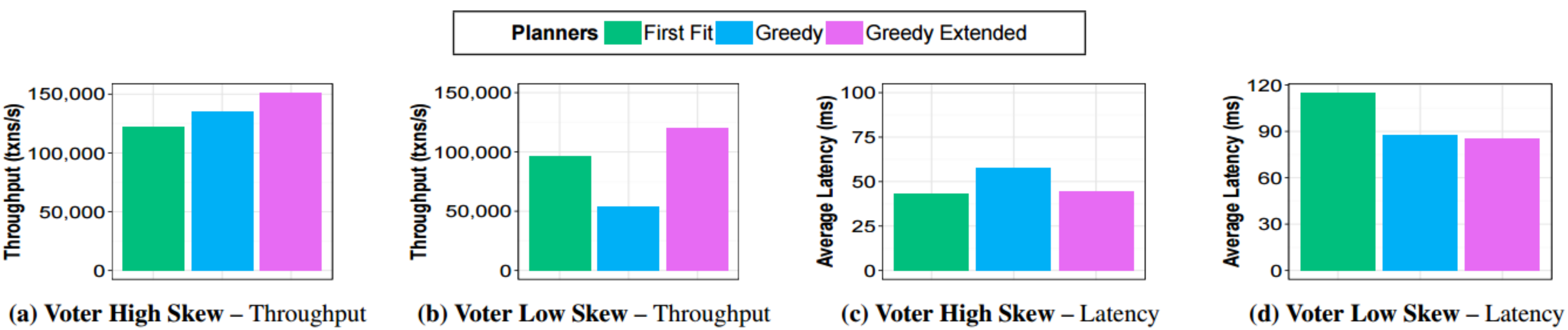
(a) **Voter High Skew** – Throughput

(b) **Voter Low Skew** – Throughput

(c) **Voter High Skew** – Latency

(d) **Voter Low Skew** – Latency

Figure 11: Comparison of approximate tuple placement methods with different types of skew on Voter.



(a) **Voter High Skew** – Throughput

(b) **Voter Low Skew** – Throughput

(c) **Voter High Skew** – Latency

(d) **Voter Low Skew** – Latency

Figure 12: Voter throughput and latency from Fig. 11, averaged from the start of reconfiguration at 30 seconds to the end of the run.

Greedy Placement with TPC-C

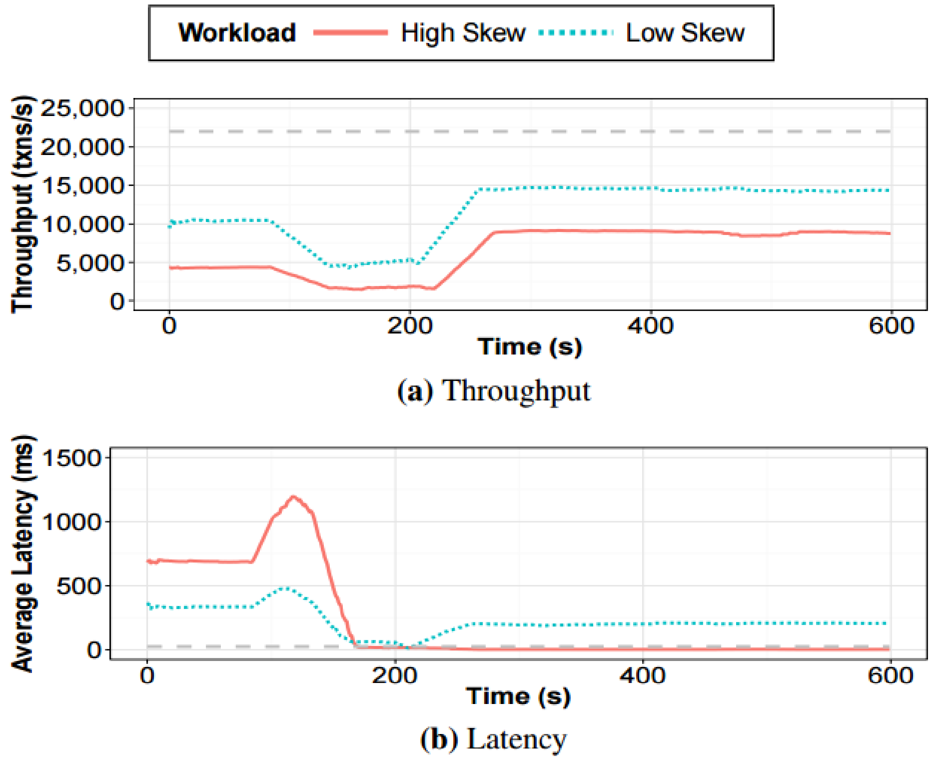
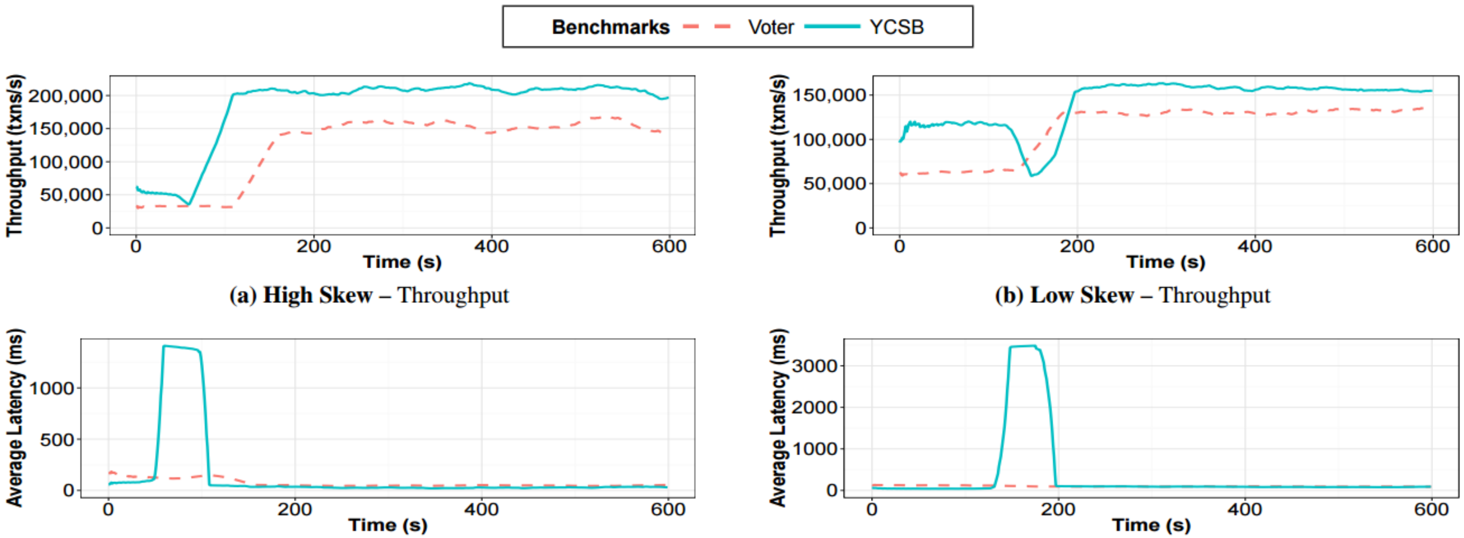


Figure 13: The Greedy planner with different types of skew on a TPC-C workload. The dashed gray line indicates system performance with no skew (a uniform load distribution).

Greedy Extended Planner – Scale Out



(a) **High Skew** – Throughput

(b) **Low Skew** – Throughput

(c) **High Skew** – Latency

(d) **Low Skew** – Latency

Figure 14: The Greedy Extended planner with different types of skew on Voter and YCSB workloads. In these experiments we overloaded the system, causing it to scale out from 5 to 6 nodes.

Greedy Extended Planner – Scale In

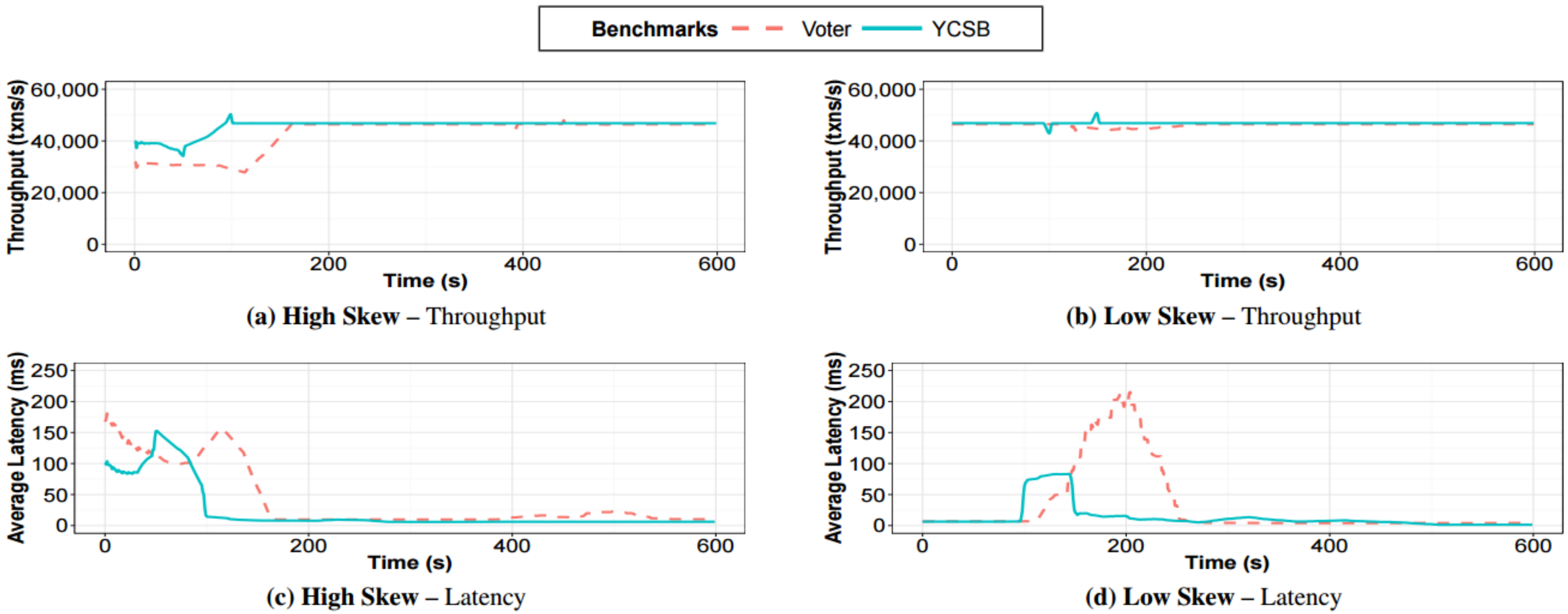


Figure 15: The Greedy Extended planner with different types of skew on Voter and YCSB workloads. In these experiments we underloaded the system, causing it to scale in from 5 to 4 nodes.

Conclusions

- Working Hot Tuple Monitoring and Migration on top of H-Store
- Can migrate tuples within 10 seconds of detecting skew
- ~4x throughput increase and ~10x latency reduction
- Future Work
  - Support Multi-partition Transactions
  - Further reduction of Monitoring Overheads
  - Planning Algorithms also use Memory as a Constraint