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%)

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.

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

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

1. E-Monitor identifies hot tuples and their weights in terms of read/write access counts.
   - Table tuples: \([r_1, r_2, \ldots, r_T]\)
   - Hot tuples: \((r_n, w_n), (r_m, w_m)\)

2. E-Monitor tracks the total access count per partition so E-Planner can divide the cold tuples into large disjoint blocks of size \(B\), weighted by total access count.
   - Cold blocks: \((b_i, w_i), (b_j, w_j), (b_k, w_k)\)
   - Where \(b_i = [r_i, \ldots, r_{i+B}]\), etc.

3. E-Planner assigns hot tuples to partitions to evenly redistribute load. Assignment varies by planner algorithm.

4. E-Planner distributes cold data over remaining capacity. Capacity is set to the average access count over all partitions. Assignment varies by planner algorithm.

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

• **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:

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

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

• Time Window, $W$

![Graphs showing throughput improvement ratio for YCSB after reconfiguration with Greedy and Greedy Extended planners with different time windows.](Figure 7)

• Top-$k$ ratio:

![Graphs showing throughput improvement ratio for YCSB after reconfiguration with Greedy and Greedy Extended planners with different top-$k$ ratios.](Figure 8)

• Selected Parameters: $W = 10 \text{ sec}; k = 1\%$
### Planning Execution Time

<table>
<thead>
<tr>
<th>Planner</th>
<th>Low skew</th>
<th>High skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-tier bin packer</td>
<td>&gt; 20 hrs</td>
<td>&gt; 20 hrs</td>
</tr>
<tr>
<td>Two-tier bin packer</td>
<td>&gt; 20 hrs</td>
<td>&gt; 20 hrs</td>
</tr>
<tr>
<td>Greedy</td>
<td>835 ms</td>
<td>103 ms</td>
</tr>
<tr>
<td>Greedy Extended</td>
<td>872 ms</td>
<td>88 ms</td>
</tr>
<tr>
<td>First Fit</td>
<td>861 ms</td>
<td>104 ms</td>
</tr>
</tbody>
</table>

*Table 1: Execution time of all planner algorithms on YCSB.*
Placement Algorithm - YCSB

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

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

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

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

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