MASSTREE
A fast key-value database designed for SMP machine
MOTIVATIONS

- Storage server performance matters
- Many systems use a single storage server
- That server is often the performance bottleneck
- Faster servers may reduce costs
- May also reduce load imbalance due to multiple servers
CONTRIBUTIONS

- In-memory concurrent tree
- Supports keys with shared prefixes efficiently
- Techniques for laying out the data that reduces DRAM time
- Demonstration that a single tree can be better for some workloads
- Complete design that addresses all bottlenecks
DESIGN
SYSTEM INTERFACE

Four different operations:

- $get_c(k)$
- $put_c(k,v)$
- $remove(k)$
- $getrange_c(k,n)$
Masstree structure: layers of B+-trees form a trie (prefix tree)

Layer 0
indexed by key bytes 0–7

Layer 1
indexed by key bytes 8–15

interior nodes
border nodes values
INVARINTS

- Keys shorter than $8h+8$ bytes are stored at layer $\leq h$.
- Any keys stored in the same layer $h$ tree have the same $8h$-byte prefix.
- When two keys share a prefix, they are stored at least as deep as the shared prefix.
ADVANTAGES OF THIS STRUCTURE

In large common prefixes

- B+-tree: $O(l \cdot \log(n))$
- Masstree: $O(l + \log(n))$
OPERATIONS
An insert in a full border node causes a split
Removing a key justs deletes the value from a node
An update changes the value of a key
CONCURRENCY
VERSION INFORMATION

Bit: 0 1 2 3 4 5 6 13 31

Locked inserting splitting isroot isborder unused

vinset vsplit
READERS

• Read version of the nodes
• Read data
• Read version again
• Retry if the version changed or was "dirty"

A dirty version is a version that is splitting or inserting
UPDATE

Using aligned write instructions
INSERT

- Lock node
- Load and rearrange the permutation
- Write new key and value
- Write the new permutation
- Unlock node
NEW LAYER

- Create new layer and border node
- Mark value as UNSTABLE
- Change pointer
- Mark value as LAYER
SPLIT

- The node is marked as splitting
- Childres are shifted from the old node to the new one
- The parent is marked as inserting
- The new node is inserted into the parent
- All the nodes involved are unlocked and their version is incremented
REMOVE

- Lock node
- Change permutation
- Unlock node

The node is deleted after a while on scheduled tasks.
PERSISTENCE

- Log every operation
- Create checkpoints after some time $t$
TREE EVALUATION

- Is Mastree the best tree to use?
- Does it really scale well with common prefixes?
- Does it scale well to multiple cores?
- Is it better than a partitioned tree?
IS MASTREE THE BEST TREE TO USE?
DOES IT REALLY SCALE WELL WITH COMMON PREFIXES?
DOES IT SCALE WELL TO MULTIPLE CORES?
IS IT BETTER THAN A PARTITIONED TREE?

The graph shows the throughput (req/sec, millions) as a function of $\delta$. The solid line represents the Masstree, and the dashed line represents the hard-partitioned Masstree.
# System Evaluation

<table>
<thead>
<tr>
<th>Workload</th>
<th>Masstree</th>
<th>MongoDB</th>
<th>VoltDB</th>
<th>Redis</th>
<th>Memcached</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Uniform key popularity, 1-to-10-byte decimal keys, one 8-byte column</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>9.10</td>
<td>0.04</td>
<td>0.22</td>
<td>5.97</td>
<td>9.78</td>
</tr>
<tr>
<td>put</td>
<td>5.84</td>
<td>0.04</td>
<td>0.22</td>
<td>2.97</td>
<td>1.21</td>
</tr>
<tr>
<td>1-core get</td>
<td>0.91</td>
<td>0.01</td>
<td>0.02</td>
<td>0.54</td>
<td>0.77</td>
</tr>
<tr>
<td>1-core put</td>
<td>0.60</td>
<td>0.04</td>
<td>0.02</td>
<td>0.28</td>
<td>0.11</td>
</tr>
<tr>
<td><strong>Zipfian key popularity, 5-to-24-byte keys, ten 4-byte columns for get, one 4-byte column for update &amp; getrange</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MYCSB-A (50% get, 50% put)</td>
<td>6.05</td>
<td>0.05</td>
<td>0.20</td>
<td>2.13</td>
<td>N/A</td>
</tr>
<tr>
<td>MYCSB-B (95% get, 5% put)</td>
<td>8.90</td>
<td>0.04</td>
<td>0.20</td>
<td>2.69</td>
<td>N/A</td>
</tr>
<tr>
<td>MYCSB-C (all get)</td>
<td>9.86</td>
<td>0.05</td>
<td>0.21</td>
<td>2.70</td>
<td>5.28</td>
</tr>
<tr>
<td>MYCSB-E (95% getrange, 5% put)</td>
<td>0.91</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
RELATED WORK
OLFIT

- Is a B+link-tree with optimistic concurrency control
- Each update to a node changes the node’s version number
Palm

- Is a lock-free concurrent B+-tree with twice the throughput of OLFIT
- Uses SIMD instructions to take advantage of parallelism within each core
- Lookups are partitioned between the cores to improve cache usage
ALPHASORT

- Stores partial keys to minimize cache misses
  Mastree uses layers to do the same thing
FUTURE WORK

- Add more features such as named columns
- Auto partition if the skewness is 0
- What happens if one core has a failure while splitting a node?
- Can this be used to create a faster distributed system?
CONCLUSIONS

- Masstree is a persistent in-memory key-value database
- It offers high concurrency
- Does not have problems with long prefixes
- Executes more than six million simple queries per second (16-core machine)