Wormhole: A Fast Ordered Index for In-memory Data Management(I)

Main Paper: Wormhole: A Fast Ordered Index for In-memory Data Management

Authors: Wu, Xingbo, Fan Ni, and Song Jiang.

Published in: In Proceedings of the Fourteenth EuroSys Conference

Published Year: 2019

Publisher: ACM

Presented by:
Pooja Ravi
1001578517
INTRODUCTION

- Wormhole is a new index data structure for sorted keys with an asymptotically low cost (O(log L)) (L is the key length).
- It leverages the advantages of three existing data structures.
  - B+tree
  - Prefix Tree (Trie)
  - Hash Table
- The advantages of the wormhole is
  - Space Efficient (large arrays)
  - Search cost is reduced compared to other structures
  - Efficient range operations
1) Show an example B+ tree and an example prefix tree. Do both support range search? For a given number of keys, which one has a lower lookup cost?

1. B+ Tree and the Prefix tree both support Range search because their keys are in sorted order.
2. The prefix tree has lower lookup cost for a smaller key when compared to the B+ tree.
3. The lookup cost of B+ tree is $O(\log N)$ key comparisons, where $N$ is the number of keys in the Index.
4. The lookup cost of the Prefix tree is $O(L)$ where $L$ is the length of the key.
2) Please design a table to compare B+-tree, prefix tree, and hash table on their lookup cost, support of range search, and space efficiency.

<table>
<thead>
<tr>
<th></th>
<th>Lookup cost</th>
<th>Range search</th>
<th>Space efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>B+ tree</td>
<td>High lookup cost with a large N(Number of keys). O(log N)</td>
<td>Allows Range search</td>
<td>Space efficient (long arrays)</td>
</tr>
<tr>
<td>Prefix tree</td>
<td>High lookup cost even with a moderate L(Length of the key) O(L)</td>
<td>Allows Range search</td>
<td>Space inefficiency</td>
</tr>
<tr>
<td>Hash table</td>
<td>O(1)</td>
<td>Unable to perform range operations</td>
<td>Space inefficiency</td>
</tr>
</tbody>
</table>
3) If we replace B+ tree’s MetaTree with a hash table, what are the issues? Can we have a B+ tree AND additionally a hash table to accelerate lookup at MetaTree?

- The hash table space cost is more than the MetaTree.
- The new key cannot be inserted at the correct position in the sorted LeafList.
- It does not support range search.

- The additional hash table along with B+ tree will improve the look up cost when compared to having B+ tree alone.
- When the key is hashed, and the value is non-existent, then the pointer will point to the root of the B+ tree which performs regular search in B+ tree.
- Issues like space inefficiency and Inconsistency are to be addressed.

Figure 2: Replacing B+ tree’s MetaTree with a hash table
4) With B+ tree’s MetaTree replaced by a MetaTrie, anchors are inserted into the trie. Use Fig. 3 as an example to explain how an anchor is determined? If the last key in the first leaf node is “Austi”, what’s the anchor between the first and the second leaf nodes?

Figure 3: Replacing B+ tree’s MetaTree with MetaTrie
An anchor key acts as a border between a node and the node immediately left to it.

Anchor key should meet the two conditions: (a) The Ordering Condition (b) The Prefix Condition.

Assume that, Nodeb is a new leaf node whose anchor key has not been determined.
- The smallest key in Nodeb is \(\langle P_1 P_2 \ldots P_k B_1 B_2 \ldots B_m \rangle\) and the largest key in previous node Nodea is \(\langle P_1 P_2 \ldots P_k A_1 A_2 \ldots A_n \rangle\) and \(A_1 < B_1\).
- If Nodeb is not the left-most node on the LeafList \((m > 0)\):
  - check whether \(\langle P_1 P_2 \ldots P_k B_1 \rangle\) is a prefix of the anchor key of the next node Nodec.
  - If not Nodeb’s anchor is \(\langle P_1 P_2 \ldots P_k B_1 \rangle\). Otherwise, Nodeb’s anchor is \(\langle P_1 P_2 \ldots P_k B_1 \bot \rangle\), which cannot be a prefix of Nodec’s anchor.
  - check whether Nodea’s anchor is a prefix of Nodeb’s anchor (Nodea is \(\langle P_1 P_2 \ldots P_j \rangle\), where \(j \leq k\)). If so, Nodea’s anchor will be changed to \(\langle P_1 P_2 \ldots P_j \bot \rangle\).
  - Otherwise (Nodeb is the left-most node), its anchor is \(\bot\).

If the last key in the first leaf node is “Austi”, the anchor between the first and the second leaf nodes will be “Austin”.
5) Use Figure 4 as an example to explain how search keys “A”, “Denice”, and “Julian” are found in the tree?

- The basic lookup operation on the MetaTrie with a search key takes place by matching tokens in the key to those in the trie one at a time and walk down the trie level by level accordingly. This leads the lookup to the target leaf node in the LeafList where the key is stored.

- If a search key is in the index, it must be in its target node. The target nodes of “A”, “Denice”, and “Joseph” are the first, second, and fourth leaf nodes in Figure 3, respectively.
REFERENCES


Thank you