

Designing an Index for ZooDB

Jonas Nick & Bogdan Vancea

May 29, 2014

Outline

- 1 Introduction
- 2 Goals & Challenges
- 3 The new Index Implementation
- 4 Benchmarks

ZooDB

- an open source object database written in Java
- JDO standard compliant
- 4 times faster than competitor db4o
- zoodb.org

Database Index

Key-Value data structure that allows for fast retrieval and ordered iteration of records stored in a file.

Database Index

Key-Value data structure that allows for fast retrieval and ordered iteration of records stored in a file.

Example:

```
ZooJdoHelper.createIndex(pm, Person.class, "name",  
false);
```

Database Index

Key-Value data structure that allows for fast retrieval and ordered iteration of records stored in a file.

Example:

```
ZooJdoHelper.createIndex(pm, Person.class, "name",  
false);
```

Attribute Index
Value \rightarrow Object-ID

Database Index

Key-Value data structure that allows for fast retrieval and ordered iteration of records stored in a file.

Example:

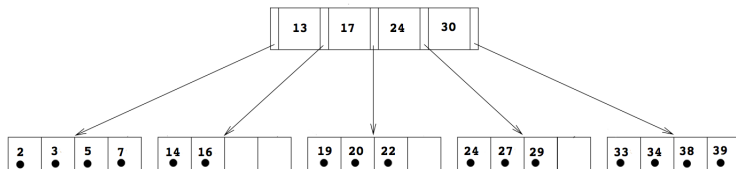
```
ZooJdoHelper.createIndex(pm, Person.class, "name",  
false);
```

Attribute Index
Value \rightarrow Object-ID

ObjectID Index
OID \rightarrow Diskpos

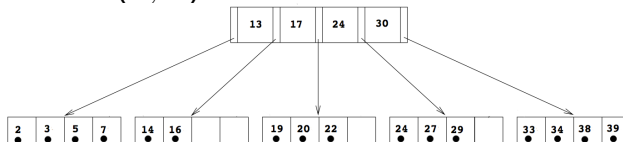
Free Space Index
Page-ID \rightarrow TxID

B+ Tree

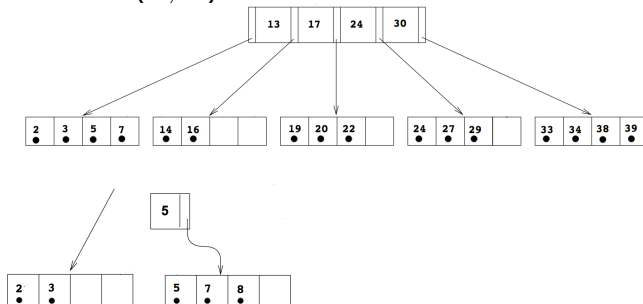


- node fills one disk page
- inner node contains keys and children pointer, leaves contain keys and values
- key unique vs. key-value unique

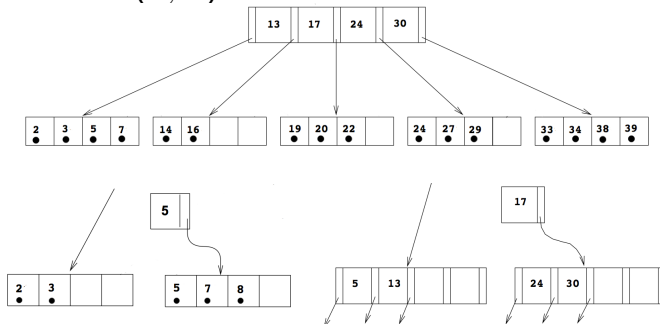
Example: insert (8, v)



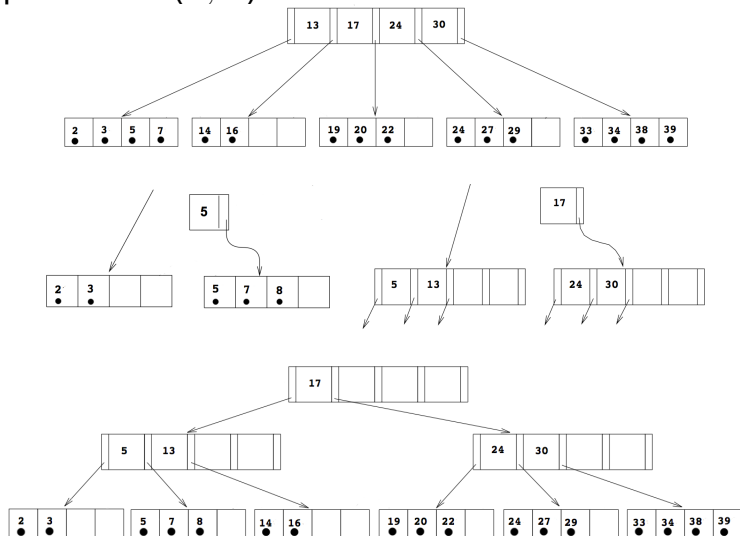
Example: insert (8, v)



Example: insert (8, v)



Example: insert (8, v)



Goals

- fast B+ tree index
- key unique and key-value unique
- buffer manager to allow caching
- prefix sharing

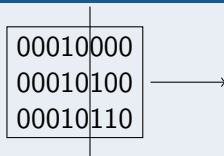
Prefix Sharing

Exploit common prefix

00010000
00010100
00010110

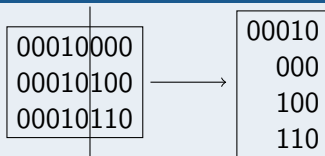
Prefix Sharing

Exploit common prefix



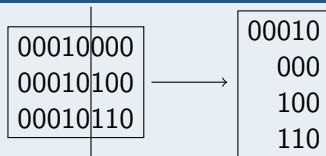
Prefix Sharing

Exploit common prefix



Prefix Sharing

Exploit common prefix

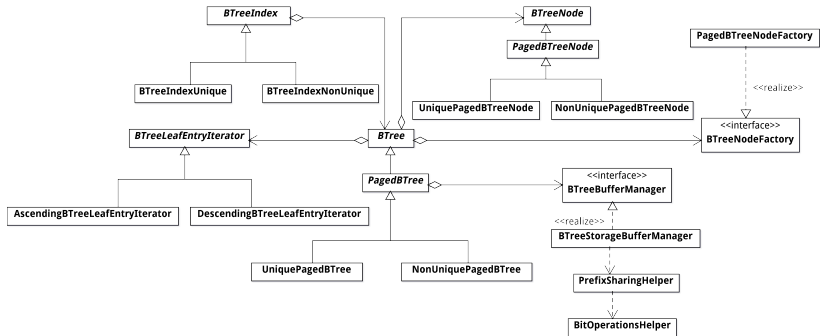


- variable number of key-value entries per page
- prefix determines
 - if 2 nodes can be merged without overflow
 - the number of entries that can be redistributed from one node to the other

Challenges

- runtime dominated by disk access
 - prefer few nodes
 - rarely modify nodes
- new features are costly
- Textbook algorithms need to be adapted
 1. not optimized for practical scenarios
 2. do not cover duplicates nor prefix sharing
- Low-level implementation optimizations

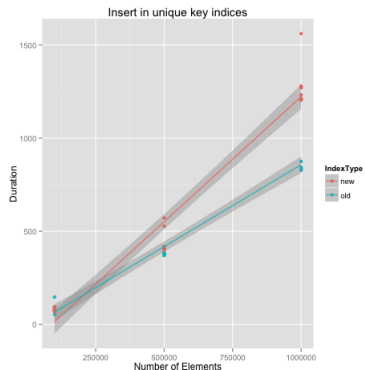
10/16



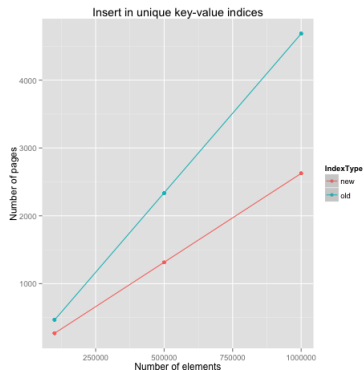
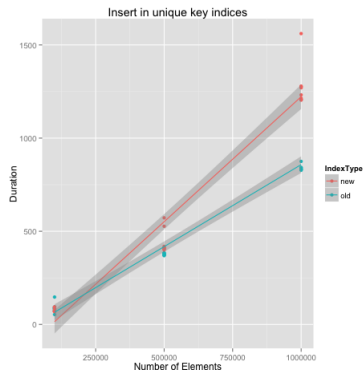
Operations

- Search - Similar to normal B+ Tree
- Insert overflow
 - attempt to redistribute values to left sibling before creating a new node
- Delete underflow
 - check if possible to merge with left or right neighbour
 - check if possible to split current node between left and right
 - redistribute from left or right
- Write
 - only write dirty nodes
 - prefix encoding

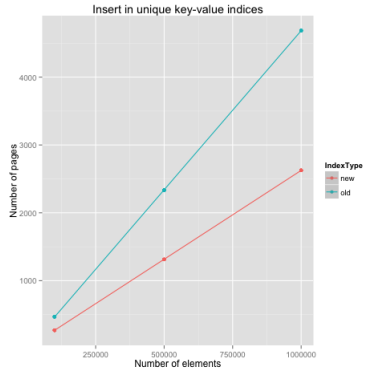
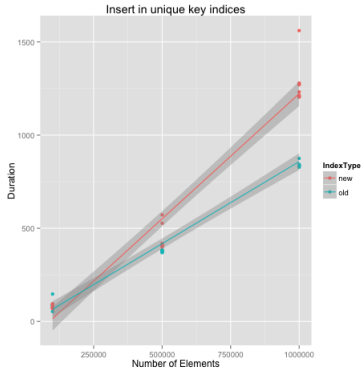
Microbenchmarks



Microbenchmarks

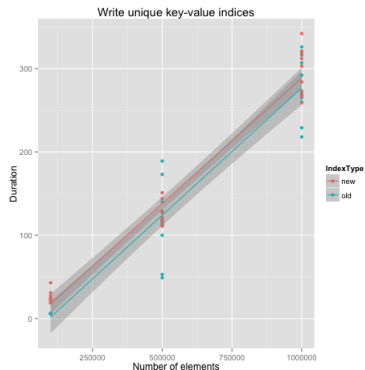


Microbenchmarks

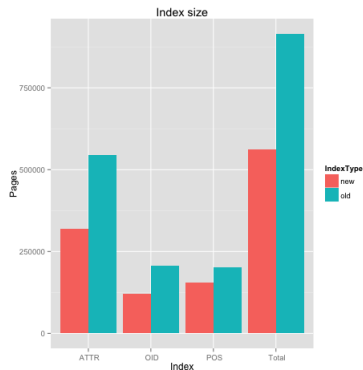


- in every microbenchmark the new index is significantly slower
- in most microbenchmarks there s a significantly lower number of nodes

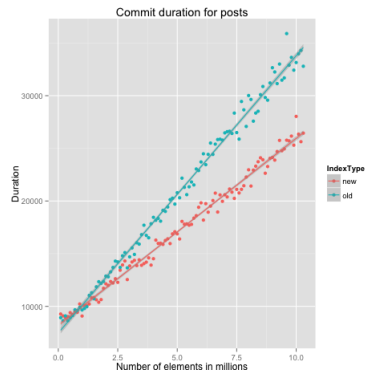
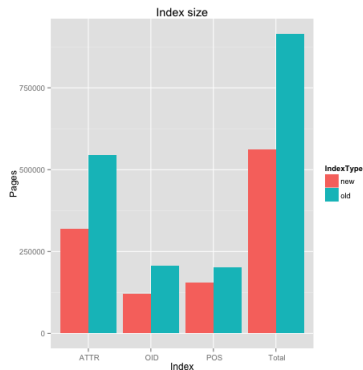
Microbenchmarks



StackOverflow Import



StackOverflow Import



Summary

- ...

Outlook

- ...