Designing an Index for ZooDB

Jonas Nick & Bogdan Vancea

May 30, 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 for fast retrieval and ordered iteration of entries stored in a file.

Database Index

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

Example:

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

Database Index

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

Example:

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

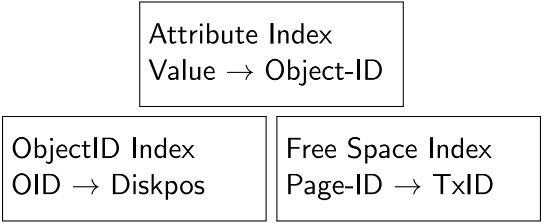
Attribute Index
Value → Object-ID

Database Index

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

Example:

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



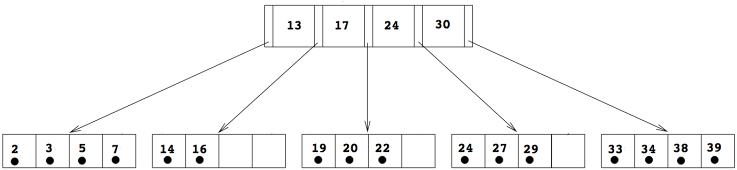
Attribute Index
Value → Object-ID

ObjectID Index
OID → Diskpos

Free Space Index
Page-ID → TxID

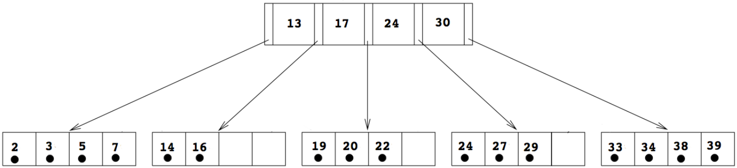
B+ Tree

B+ Tree



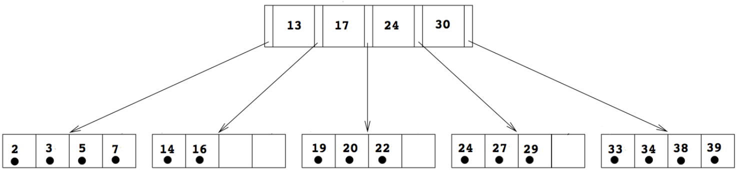
- Inner node contains keys and children pointer, leaf contain keys and values.

B+ Tree



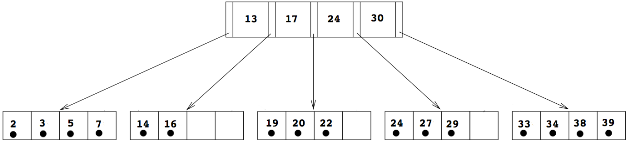
- Inner node contains keys and children pointer, leaf contain keys and values.
- **Node fills one disk page.**

B+ Tree

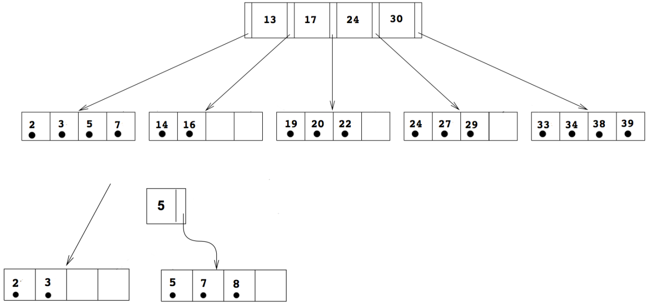


- Inner node contains keys and children pointer,
  leaf contain keys and values.
- **Node fills one disk page.**
- Node has maximum and minimum number of entries.

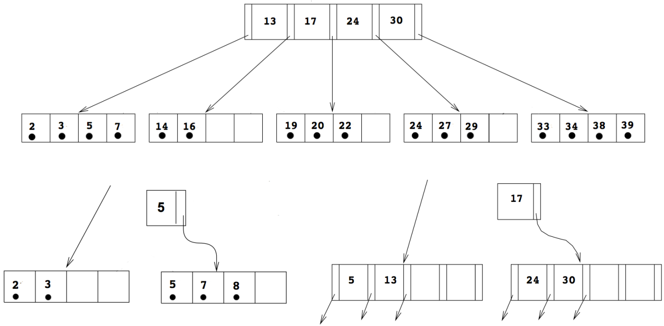
Example: insert $(8, v)$



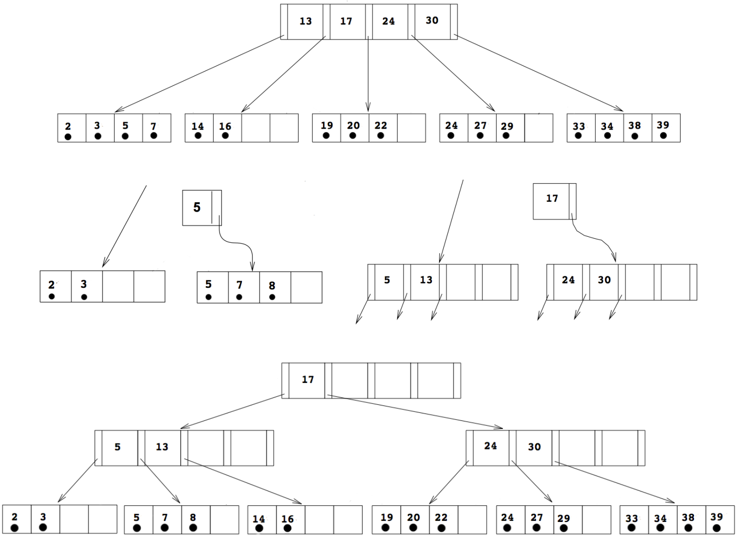
Example: insert $(8, v)$



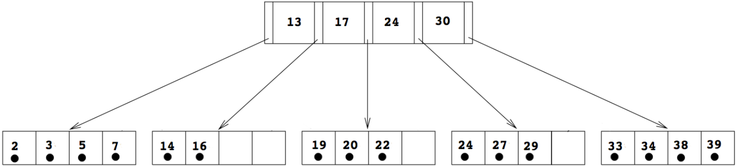
Example: insert $(8, v)$



Example: insert $(8, v)$

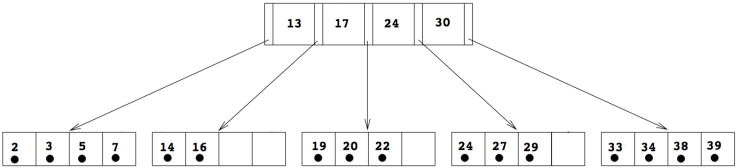


B+ Tree



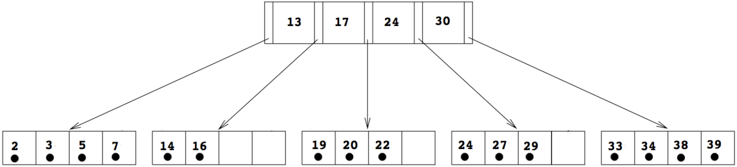
- Inner node contains keys and children pointer, leaf contain keys and values.
- **Node fills one disk page.**
- Node has maximum and minimum number of entries.

B+ Tree



- Inner node contains keys and children pointer, leaf contain keys and values.
- **Node fills one disk page.**
- Node has maximum and minimum number of entries.
- Rebalancing on insert: split, on delete: redistribute or merge

B+ Tree



- Inner node contains keys and children pointer, leaf contain keys and values.
- **Node fills one disk page.**
- Node has maximum and minimum number of entries.
- Rebalancing on insert: split, on delete: redistribute or merge
- Insert, remove, search are logarithmic.

Goals

- faster B+ tree index

Goals

- faster B+ tree index
- key unique and key-value unique
    - Ex. insert (1,1), (1,2)

Goals

- faster B+ tree index
- key unique and key-value unique
    - Ex. insert (1,1), (1,2)
- range query iterators

Goals

- faster B+ tree index
- key unique and key-value unique
    - Ex. insert (1,1), (1,2)
- range query iterators
- buffer manager to allow caching
    - fetches pages

Goals

- faster B+ tree index
- key unique and key-value unique
    - Ex. insert (1,1), (1,2)
- range query iterators
- buffer manager to allow caching
    - fetches pages
- prefix sharing

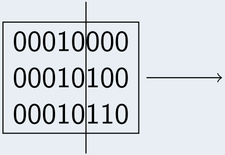
Prefix Sharing

Exploit common prefix

00010000
00010100
00010110

Prefix Sharing

Exploit common prefix



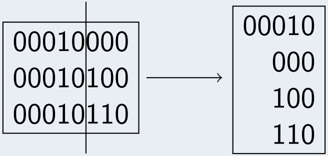
00010000

00010100

00010110

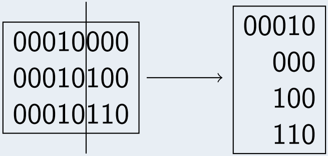
Prefix Sharing

Exploit common prefix



Prefix Sharing

Exploit common prefix



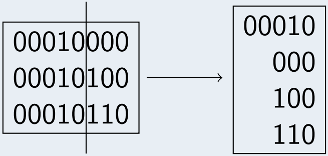
00010000
00010100
00010110

00010
000
100
110

- variable number of key-value entries per node

Prefix Sharing

Exploit common prefix



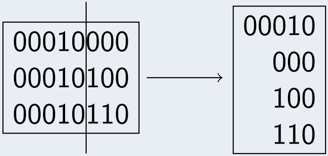
00010000
00010100
00010110

00010
000
100
110

- variable number of key-value entries per node
- prefix determines
    - if two nodes can be split without underflow

Prefix Sharing

Exploit common prefix

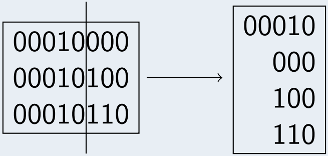

00010000
00010100
00010110

00010
000
100
110

- variable number of key-value entries per node
- prefix determines
  - if two nodes can be split without underflow
  - if two nodes can be merged without overflow

Prefix Sharing

Exploit common prefix



00010000
00010100
00010110

00010
000
100
110

- variable number of key-value entries per node
- prefix determines
    - if two nodes can be split without underflow
    - if two 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

Challenges

- runtime dominated by disk access
    - prefer fewer nodes

Challenges

- runtime dominated by disk access
    - prefer fewer nodes
    - rarely modify nodes

Challenges

- runtime dominated by disk access
  - prefer fewer nodes
  - rarely modify nodes
- New features are costly.

Challenges

- runtime dominated by disk access
    - prefer fewer nodes
    - rarely modify nodes
- New features are costly.
- Textbook algorithms need to be adapted.

Challenges

- runtime dominated by disk access
    - prefer fewer nodes
    - rarely modify nodes
- New features are costly.
- Textbook algorithms need to be adapted.
    1. not optimized for practical scenarios

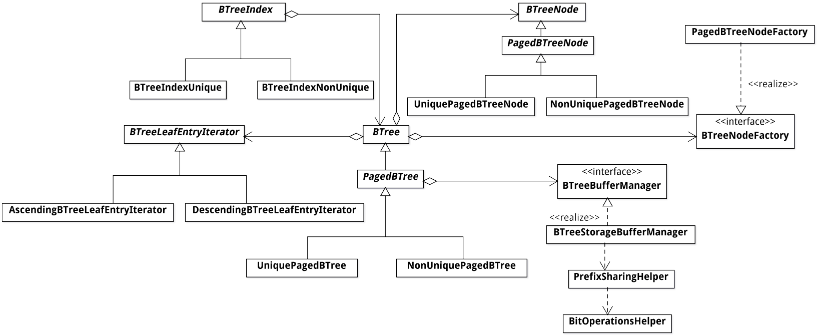
Challenges

- runtime dominated by disk access
    - prefer fewer 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

Challenges

- runtime dominated by disk access
  - prefer fewer 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

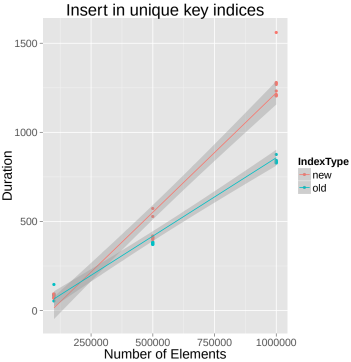
Index Implementation



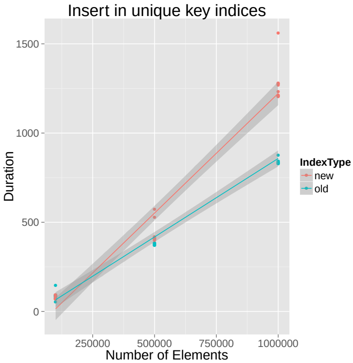
Operations

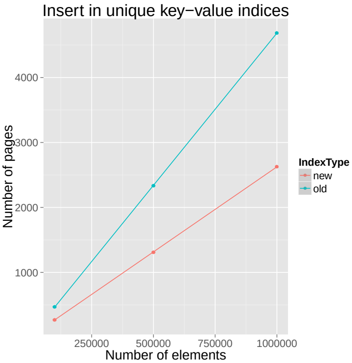
- Search - Similar to normal B+ Tree
- Insert overflow
    - attempt to redistribute values to left sibling before creating a new node
- Delete underflow
    - check if possible to merge with left or right neighbour
    - check if possible to split current node between left and right
    - redistribute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

Microbenchmarks

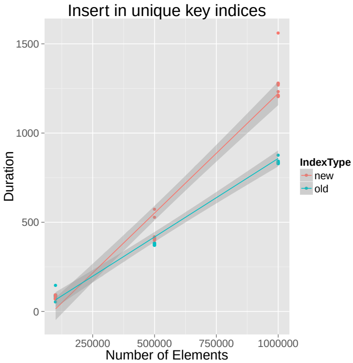


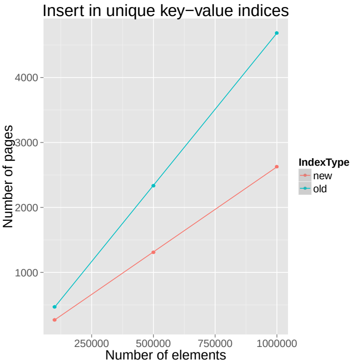
Microbenchmarks





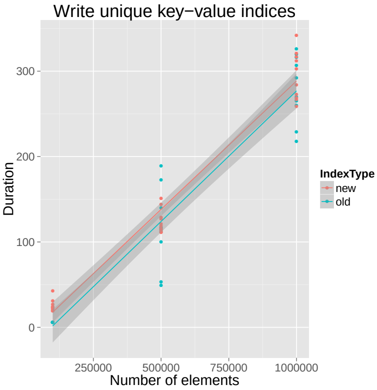
Microbenchmarks


Insert in unique key indices

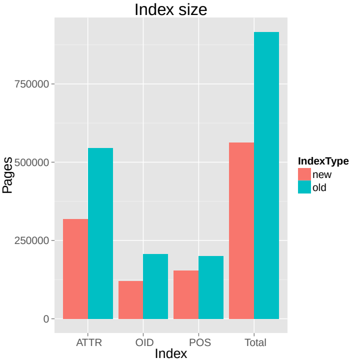

Insert in unique key−value indices

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

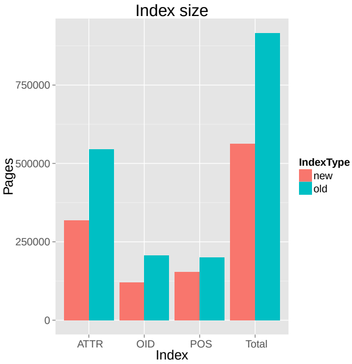
Microbenchmarks

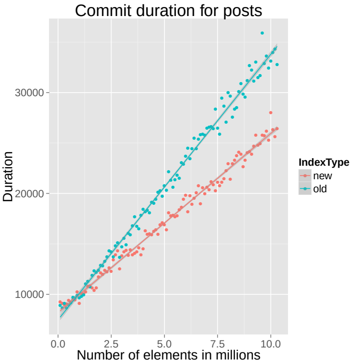


StackOverflow Import



StackOverflow Import





Summary

- ...

Outlook

- ...