Designing an Index for ZooDB

Jonas Nick & Bogdan Vancea

June 1, 2014

Outline

ZooDB
The JDO Database

- 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

1. **fast** retrieval
2. **ordered** iteration
3. stored in a **file**

Database Index

Key-Value data structure

1. **fast** retrieval
2. **ordered** iteration
3. stored in a **file**

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

Database Index

Key-Value data structure

1. **fast** retrieval
2. **ordered** iteration
3. stored in a **file**

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

Attribute Index
Value → Object-ID

Database Index

Key-Value data structure

1. **fast** retrieval
2. **ordered** iteration
3. stored in a **file**

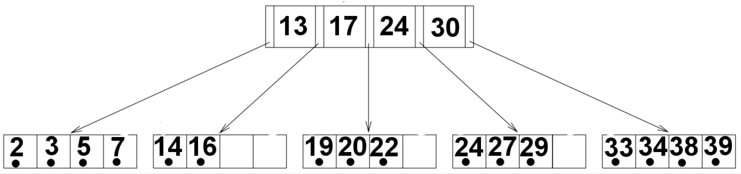
```
ZooJdoHelper.createIndex(pm, Person.class, "name",
false);
```

Attribute Index	ObjectID Index	Free Space Index
Value $\rightarrow$ Object-ID	OID $\rightarrow$ Diskpos	Page-ID $\rightarrow$ TxID

B+ Tree

Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

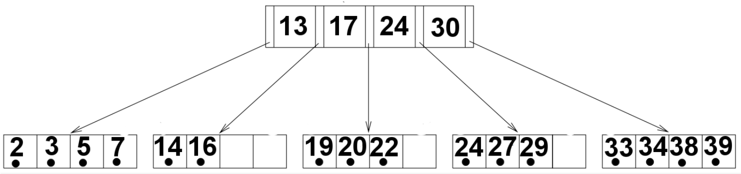
B+ Tree



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

Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

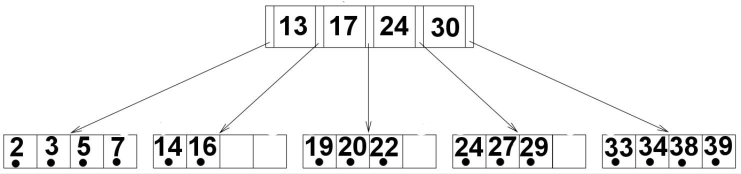
B+ Tree



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

Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

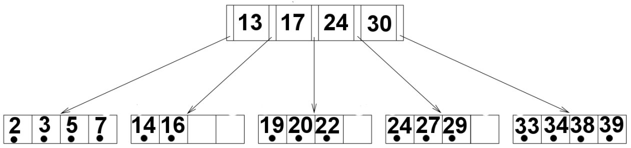
B+ Tree



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

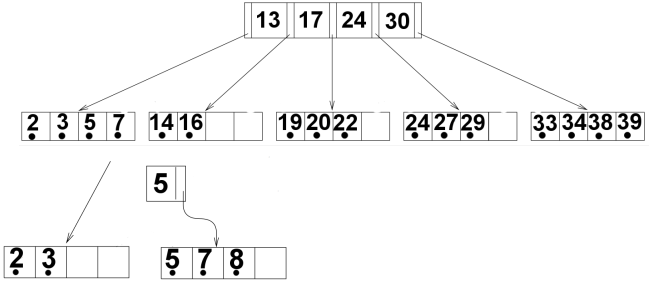
Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

Example: insert $(8, v)$



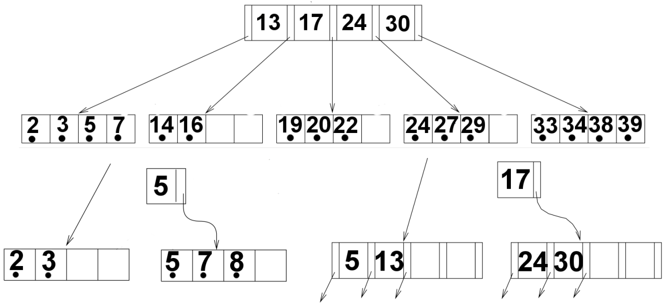
Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

Example: insert $(8, v)$



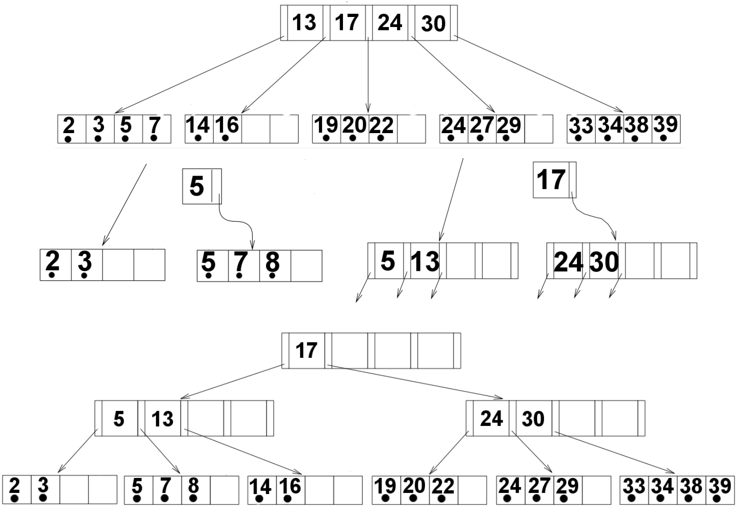
Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

Example: insert $(8, v)$



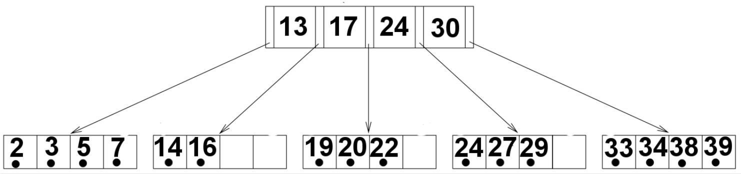
Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

Example: insert $(8, v)$



Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

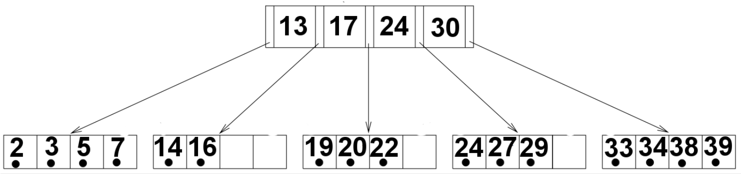
B+ Tree



- 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.

Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

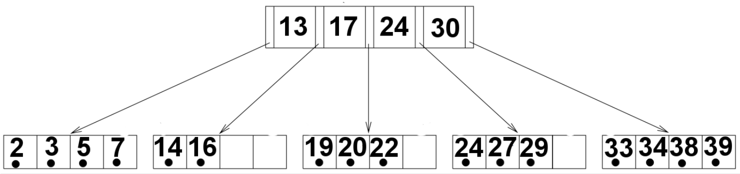
B+ Tree



- 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

Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

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.

Images adapted from Database Management Systems by Ramakrishnan and Gehrke.

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

$$\boxed{\begin{array}{c}00010000\\00010100\\00010110\end{array}}$$

Prefix Sharing

Exploit common prefix

00010|000
00010|100 ⟶
00010|110

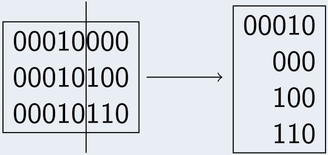
Prefix Sharing

Exploit common prefix

00010 | 000 00010 | 100 00010 | 110	→	00010 000 100 110

Prefix Sharing

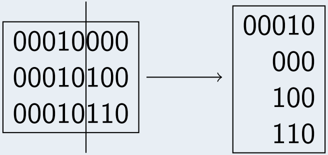
Exploit common prefix



- variable number of key-value entries per node

Prefix Sharing

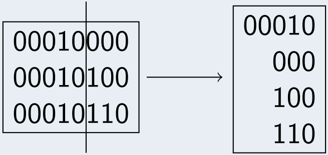
Exploit common prefix



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

Prefix Sharing

Exploit common prefix



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

Prefix Sharing

Exploit common prefix

00010	000	→	00010
00010	100		000
00010	110		100
			110

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

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

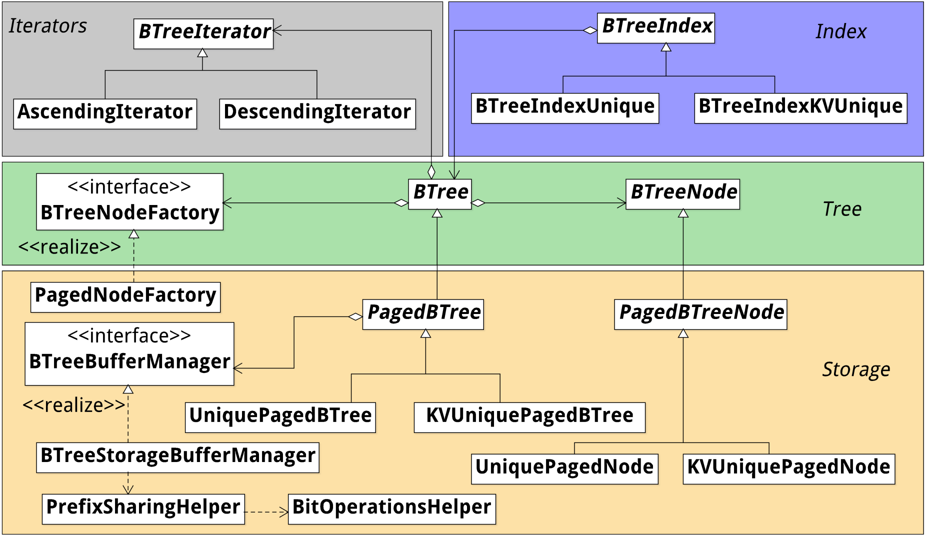
Challenges

- 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

Operations

- Search - similar to normal B+ Tree

Operations

- Search - similar to normal B+ Tree
- Insert overflow
  1. redistribute left ?
  2. split

Operations

- Search - similar to normal B+ Tree
- Insert overflow
  1. redistribute left ?
  2. split
- Delete underflow
  1. merge with left/right ?
  2. split between left and right ?
  3. redistribute left/right

Operations

- Search - similar to normal B+ Tree
- Insert overflow
  1. redistribute left ?
  2. split
- Delete underflow
  1. merge with left/right ?
  2. split between left and right ?
  3. redistribute left/right
- Write
  - only write dirty nodes
  - prefix encoding

Operations

- Search - similar to normal B+ Tree
- Insert overflow
  1. redistribute left ?
  2. split
- Delete underflow
  1. merge with left/right ?
  2. split between left and right ?
  3. redistribute left/right
- Write
  - only write dirty nodes
  - prefix encoding
- insert/delete more constly, exactly how much?

Microbenchmarks

- full in-memory, index only tests

Duration

Operation	No Prefix sharing	Prefix sharing
Search	1	0.9 - 1.1
Insert	1	1.6 - 2.8
Delete	1	1.45 - 2.9

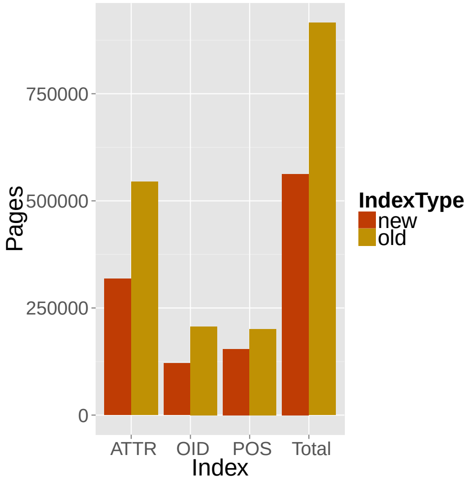
Size of B+ tree

Operation	No Prefix sharing	Prefix sharing
Insert	1	0.5 - 1.1
Delete	1	0.5 - 0.75

StackOverflow Data Import

- real-world workload
- StackOverflow data
  - 1.3 million users
  - 10.3 million posts
  - 13 million comments
  - 25 million votes
- 3 key unique attribute indexes
- 9 key-value unique indexes

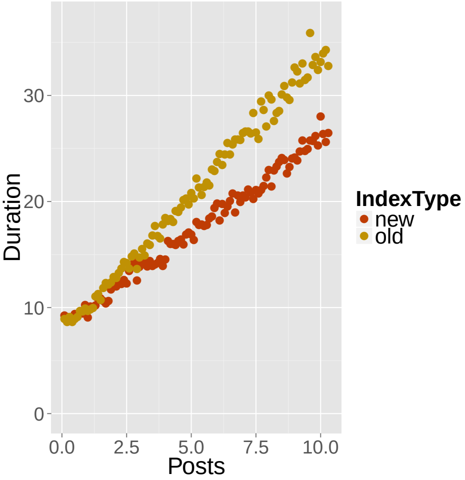
StackOverflow Import - Index Sizes



- page size: 4KB
- database size: 31 GB

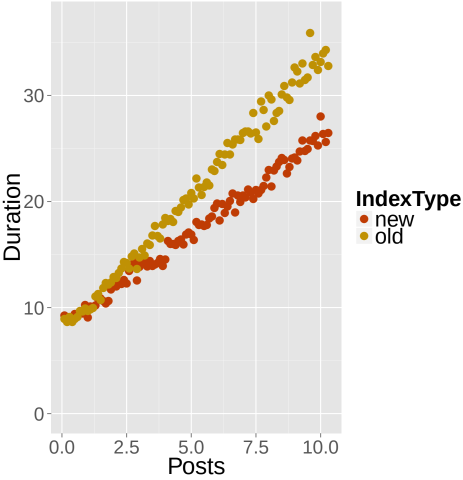
Index	Space saving (%)
Atrribute	41.6
OID	41.5
POS	23.1
Total	38.5

StackOverflow Import - Commit times



- import with new index 25% faster
- why?

StackOverflow Import - Commit times



- import with new index 25% faster
- why?
- more entries in a node
  → fewer dirty nodes
- data locality

Summary

- prefix sharing: trade-off between speed and space
- works well in practice
- microbenchmarks
- implementation complexity.

Q&A

- Thank you for your attention!
- Questions ?