Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

INDEX COMPRESSION

Manuel Blechschmidt

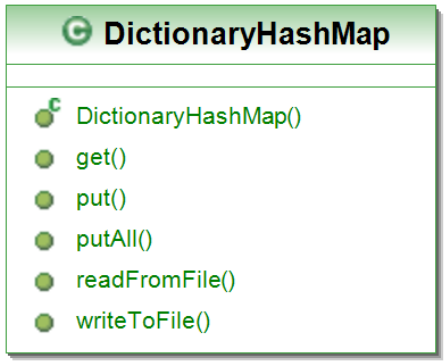
Benjamin Eckart

SoSe 2009

Agenda

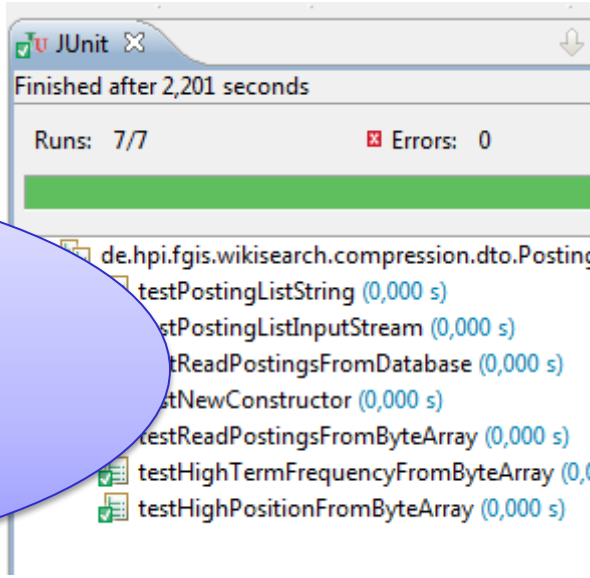
1. What We Have Achieved
2. Structure of Compressed Index
3. Evaluation / Comparison
4. Live Demonstration
5. Conclusion

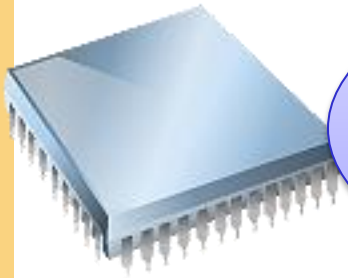
What We Have Achieved



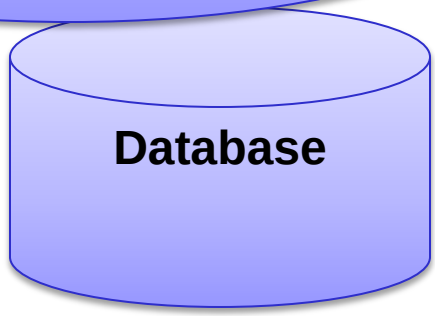

```
Index Compression by Manuel Blechschmidt and Benjamin Eckart
Please enter command:
help
Unknown command. Valid commands are:

load      Load index from file
save      Saves index to file
index     Starts indexing from database
query     Queries term from index, e.g.
exit      Exit program
```





Index Compression Tool



Index Compression Structure

Use multiple Byte Buffers and store them in a HashMap

hashCode → [size of postings, size of term, term, postings], [size of postings, size of term, term, postings]…

- Posting List store documentIds and positions
- Numbers in Posting List are stored using delta encoding and vints

Problem: Java HashMaps need lots of RAM

e.g. index with terms only, about 750 MB with Java HashMaps

one instance of *Object* takes 16 Byte!

Solution:

Implementation of own HashMap algorithm, which consumes less objects per entry

Implementation of Own HashMap

ID	Bytes
0	10111010101110…
1	10101110110010…
2	
3	
4	01101101010100…
5	
6	11110110101010…
7	
…	

Put

1. Calculate HashCode of Entry
2. Map HashCode to number of array
3. Add and compress Dictionary Entry to array slot

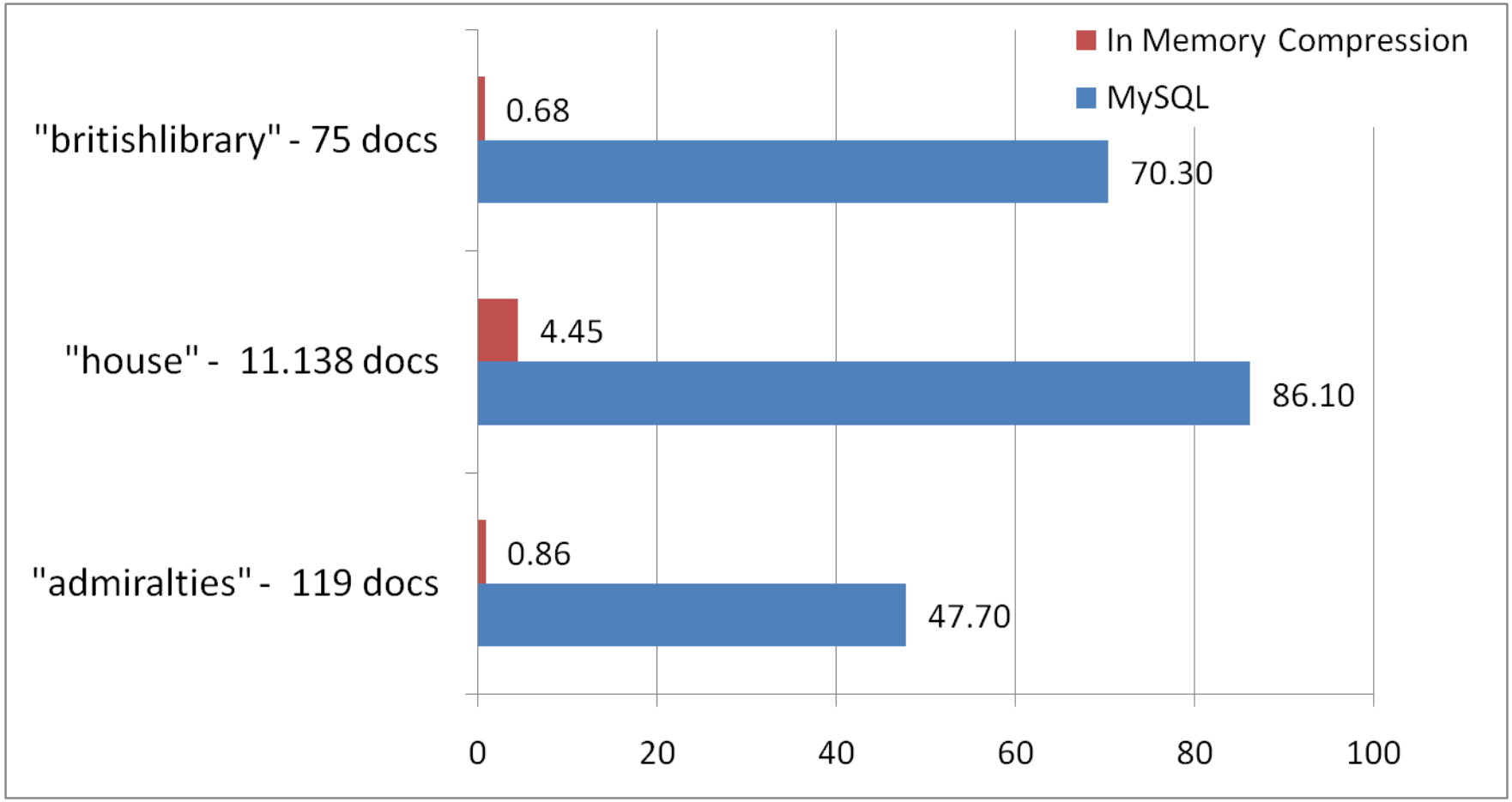
Get

1. Calculate HashCode of Entry
2. Map HashCode to number of array
3. Decompress array slot
4. Return Postings

Evaluation / Comparison

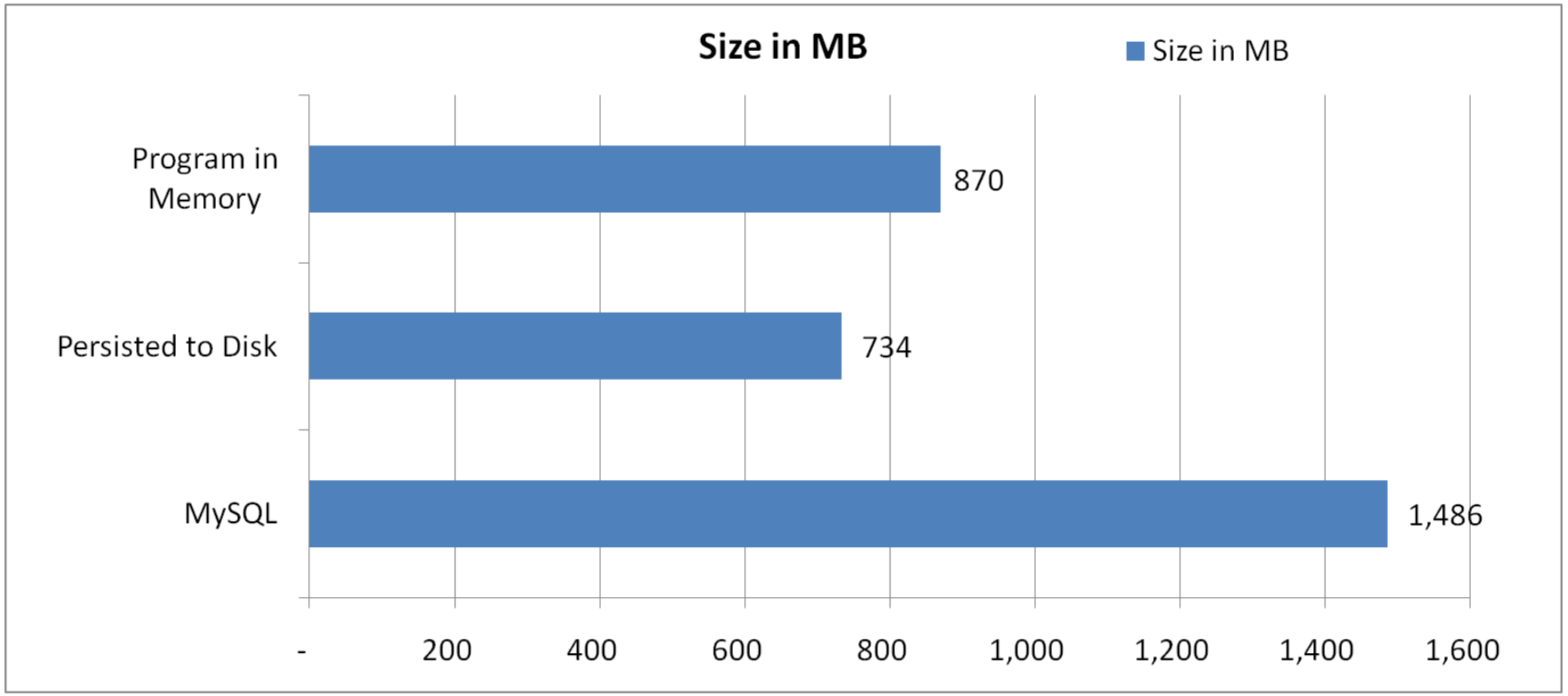
- In-memory representation vs. MySQL table
- Test Data Set: all rows of example inverted index
- Performance measurement
- Size of index

Performance



All times in miliseconds - MySQL without Cache

Memory cosumption



Live Demo


```
Index Compression by Manuel Blechschmidt and Benjamin Eckart
Please enter command:
help
Unknown command. Valid commands are:

load      Load index from file
save      Saves index to file
index     Starts indexing from database
query     Queries term from index, e.g. query house
exit      Exit program
```

Conclusion

- Standard Java Library does not provide memory efficient handling of objects
- About 50% memory saving can be achieved with simple compression techniques
- In-Memory handling makes querying a lot faster
- Also updates of index are possible