Mitchell Neville

CSC 591

Capstone Project: Bitmap vs. B-Tree Indices for Big Data

INTRODUCTION

Despite the fact that there is much evidence to the contrary, it is still a popular belief that Bitmap Indices should be used primarily on columns with few distinct values. This is evidenced by the fact that Wikipedia's first few sentences support this idea[1], as well as popular Oracle consultants[2]. Recent research, however, indicates that this idea is a myth and that Bitmap Indices are well suited to columns with many distinct values. This project seeks to provide some clarity on the issue by benchmarking Bitmap and B-Tree Indices and comparing and contrasting their performance in a variety of situations. In particular, this project will explore Bitmap and B-Tree Indices by analyzing their performance with a variety of dataset cardinalities and column value ranges.

IMPLEMENTATION DETAILS

To begin the investigation, both Bitmap and B-Tree Indices will be benchmarked with a variety of queries. Both equality and range queries will be performed, as well as a combination of the two. The dataset for this be purely numerical, and queries on string values will not be attempted. In addition, benchmarks will be based on the cardinality of the dataset, so that the number of rows will vary from test to test in order to establish a clear correlation between query types, index types, and dataset cardinality. Rather than implement Bitmap or B-Trees independently, two existing technologies will be adopted instead: FastBit[3] and MapDB[4].

FastBit is a library for data processing developed by Berkely Lab's Scientific Data Management Research Group. It uses compressed bitmap indexes, specifically Word-Aligned Hybrid (WAH) compression, to enable efficient search. Since this benchmark project is based in Java, the Java Native Interface (JNI) code for FastBit will be used. FastBit also offers several binning and encoding strategies that will be explored, and these are explained in more detail in the Results section. FastBit is an open-source project, and its code along with the JNI can be found on GitHub.

MapDB is a Java implementation of an embedded database engine, which provides a persistent data model both in-memory and on-disk. MapDB's B-Tree implementation is the Java class "BTreeMap," and the authors of MapDB indicate that its design was based primarily on Sagiv's "Concurrent Operations on B*-Trees with Overtaking". The implementation provides concurrent insertion, removal, updating, and reading, using B-Link-Trees. MapDB is an open source project, so that its B-Tree implementation can be examined in GitHub. While FastBit provides a framework to easily make queries similar in nature to SQL expressions, MapDB is a little less straightforward. Queries on attributes in MapDB requires binding a secondary key, and while equality queries are easy, range queries are more challenging. For more information about how MapDB queries were built, consult the "MapDbBookRepository" class of the source code.

The dataset used for this project consists of information about books, and it has been imported and modified from open-source resources for "A Programmer's Guilde to Data Mining"[6]. This data set contained information about many books, including ISBN, title, author, and publishing date. While text

information is not used for indexing, numerical data like the publishing year is. In addition, random, fictional values for book price have been added to the dataset. The values of the price column are in USD, with dollar amounts ranging from 0 to 200 and cents ranging from 0 to 99. While these price values are not close to their real-world counterparts, the idea is that the publishing year provides a column for values whose range is fairly small, while the price column will provide a much wider range of values. The entire dataset consists of 72,177 rows; however, benchmarks will be performed on three cardinalities: the first 1,000 rows, the first 10,000 rows, and the first 70,000 rows.

QUERY RESULTS AND ANALYSIS

The results from each query are depicted below. The results measure the time it took for the query to be processed in milliseconds. Three different types of queries are conducted on varying cardinalities of the dataset, including 1,000 rows, 10,000 rows, and 70,000 rows. In addition, different types of FastBit encodings and binning strategies are explored.

FastBit offers encoding options geared toward both equality and range queries. Using the range encoding, range queries should be more efficient at the cost of more disk storage. According to FastBit documentation, this encoding is recommended for "low" column cardinalities. Equality coding is recommended for basic schemes with no binning. In addition, FastBit offers option concerning binning. We will examine cases where there are no bins and where a certain number of bins are prescribed.

1. Query: "SELECT * FROM books WHERE year = 2001"

Cardinality	FastBit (msec)			MapDB (msec)
	No Bins, Equality Encoding	2,000 Bins, Equality Encoding	2,000 Bins, Range Encoding	
1,000	6	8	6	6
10,000	15	14	10	12
70,000	22	7	20	27

Table 1 - Equality Query Results

With low cardinality, the results for both MapDB and FastBit are relatively the same. When the cardinality approaches 70,000, however, it is clear the FastBit has an advantage when using binning. It also appears that using FastBit's equality encoding has an advantage over range encoding.

2. Query: "SELECT * FROM books WHERE year $\geqslant$ 2000"

Cardinality	FastBit (msec)			MapDB (msec)
	No Bins, Equality Encoding	2,000 Bins, Equality Encoding	2,000 Bins, Range Encoding	
1,000	2	3	>1	9
10,000	4	1	>1	30
70,000	5	>1	2	57

Table 2 - Range Query Results

From the results, it appears that FastBit has a clear advantage over MapDB's B-Tree Indices when it comes to range queries. It is also apparent that range encoding provides a clear advantage even with a column whose values are pretty highly distinct.

3. Query: “SELECT * FROM books WHERE year = 2000 AND price ⩽ 100.00”

Cardinality	FastBit (msec)			MapDB (msec)
	No Bins, Equality Encoding	2,000 Bins, Equality Encoding	2,000 Bins, Range Encoding	
1,000	8	6	8	3
10,000	27	30	28	9
70,000	65	105	68	21

Table 3 - Mixed Query Results

From these results, it appears that MapDB’s B-Tree Indices perform better for a mix of range and equality queries.

CONCLUSION

It is a somewhat common misconception that Bitmap Indices do not perform well for columns with a high cardinality of distinct values. From the results collected for this project, it is clear that is not the case. FastBit provides a means of Bitmap Indices; however, it has a lot of customization options to tune the indices to the data. From Table 2, it is clear that, with a column of highly distinct values like the price column, Bitmap Indices can provide very good performance compared to B-Tree Indices. In addition, according to Table 1, Bitmap Indices can provide good performance on equality queries if the number of rows is high enough. However, Table 3 indicates that B-Tree Indices provide somewhat better support for a mixed query. This may be because of the limited tuning options explored in this project for FastBit.

FUTURE WORK

It is worth exploring FastBit’s options more in-depth in the future in order to determine how best to suit the indices to the dataset. This investigation was an early exploration of Bitmap and B-Tree Indices technologies, and a better understanding of FastBit’s options may provide better performance in mixed queries. In addition, other implementations of Bitmap and B-Tree Indices are worth exploring to determine how they compare to the baseline set in this project.

REFERENCES

1. "Bitmap Index." *Wikipedia*. N.p., n.d. Web. 4 May 2015. <http://en.wikipedia.org/wiki/Bitmap_index>.
2. "Oracle Bitmap Index Maximum Distinct Values." *Burleson Consulting*. N.p., 8 Mar. 2010. Web. 4 May 2015. <http://www.dba-oracle.com/t_bitmap_index_maximum_distinct_values_cardinality.htm>.
3. *An Efficient Compressed Bitmap Index Technology*. Berkeley Lab Scientific Data Management Research Group, n.d. Web. 4 May 2015. <https://sdm.lbl.gov/fastbit/>.
4. *MapDB*. N.p., n.d. Web. 4 May 2015. <http://www.mapdb.org/>.
5. Sagiv, Yehoshua. "Concurrent Operations on B*-Trees with Overtaking." *Journal of Computer and System Sciences* 33.2 (1986): 275-296. Web. 4 May 2015.
6. "A Programmer's Guide to Data Mining." N.p., n.d. Web. 4 May 2015. <https://github.com/bigsnarfdude/guide-to-data-mining>.