Eindhoven University of Technology

Department of Mathematics and Computer Science

2ID35 - Database Technology

Verifying "Lightweight Indexing of Observational Data in LogStructured Storage"

Authors:
Arjan Broos
Philip Klees
Ramon de Vaan

Professor:
dr. George Fletcher

June 22, 2014
Eindhoven

Abstract

For the course Database Technology (2ID35), the paper: "Lightweight indexing of observational data in log-structured storage" is analyzed. In this paper the authors describe CR-Index, a lightweight indexing method. The authors make claims concerning the performance of CR-Index in comparison with B^+-tree and LSM-tree. In this report these claims are validated. The process of validating the claims is described. At the end of the report an extension to the paper is given. Apart from the CMOP and DEBS dataset (explained later in the report in more detail), a stock market dataset is added as well.

1 Literature survey

Data collection is a very important aspect in scientific research, and in this day and age, the methods we have to record data are growing rapidly. Sensors are increasing in resolution temporally, spatially, and the bits of precision captured, thereby increasing the data rate of an individual sensor. Moreover, sensors are decreasing in price and power consumption, meaning that more sensors can be deployed.

These developments call for data storage with high write throughput, to capture the ever-increasing observational data rate in a timely matter. Such data storage already exists in the form of log-structured storage. However, existing indexing and querying techniques do not take into account traits specific to observational data, and as such, leave much to be desired performance-wise. The paper proposes an approach to indexing and querying observational data in log-structured storage, increasing performance by exploiting traits specific to observational data.

2 Research problems

In some cases, a database is required to have a high throughput when inserting data. Databases and indexing schemes designed for write-intensive scenarios already existed, but their setup caused slow querying of the database. The paper focuses on a certain type of data, observational data, and makes use of some of its properties. By being able to make reasonable assumptions about this type of data, such as continuity, the authors came up with an indexing method. This method not only handles write-intensive scenarios well, but performs queries on the database rather efficiently, compared to existing solutions.

3 Description CR-index

In this section, the indexing method of the paper is summarized.

The authors of the paper make some assumptions about observational data, on which they built their indexing method. These assumptions are the following:

- No update - Once a record is inserted, it will not be altered.
- Continuity - Values change continuously at some maximum rate.
- Potential discontinuity - Due to noise or data loss, gaps could arise.

Data is stored in a log file, which means that any record is simply appended to the file in the order that they arrive in. The arrival order is expected to be very close to an ordering on their actual timestamps. At least, when a record a is generated by sensor S before a record b from the same sensor, a is expected to be inserted before b.

This log file is divided into many chunks of equal size. For every chunk, we keep an interval of the minimum and maximum value of any record within it. If we make the continuity assumption, querying for a range that overlaps this interval means that we will always find a record in this chunk that satisfies the query. In reality, false positives will happen, but rarely. The technique of using an interval to describe the data within each chunk in a log structure is called the CR-index.

A list with all of these CR-indices is called the CR-log. Any record in this log contains information about the block's interval, its position on disk and some extra hole information. If this CR-log fits into main memory, you can do a linear scan on the log for a query. If it does not, you need to index it. The paper describes the use of B^+-trees and interval trees to do so.

Furthermore, the paper describes optimizations of this structure: how to speed up queries, deal with continuities and disordered records.

This very light-weight indexing scheme allows for high write throughput, while still allowing efficient handling of queries.

4 Evaluation plan

In this chapter, the steps taken for the validation of the claims of the paper are discussed. The steps are divided in 3 project phases, namely: Literature survey, implementation and the verification phase.

4.1 part 2: literature survey

When starting working on the project, each team member read through the paper alone. As soon as everyone finished reading the paper, the paper was divided among the different group members. By doing so, each of the three group members could start working on the literature survey, which was the first step (after choosing a project). Plans where made on how to keep working in the right direction. We decided to ask the authors for the source code they used for CR-Index. In this phase also the plans where made to add the YHOO stock market dataset to the to be analyzed datasets. There were also plans to create a dataset ourselves, and see how CR-Index performed on it. However, this plan was dropped after discussing it with dr. Fletcher. After making these plans, the first phase of the project was finished.

4.2 part 3 - implementation phase

This phase was focussed on implementing the ideas of the paper. When the authors replied on our request to get their source code, we thought this would save us much time. When diving into the code however, it became clear that we only got the implementation for CR-Index from the authors. This means that implementations for the B^+-trees and LSM-trees had to be realized. This delayed the plans to start working on the validation. The implementation of CR-Index did not compile immediately either, and took a lot of time to get it working properly. At the end of this phase a meeting with dr. Fletcher was scheduled and these problems were discussed. At this point the progress we made looked nice, though nothing was working as it should yet and we were not ready for validating the claims yet. After a few weeks of hard work, the implementations of both the B^+-trees and LSM-trees as the CR-Index was finished and we were ready for the next phase.
In this phase B^+-trees, LSM-trees and CR-Index have been applied on the YHOO set as well. This set was chosen, because the paper mentioned that CR-Index might also be interesting while using it on stock market datasets.

4.3 part 4 - validating claims

When starting the validation of the claims, some problems were detected again. While running CR-Index for the first time on the CMOP dataset, it took around 84 minutes to finish. Our implementation of the B^+-tree and LSM-tree took around 10 minutes to finish. This obviously is in heavy contrast with the paper. This was due to the open source implementations of the trees not supporting duplicate values. We fixed this by storing an array of offsets with every value, which might have been a different solution from the one the authors used. Now, with the correct amount of indexed values, the B^+-tree and LSM-tree took many hours to finish.

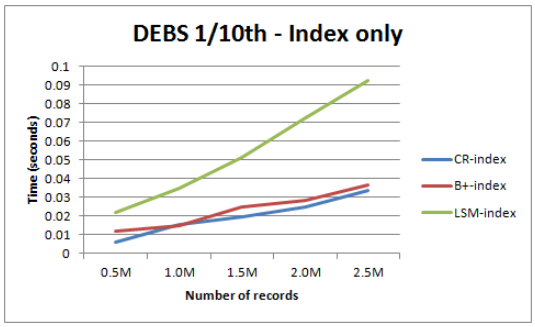
5 Validation of results

In this section, we will attempt to validate the claims made in the paper. We compare CR-index to LSM and B+ based indexing algorithms. The paper uses two datasets to test all methods on, namely DEBS and CMOP.

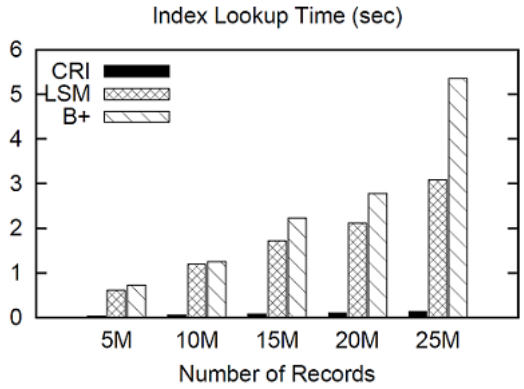
We specifically compare the methods on index-only search, query by index, and write performance. Index-only is where we search the index for any values within a given range, but do not actually retrieve those values from disk. In query by index, the index is used in a query to find all values within a given range. Write performance measures how long it takes for the dataset to be written to the database, including the building of the index. In the following sections, we will present our results and the paper's results for both the DEBS and the CMOP datasets, compare the results, and explain any differences in the data.

5.1 DEBS

The DEBS dataset contains real-time data on the velocity, acceleration of the ball during a soccer game. However, in the paper, only the velocity attribute is indexed and queried. The results are shown in figures 1, 2, and 3.

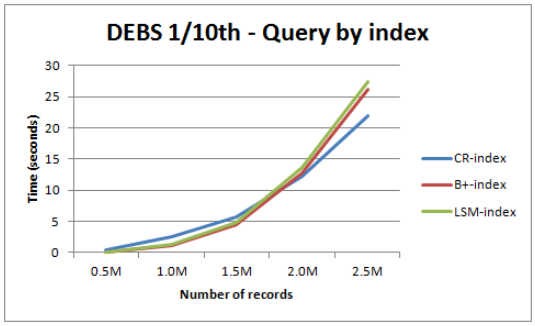


(a) our results

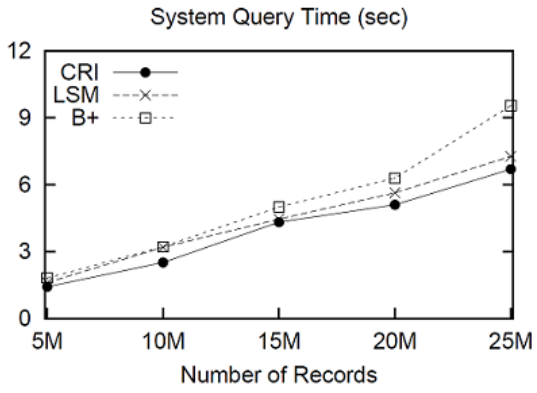


(b) paper's results

Figure 1: debs: Index-only lookup times

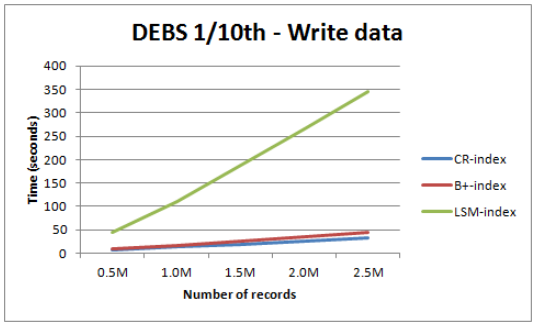


(a) our results

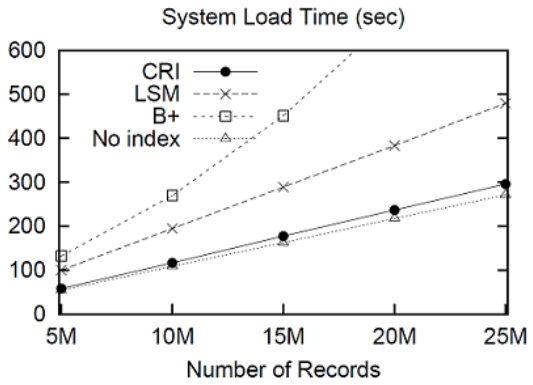


(b) paper's results

Figure 2: debs: Query by index



(a) our results

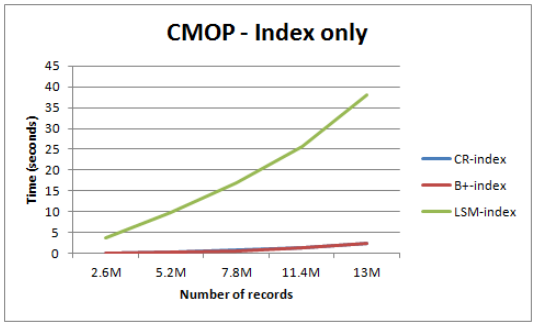


(b) paper's results

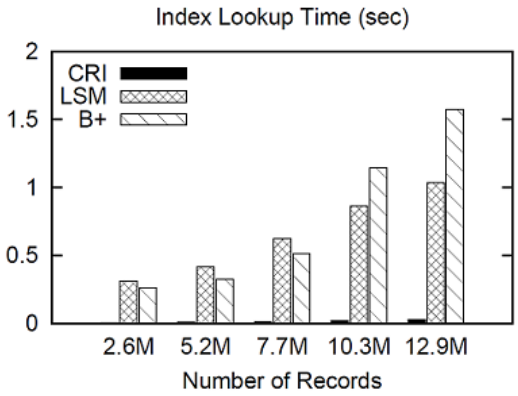
Figure 3: debs: Write performance

5.2 CMOP

The CMOP dataset contains coastal margin data collected from the CMOP SATURN Observing System. It contains data on the salinity, temperature and oxygen saturation of oceans and rivers. However, only the salinity variable is indexed and queried. The results are shown in figures 4, 5, and 6.

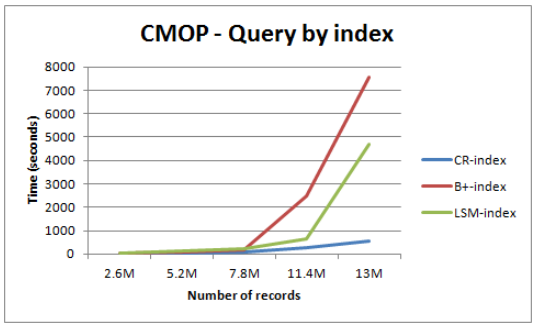


(a) our results

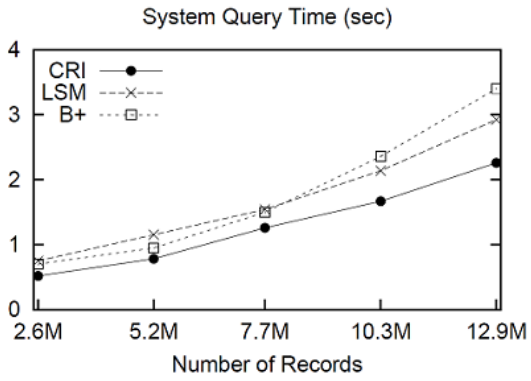


(b) paper's results

Figure 4: CMOP: Index-only lookup times

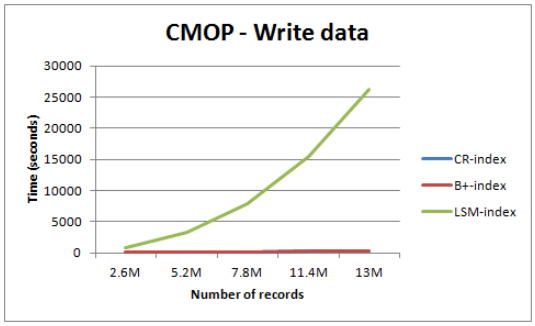


(a) our results

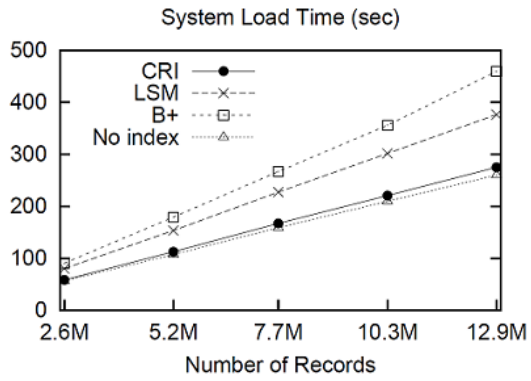


(b) paper's results

Figure 5: CMOP: Query by index



(a) our results



(b) paper's results

Figure 6: CMOP: Write performance

5.3 Comparison

First of all, supporting the paper's claims, cr-index performs admirably in the tests, performing either equally well, or better than both B^+-trees and LSM in all fields of measurement. However, LSM seems to perform much worse than described in the paper. In the paper, LSM performs better than B^+-trees in most cases. In our results, LSM performs very badly in index-only lookup, and write data. In query by index it seems to perform more proportional to the paper's results.

This may be due to the fact that we had to implement the B^+-trees and LSM ourselves. Even though we had access to the same libraries, our implementations will probably have differed from the implementation used in the paper. Another reason may be that the environment on which we ran the tests is not exactly equal to the paper's, as our computer had slightly lower RAM.

It should also be considered that for the DEBS dataset, our test ran on less tuples, because querying was taking too long, perhaps taking multiple days to complete. However, since scalability is one of the major benefits of cr-index, as shown in the CMOP tests, it can be argued that if the test was run on a larger part of the DEBS dataset, cr-index would have still performed better in comparison to indexing methods not aimed at scalability, like LSM and B^+-trees.

6 Extension of paper - YHOO dataset

In the paper it is mentioned that the CR-Index might give interesting results on a stock market dataset. Therefore, we decided to look for such a set, and the YHOO set was selected. YHOO is the name registered by Yahoo! on the stock market.

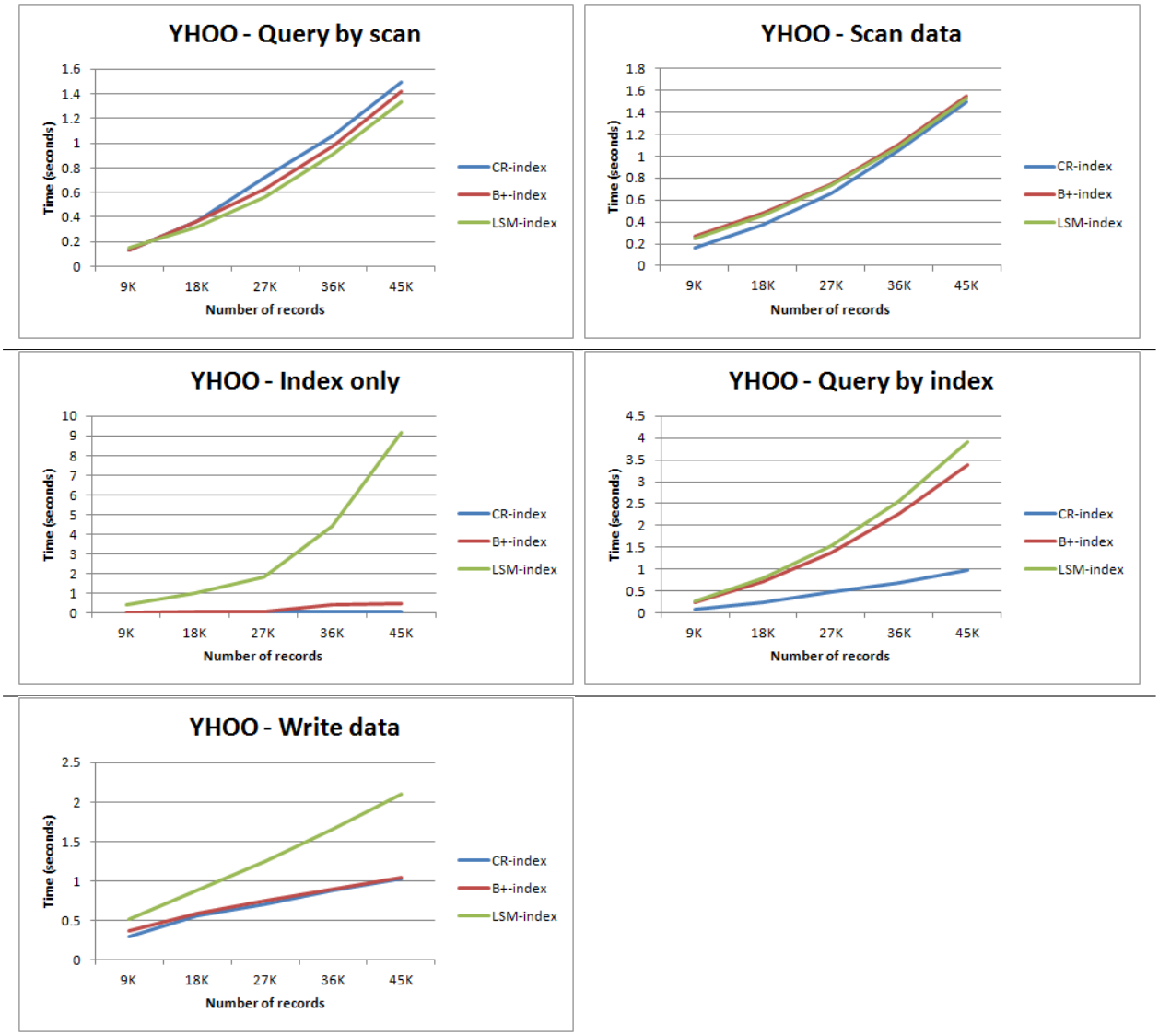


Table 1: results of the YHOO dataset

6.1 Scan data

'Scan data' means opening the log file and scanning through it without the use of an index. Just for safety, the required time for this is measured every time a test for an indexing method was set up. As can be seen, they remain very similar.

6.2 Query by scan

'Query by scan' is similar to 'Scan data', but instead of just scanning through, it finds values in a given range and returns them once done with the scan. Of course, these results remain very similar again.

6.3 Index only

'Index only' is where we search the index for any values within a given range, but do not actually retrieve those values from disk. What is interesting to see in this graph, is that LSM-tree performs many times worse than both B^+-tree and CR-Index. LSM-tree runs in 9,19 seconds, while B^+-tree and CR-Index run in 0,44 and 0,08 seconds respectively.

6.4 Query by index

'Query by index' is where the index is used in a query to find all values within a given range. The power of CR-Index really is shown here. In the graph we can clearly see that CR-Index performs 3 to 4 times better than B^+ and LSM-tree. This was expected, because CR-Index was optimized for querying on observational, continuous data.

6.5 Write data

'Write data' measures how long it takes for the dataset to be written to the database, including the building of the index. Here, we see that LSM-tree performs worse than both B^+ and CR-Index again. The performance is almost twice as bad as B^+ and CR-Index. The latter two, however, have a similar performance.

7 Conclusion

The authors claimed that CR-Index had the better write performance, while still maintaining a similar query performance, when compared to B^+ and LSM trees. Even though our results most certainly support this claim, our results are far more in favor of the CR-index than the ones in the original paper. This might be due to our adaptations of the trees, used for generating the results.

8 Future work

In this section some suggestions are given in order to further validate the claims of the authors of "Lightweight Indexing of Observational Data in LogStructured Storage". Unfortunately, there was too little time to test all claims by the authors. We had plans to test CR-Index on a dataset created by ourselves, however we had to drop this idea. For this project, we focussed on researching how CR-index perform compared to B^+ and LSM-tree.

8.1 Validate various implementations of CR-Index

Because of the short period available to work on the project, there was too little time to validate different implementations of CR-Index. In the paper, different implementations of CR-Index are discussed. In the paper memory-based (mCRL) as well as disk-based (dCRL) implementations are discussed. Both these implementations can be extended with indexing, in the paper these got the names: (mCRL + Index) and (dCRL + Index). It might be interesting to find out which implementation works best on what sort of dataset. Therefore we want to mention this as future work.

8.2 Validate fairness

One possible improvement on our research is to check if the comparison between the implementation of CR-Index is really fair. Before every query that is measured some prequery is executed which is not measured. This prequery might positively influence the running time of the measured query and therefore result in better running times. If this research is going to be extended, this definitely has to be looked in to.

Appendices

A Retrieving datasets

Due to the size of the datasets used (several gigabytes), we thought it might be more convenient to give a description of how to retrieve those datasets from the web.

A.1 CMOP

- Go to `http://www.stccmop.org/datamart/observation_network/dataexplorer`
- Click "Add a new plot"
- For source, fill in "SATURN-01"
- For variables, select the following: "Oxygen saturation [Oxygen]", "Salinity [CT]" and "Temperature [CT]"
- Click on "Add series"
- Click "Next"
- For "Start date", select 2011-4-1 00:00
- For "End date", select 2012-8-31 23:59
- Click "Next"
- Click "Next" three more times
- For all three graphs, click the download icon and click "Download"
- Merge these three files, as a natural join on the timestamp
- Since oxygen saturation is stored at a different interval from the other two, fill in the blanks with the last known value for oxygen saturation

A.2 DEBS

- Go to `lafayette.tosm.ttu.edu/debs2013/grandchallenge/full-game.gz`

A.3 YHOO

- Go to `finance.yahoo.com`
- Select "YHOO" and click on historical prices
- Click "Download to spreadsheet"