Department of Informatics, University of Zürich

BSc Thesis

Implementing an Index Structure for Streaming Time Series Data

Melina Mast

Matrikelnummer: 13-762-588

Email: melina.mast@uzh.ch

August, 2016

supervised by Prof. Dr. Michael Böhlen and Kevin Wellenzohn

University of Zurich[UZH]

Department of Informatics

Acknowledgements

I especially would like to thank my supervisor Kevin Wellenzohn, who has supported me with guidance and constructive feedback during my work. Besides, I would like to thank Prof. Dr. Michael Böhlen for the opportunity to write my bachelor thesis at the Database Technology Group of the University of Zurich. ...

Abstract

...

Zusammenfassung

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

A streaming time series s is a unbounded sequence of data points that is continuously extended, potentially forever. They are relevant to applications in diverse domains like in finance, meteorology or sensor networks to name a few. All these applications need to be fed continuously with the latest data. But the processing of large volumes of time series data is impractical because a system can only keep a limited size of data in main memory.
Since streaming time series are unbounded, a system cannot hold the constantly generated data in his main memory endlessly. Therefore, the data that is kept in main memory needs to be limited to just a portion of the streaming time series. Besides, in order to be practical for a application like the financial stock market, the data that arrives in a defined time interval (e.g. every 2 minutes) needs to be completely processed until the succeeding data arises.
The thesis presents a way to implement the described data structures after discussing the requirements. Furthermore, it documents the out coming experimental results. In the end of the thesis, in Chapter 6, the findings will be summarized and concluded.

2 Background and Related Work

Time series data is not always gapless. E.g. due to sensor failures or transmission errors values can get missing. But since many applications need complete data before further data processing is possible, the missing values need to be recovered first.
The paper *Continuous Imputation of Missing Values in Highly Correlated Streams of Time Series Data* [1] presents a query pattern over the most recent values of a set of time series. The algorithm was developed in the context of meteorology. The *Südtiroler Beratungsring (SBR)* operates a network of weather stations where each station collects continuously (every five minutes) new temperature data. If a value is missing due to sensor failures or other problems, it will be continuously imputed. The imputation needs to fulfill two main requirements:

- The algorithm needs to be efficient enough to complete the data imputation before the new measurement arrives.
- The imputation must rely on past measurements.

The algorithm provided in the paper is called Top-k Case Matching (TKCM) algorithm. It defines a two-dimensional query pattern that contains the measurements of the reference time series in a window of time. It seeks for the k patterns that match the query pattern best. The missing value is derived from the k past pattern. Therefore, it determines for each *time series* a small set of highly correlated *reference time series*. To scan the entire data set to identify the k best is not practical. Hence, only a small portion of the data is scanned. The algorithm is described in Section 2.

2.1 TKCM

The implementation of the index structure described in the following thesis is a fundamental part of the TKCM algorithm.

3 Preliminaries

3.1 Handling Duplicate Values

B+ trees are often

3.1.1 Add the time to the key

If you would add the timestamps to keys would be unique. So if the case occurs where you have the same keys but in different leaves because the leaves were full and had therefore be splitted you could include the timestamps which divides the two different values to the parent. –> every key with timestamp smaller than the parents timestamp is in the left leaf and every key with timestamp bigger or equal is in the right one –> Problems: more memory needed. always check needed if there is a timestamps in the leaf + neighbour method would still work because you have the value and the set of time stamps

3.1.2 Add several times to the key

The second was to have just one entry for each key, but the 'payload' associated with each key points to a different block of data, which is a linked list pointing to all instances of items that are having the same key

3.1.3 Not change insertion but the search algorithm

There are several different ways of handling duplicate keys. One way is use an unmodified insert algorithm which allows duplicate keys in blocks but is otherwise unchanged. The issue with a structure such as this is the search algorithm must be modified to take into account several corner cases which arise For instance one of the invariants of a B+Tree may be violated in this structure. Specifically if there are many duplicate keys, a copy of one of the keys may be in a non-leaf block. However, the key may appear in blocks that which appear logically before the block which is pointed at by the key in the internal block. Thus the search algorithm must be modified to look in the previous blocks to the one suggested by the unmodified search algorithm. This will slow down the common case for search.There is another issue with this straight forward implementation, if there are many duplicate keys in the index, the index size may be taller than necessary. Consider a situation were for each unique key there are perhaps hundreds of duplicates, the index size will be proportional to the total number of keys in the main file, however, you only need to index an index on the unique keys. One of the files indexed in our program will be indexing has such characteristics to its data. It indexes strings

(as the keys) with associated instances where those strings show up in our documents. There can be hundreds to thousands of instances of each unique string. Therefore the approach I took was to store only the unique keys in the index, and have the duplicates captured in overflow blocks in the main file. An example of such a tree can be seen in figure 2. Consider key 6; there are 5 instances of this key in the tree. The tree is order 3, indicating the keys cannot all fit in one block. To handle this situation an overflow block is chained to the block which is indexed by the tree structure. The overflow block then points to the next relevant block in the tree.To create a structure such as this, the insert algorithm had to be modified. Like the previous version these modifications do not come without a cost, in particular the invariant which states all block must be at least half full has been relaxed. This is not true in this B+Tree, some blocks like the one containing key number 7, are not half full. This problem could be partially solved by using key rotations to balance the tree better. However, there are still corner cases where there would be a block which is under-full. One such corner case includes when a key falls between two keys which have overflow blocks. It must then be in a block by itself, since this B+Tree has the invariant which state that if a block is overflowing it can only contain one unique key. In the future we would like to implement key rotations to help partially alleviate the problem of under-full blocks.

The advantage of this approach to B+Trees with duplicate keys is the index size is small no matter how the ratio of duplicate keys to the total number of keys in the file. This property allows our searches to be conducted quicker. Since the overflow blocks are chained into the B+Tree structure we still have the property of being able to fast sequential scans. One consequence is we have defined all queries on our B+Trees to be range queries. This is fine because all of our queries were already range queries. In conclusion we relax the condition that all blocks must be at least half full to gain higher performance during search.

http://hackthology.com/lessons-learned-while-implementing-a-btree.html

4 Implementation

Let $W = [\underline{t}, \bar{t}]$ be a sliding window of length $|W|$. Time $\underline{t}$ stands for the oldest time point that fits into the time window and $\bar{t}$ stands for the newest time point for which the stream produced a new value.

The system uses two different data structures: a circular array and a $B+$ tree. needs to be efficiently perform the following operations on the streaming time series s in a sliding window $|W|$:

- shift($\bar{t}, v$): add value *v* for the new current time point $\bar{t}$ and remove value *v'* for the time point $\underline{t} - 1$ that just dropped out of time window W.
- lookup(t): return the value of time series *s* at time *t*, denoted by $s(t)$.
- neighbour(v, T): given a value *v* and a set of time points T, return the time point teT such that $|v - s(t)|$ is minimal.

5 Experimental Evaluation

5.1 Experimental Setup

5.2 Results

5.3 Discussion

6 Summary and Conclusion

"The conclusion (10 to 12 per cent of the whole research thesis) does not only summarize the whole research thesis, but it also evaluates the results of the scientific inquiry. Do the results confirm or reject previously formulated hypotheses? The conclusion draws both theoretical and practical lessons that could be used in future analyses. These lessons are to be embedded as 2 recommendations for the research community and for policy-makers (note: policy relevance instead of policy prescriptive). In addition, the conclusion gives insights for further research."

Bibliography

[1] K. Wellenzohn, M. Böhlen, A. Dignos, J. Gamper, and H. Mitterer: *Continuous imputation of missing values in highly correlated streams of time series data*; Unpublished, 2016.

[2] Themistoklis Palapanas, Michail Vlachos, Eamonn Keogh, Dimitrios Gunopulos, Wagner Truppel: *Online Amnesic Approximation of Streaming Time Series*; University of California, Riverside, USA, 2004. `http://www.cs.ucr.edu/~eamonn/ICDM_2004.pdf`

[3] Xiaomei Dong, Xiaohua Li: *An Authentification Method for Self Nodes Based on Watermarking in Wireless Sensor Networks*; Northeastern University, Liaoning, China, 2009.

[4] Xiaomei Dong, Xiaohua Li: *An Authentification Method for Self Nodes Based on Watermarking in Wireless Sensor Networks*; Northeastern University, Liaoning, China, 2009.

[5] All-Sakib Khan Pathan, Hyung-Woo Lee, Choong Seon Hong: *Security in Wireless Sensor Networks: Issues and Challenges*; ISBN 89-5519-129-4, DOI: 10.1109/ICACT.2006.206151, Februar 2006. `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1625756&isnumber=34121`