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

The financial stock market, wireless sensor networks, weather forecasts or monitoring systems use streaming time series. A streaming time series is a unbounded sequence of data points that is continuously extended, potentially even forever.
Since it is 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 a part of the 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.

2 Background and Related Work

3 Problem Statement

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 timestamp 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 Solution

5 Experimental Evaluation

5.1 Runtime Complexity

5.2 Space Complexity

6 Summary and Conclusion

6.1 Fixed-Period Problems: The Sublinear Case

With this chapter, the preliminaries are over, and we begin the search for periodic solutions to Hamiltonian systems. All this will be done in the convex case; that is, we shall study the boundary-value problem

$$\begin{aligned} \dot{x} &= JH'(t,x) \\ x(0) &= x(T) \end{aligned}$$

with $H(t,\cdot)$ a convex function of x, going to $+\infty$ when $\|x\| \to \infty$.

Example 1 ((External forcing)) *Consider the system:*

$$\dot{x} = JH'(x) + f(t) \tag{6.1}$$

where the Hamiltonian H is $(0, b_\infty)$-subquadratic, and the forcing term is a distribution on the circle.

6.1.1 Autonomous Systems

We assume a *time domain*, $\mathcal{A}$, as a set of time instants equipped with a total order $\leq$ and isomorph to integers. Time granularities are partitionings of subsets of $\mathcal{A}$ into non-empty intervals of time instants, termed *granules*. Examples of time granularities are minutes (min), hours (hou), days (day), weeks (wee), months (mth), and years (yea). The granularity day, for instance, divides the time domain into granules of 1440 minutes. We assume a *bottom granularity*, $G_\perp$, such that each granule of $G_\perp$ contains exactly one time instant. In our running example minutes represent the bottom granularity, and we use the ISO 8601:2004 notation to denote time instants, e.g., 2007-02-12 07:15. The granules of each granularity G are ordered according to the time domain order and indexed with a subset of integers, $\mathcal{L}_G$, such that the indexing function $\mathcal{M}_G : \mathcal{L}_G \to G$ is an isomorphism that preserves the total order $\leq$. For each granularity we assume that the granule with index 0 contains the time instant 2000-01-01-00:00. Figure **??** illustrates some correspondences of indexes between different granularities, e.g., $\mathcal{M}_{day}(2599)$ = [2007-02-12 00:00, 2007-02-12 23:59].

For the conversion between different granularities, we adopt the *bigger-part-inside semantics* [**?**, **?**]. The conversion from a coarser granularity H to a finer granularity G is defined as

$$\begin{aligned} \lfloor {}^{H}_{G}\ (i) = \{j\ ||\mathcal{M}_G(j) \cap \mathcal{M}_H(i)| > |\mathcal{M}_G(j) \setminus \mathcal{M}_H(i)| \vee \\ (|\mathcal{M}_G(j) \cap \mathcal{M}_H(i)| = |\mathcal{M}_G(j) \setminus \mathcal{M}_H(i)| \wedge \max(\mathcal{M}_G(j)) \in \mathcal{M}_H(i))\} \end{aligned}$$

$\lfloor^H_G\ (i)$ returns the indexes of those granules in G that are covered by granule i in H for more than a half or, if exactly half of a granule in G is covered, the second half.

6.2 Time Slices and Multislices

A (time) *slice* [**?**] is a finite list of pairs, $\lambda = (G_1X_1, \ldots, G_dX_d)$, where G_l are granularities and X_l are *selectors* that are defined as sets of integers. Each selector X_{l+1} specifies a set of granules in G_{l+1} with a relative positioning with respect to granularity G_l. The sequence of granularities $(G_1, \ldots, G_d)$ is the *hierarchy* of the slice.

Consider the slice $(yea\{7\}, wee\{0\text{-}25\}, day\{0\text{-}4\}, hou\{7\}, min\{0,25,55\})$. The hierarchy is (wee, day, hou, min). The first selector $\{7\}$ selects the year 2007, the selector $\{0\text{-}25\}$ selects the first 26 weeks of this year, the selector $\{0\text{-}4\}$ selects the days from Monday to Friday from each of these weeks, etc.

The semantics of a slice $\lambda = (G_1X_1, \ldots, G_dX_d)$ is defined through the following mapping $\mathcal{I}$ to a subset of the time domain:

$$\mathcal{I}(\lambda) = \begin{cases} \bigcup_{k \in X_1} \mathcal{M}_{G_1}(k) & d = 1 \\ \mathcal{I}\left((G_2 \bigcup_{k \in X_1} (\lfloor^{G_1}_{G_2}\ (k)/^+ X_2), \ldots, G_dX_d)\right) & d > 1 \end{cases}$$

Here, $\lfloor^{G_1}_{G_2}$ is a bigger-part-inside conversion from a granularity G_1 to a granularity G_2, and $\lfloor^{G_1}_{G_2}\ (k)/^+ X_2$ is defined as $\lfloor^{G_1}_{G_2}\ (k) \cap \{\min(\lfloor^{G_1}_{G_2}\ (k)) + i \mid i \in X_2\}$.

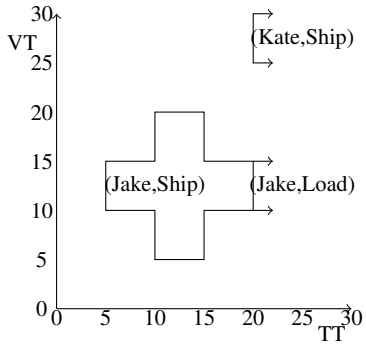

Figure 6.1: Relationships between P, M, and $M_{\min}$

7 The Final One

Lemma 1 *Assume that H is C^2 on $\mathbb{R}^{2n}\setminus\{0\}$ and that $H''(x)$ is non-degenerate for any $x \neq 0$. Then any local minimizer $\widetilde{qx}$ of ψ has minimal period T.*

Proof: We know that $\widetilde{x}$, or $\widetilde{x} + \xi$ for some constant $\xi \in \mathbb{R}^{2n}$, is a T-periodic solution of the Hamiltonian system:

$$\dot{x} = JH'(x)\ . \tag{7.1}$$

There is no loss of generality in taking $\xi = 0$. So $\psi(x) \geq \psi(\widetilde{x})$ for all $\widetilde{x}$ in some neighbourhood of x in $W^{1,2}\left(\mathbb{R}/T\mathbb{Z};\mathbb{R}^{2n}\right)$.

But this index is precisely the index $i_T(\widetilde{x})$ of the T-periodic solution $\widetilde{x}$ over the interval $(0,T)$, as defined in Sect. 2.6. So

$$i_T(\widetilde{x}) = 0\ . \tag{7.2}$$

Now if $\widetilde{x}$ has a lower period, T/k say, we would have, by Corollary 31:

$$i_T(\widetilde{x}) = i_{kT/k}(\widetilde{x}) \geq k i_{T/k}(\widetilde{x}) + k - 1 \geq k - 1 \geq 1\ . \tag{7.3}$$

This would contradict (7.2), and thus cannot happen. □

A reference to a Figure

Bibliography