Computational Indexes for Time Series

Esa Falkenroth
Real-Time Systems Laboratory,
Dept. of Computer and Information Science,
Linköping University, Sweden
esafa@ida.liu.se

Abstract

A fast temporal indexing method for main memory databases is proposed. We consider storing time stamped data from simulation and sensing applications and the subsequent retrieval of the information given a time stamp. The indexing technique is based on super-linear equation solving methods and assumes that new data arrives with monotonically increasing time stamps. We also present some results of a performance measurement of the storage structure and the indexing technique. The time complexity is O(log(log(n))) on the average for typical simulation data.

1. Introduction

Performance problems arise when databases are used in data-intensive applications involving time. For example, when databases are used to store time-stamped output data from *e.g.* simulators, the volumes of the data get very large, easily in the order of gigabytes [57][32][58]. We have worked with a prototype of a simulator based on transmission line delays. To simplify the handling of output data from the simulation, an object-relational main memory database management system was embedded into the simulation system. This made it possible to state queries over the simulation output data in terms of the object-oriented model of the simulated system. Unfortunately, the performance of the simulation dropped drastically [7] due to the time spent on storing the output data in the main memory database [16]. It seems that todays database systems can not handle the high rate of updates and large volumes of data from high performance simulators. This paper aims at improving the performance of simulation databases by introducing a new storage structure and indexing method for temporal data.

We have studied a simulator for hydraulic circuits developed at the division of fluid power technology [25]. The simulator is based on transmission line delays [27] as opposed to conventional simulators where all differential equations are solved simultaneously in a centralized solver. The limited signal propagation speed in physical systems is used to partition the simulation into weakly coupled subsystems where the differential equations can be solved locally. This makes it possible to simulate complex systems in reasonable time since the computational effort is approximately linear with the size of the simulated system.

The time granularity of the simulation is not constant. For example, in simulation, the time-step must be adjusted to keep the simulation stable. The simulation process becomes instable whenever the derivates of values in the simulation become to large. A decrease of time-step (Δt) will stabilize the simulation but also increase the number of updates to the database. Typical control applications involving sensors also display varying time granularity due to jitter and cascades of alerts.

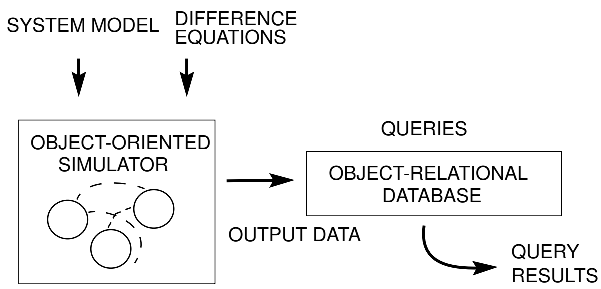

Figure 1. Object-oriented simulator

We have identified the following fundamental characteristics of simulation and sensing applications:

- Because of the nature of simulation and sensing applications, the insertions of new data typically occur with increasing time stamps.
- The time stamped information comes from several sources.
- The number of time stamps is several orders of magnitude larger than the number of data sources.
- The time stamps are generally not equidistant but locally equidistant sections can exist.
- The volume of stored time stamped data will be large, easily several gigabyte.

This work has been supported by NUTEK, the Swedish National Board for Industrial and Technical Development.

- The applications require high performance: 10^4-10^5 updates per second.
- The application will typically operate in two phases: producing the data and analyzing the data. These phases can be interleaved or even be concurrent in interactive simulation.
- Deletion of time stamped data will generally not occur during the analysis phase.
- After the data production phase, the stored information very seldom change. The accesses are predominantly read-only.
- The application may want to undo (roll-back) all updates done after a certain point in time.
- Furthermore, in many applications the volume of data will continuously expand. We will require a mechanism to expand the storage structure and a mechanism to dispose, archive, or compress the oldest items.
- Database recovery in simulation applications can be achieved by restarting or rerunning the simulation. In control applications, the recovery is more complicated. *If* the control system can restart from a blank state or a state with corrupt data the recovery can be handled. Normally this should be true to satisfy the robustness requirement. For the same reasons, the persistence is generally not critical.

These properties and restrictions will be exploited in the proposed storage structure and indexing technique to gain performance over other existing temporal indexing techniques that do not make these assumptions. Many of the properties are also valid for other scientific applications where large volumes of time stamped data is involved. Here, we consider storing time stamped simulation data and the subsequent retrieval of the information where the speed of insertion of new data is of major concern. The requirement for fast insertion leads to the idea of a simple storage structure where we use arrays which can be addressed using direct machine address pointers. A minimum of housekeeping will be done at insert time to speed up the insertion. To solve the problem of fast lookup we propose the use of computational indexes based on super-linear equation solving techniques from the area of numerical analysis.

2. Related work

During the last ten years, work has been done in the area of efficient storage structures and indexing techniques for temporal databases [1] [45] [34] [33] [52] [21] [55]. An analysis of a system which added time to a conventional database showed rapid performance degradation when data volumes increased [50]. Work on temporal databases describe two or more orthogonal time dimensions. *Valid time* denote a time dimension which represent when some stored information is valid. *Transaction time* is when the information was entered into the database [26]. Many of the storage structures are aimed towards representing *valid time* which is more difficult to handle. The early approaches [34][50][2][11] used links from the current version of each individual object to older versions. Accessing earlier versions of an object resulted in traversing a series of these links.

A variety of search trees has been developed to solve specific problems in databases [22][5][46][23]. A survey of search trees is provided by Knuth [30] and B-trees are explained in [9]. A method called *time index* was presented by Elmasri *et. al.*[15]. This access structure is based on B+-tree representation of the valid time intervals and support temporal operators, such as *when, select,* and *temporal join*. The append-only modified multiway B+-tree, AP-tree, was proposed to speed up insertions by avoiding rebalancing and at the same time increase node loading [21]. The same paper [21] propose nested ST-trees which are similar to K-b-trees [31]. In time-polygon indexing, TP-index, time intervals are mapped into two temporal dimensions: time, and duration [48]. The datapoints in the temporal space are then partitioned into polygons. Another approach is based on files with time points rather than time intervals [43]. This storage structure is basically a multidimensional partitioned file where the time is one of the dimensions. The approach supports range queries as well as indexing by other search attributes. A bitemporal indexing method [38] uses a tree of transaction times which points to a forest of incremental valid time trees.

All these storage structures are general methods to represent temporal data. They imply fairly complicated and time consuming insertion operations to support fast retrieval and complex temporal operations. But temporal data management in engineering applications differs from the traditional areas of administration [55] and banking [14]. Simulators and sensing applications do not require bitemporal databases. The performance is more critical and in particular the insertion operations should be fast.

The simplest method to store time stamped data is to use an array of tuples where the first element of each tuple is the time stamp. The speed of insertion into such a structure is near optimal but the retrieval of the data will be slower unless special measures are taken. There are several methods for searching in such array structures when the keys are linearly ordered. Binary search was first mentioned in a paper by J. Mauchly [36] and later generalized and developed by others [3][24][29]. Fibonaccian search was invented by Ferguson [17] and uses only addition and subtraction in the search process. It is thus faster on some computer architectures. Interpolation search was suggested by W.W. Peterson [40].

Key conversion algorithms or algorithmic indexing [35] is described as a method to calculate a physical address from entity identifiers. This approach is particularly useful in situations where records form fully populated matrices. The conversion is done in two steps. First the key is converted into an ordinal number which in the second step is converted to a file address. Calculated guess addressing [35] is based on statistical information on the distribution of the data. For example, when the keys are names that are sorted in alphabetical order, a particular key can be located faster if we use the probabilities for names to begin on certain letters. The poor worst case performance of pure interpolation search is generally solved by using combinations of interpolation, binary search, and sequential search [54][56].

This paper could be viewed as continued work along the lines of key conversion, interpolation search and calculated guess addressing. We describe a storage structure for time series in main memory databases which features very fast insertion. An associated indexing technique based on super-linear equation solving methods is presented. We adapt the results from the area of numerical analysis so they can be used as indexing techniques. The indexing techniques assumes that new data is inserted with monotonically increasing time stamps, which is natural in simulation applications. The indexing technique is therefore useful for transaction time [55] as well as special cases of valid time (e.g. simulated time). We also present some results of performance measurements of the storage structure and the indexing technique.

3. Equation solving, the dual of the indexing problem

Given an array structure with temporal data where the time stamps are ordered but not necessarily equally spaced, we search for the array index closest to a particular key (time point). This process is usually referred to as temporal forward indexing. There is a duality between searching in this sequence of linearly ordered indexes and solving an equation f(x)=0 where the function f(x) is monotonically increasing. This duality allows us to use numerical equation solving techniques as indexing methods.

Let t^* be the time point we are searching for and t(i) an array containing the ordered timepoints. Given t^*, we wish to locate the index of the array that contains the timepoint. In a mathematical sense, t(i) is a discrete relation between an index and a timepoint. The index dimension is the set of integers in the interval $[0,i_{max}]$. The time dimension is a set of ordered discrete time points, denoted t_n. We call t(i) the *time index function*. We illustrate this duality with the graph in figure 2.

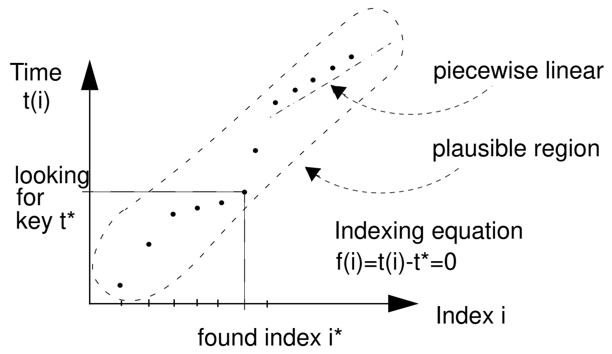

Figure 2. Time index function and index equation

The time index function is monotonically increasing since each new time stamped value is inserted into the array in append mode and the time stamps will have increasing values. There are limitations on the slope of the time index function since the applications generally have a lower limit on the time interval between the values they generate. Given a time stamp the index can be found by crude sequential scanning (linear search) of the timepoints in t(i) or by binary search.

In this paper we propose the use of a more sophisticated method where we numerically solve the dual problem stated by the indexing equation $t(i)-t^*=0$. The solution to the indexing equation will be the index i^* *(see fig. 2)*

First the time-index function is transformed to a *continuous monotonic differentiable function f* such that

$$\forall i\,(i \in Z \wedge 0 \le i \le i_{max} \Rightarrow f(i)=t(i)-t^*) \wedge \forall x (x \in R \Rightarrow f'(x) > 0). \qquad \text{(eq. 1)}$$

If x is a root of continuous function f(x)=0 and x is an integer then x will also be an exact solution to the discrete indexing equation $t(x)-t^*=0$. This follows from mean value theorem for derivates and the monotonicity of f(x). If the root is not an integer then timepoint t^* do not exist in the structure.

There are many ways to handle non-stored timepoints. If the stored data can be assumed to be stepwise constant [21] then $i^*=\lfloor x \rfloor$ will at least satisfy the inequality $t(i^*) \le t^* < t(i^*+1)$. In short, the floor of the root corresponds to the timepoint which is closest to t^* but not later than t^*. It is also possible to interpolate values of non-stored timepoints or to simply regard non-stored timepoints as failed searches. Since we do not require the timepoints to be equidistant, the indexing equation f(x)=0 can be non-linear. We will look at several numeric root finding methods and examine their efficiency as indexing methods in simulation databases.

3.1 Iterative computational indexing

The continuous function f(x) must be constructed before we can use traditional numerical methods to solve the indexing equation. Fortunately the construction of such a function can be avoided by rewriting the root finding algorithms into iterative operators that operate on sets of indexes called index brackets.

Let t(i) be the time index function, and t^* be the timepoint we are trying to locate, the problem of finding the particular index i^* that correspond to t^* can be dealt with in the following way using numerical analysis. The solution to the indexing problem can be regarded as a process of iteratively applying an operator **H** to a search space $\mathbf{B_0}$ consisting of a sequence of linearly ordered indexes. The indexes are isomorphic with the natural numbers. A search space consisting of an interval $\mathbf{B}=[a_n,b_n]$ is called an index bracket *iff* $t(a_n) \le t^* \le t(b_n)$.

The indexing methods we will describe are processes in the form $\mathbf{B_{n+1}}=\mathbf{H}(\mathbf{B_n})$ where the index bracket, $\mathbf{B_n}$, is an interval of indexes $[a_n,b_n]$. By an iterative indexing process we mean a process where the sequence of index brackets $\mathbf{B_0}$, $\mathbf{B_1}$, $\mathbf{B_2}$, ... , $\mathbf{B_n}$ can converge towards unique index i* which satisfies the relationship $\mathbf{B}^*=\mathbf{H}(\mathbf{B}^*)$ where $\mathbf{B}^*$ is the unique bracket $[i^*,i^*]$ that is the fixpoint of **H** that satisfies $t(i^*)=t^*$. The index searched for is i^*.

Several problems are involved in the construction of iterative computational indexing methods.

- the choice of the initial bracket $\mathbf{B_0}$
- the convergence of the sequence
- the rate of the convergence of the sequence
- the conservation of bracketing
- the existence of the index
- the uniqueness of the index
- the choice of the iteration operator **H**

The initial bracket should be selected to be the bracket containing all indexes $[0, i_{max}]$. The operator **H** is said to be convergent if starting from any bracket $[a_n,b_n]$ where $t(a_n) \le t^* < t(b_n)$, the next bracket $H([a_n,b_n])$ tends toward the correct index and conserve bracketing. Formally, the new bracket must be a subset of the previous bracket; $[a_{n+1},b_{n+1}] \subseteq [a_n, b_n]$ that is, $a_{n+1} \ge a_n$ and, $b_{n+1} \le b_n$. The new bracket should contain fewer indexes than the previous bracket $|\mathbf{B_{n+1}}| < |\mathbf{B_n}|$ that is, $b_{n+1}-a_{n+1}<b_n-a_n$ and the new bracket should contain the correct index: $t(a_{n+1}) <= t^* < t(b_{n+1})$. The operator **H** must admit at least one fixpoint B^* for each legal initial bracket, $\mathbf{B}^*=\mathbf{H}(\mathbf{B}^*)$.

3.2 Iterative operators

A variety of methods has been developed for the solution of non-linear equations. We will restrict ourselves to methods which can be adapted to robust bracket-based searching such as the bisection method [36]. The bisection algorithm evaluates the function at the midpoint of the current bracket and examine the sign. It replace whichever bracket limit has the same sign. After each iteration the size of the bracketed interval containing the root will be decreased by a factor of two. After n iterations with the bisection method the size of the interval will be:

$$|B_n| = \frac{1}{2}|B_{n-1}|^{\varphi} = \frac{1}{2^n}|B_0|^{\varphi} \quad \text{where } \varphi=1 \qquad \text{(eq. 2)}$$

In the terminology of numerical analysis the bisection method has linear convergence (φ=1). The corresponding time complexity of this form of search, *O(log(*$|B_n|$*))*, can easily be derived from the convergence relation above [35].

Linear search also known as sequential scanning [35], bisection also known as binary search [30], regula falsi also known as interpolation search [19][40][35], are already used in computer science. We will instead focus on recently developed super-linear methods with higher asymptotic order of convergence (φ >1.0) [19][13][10][12][6][14][42].

For smooth functions, these methods converge faster than the bisection search and other algorithms with linear convergence, but for patological shapes of functions, they can converge *slower.* For example, regula falsi and the secant method are slower than the bisection method if the second derivate of the functions changes abruptly close to the root [41]. Some of the methods with higher asymptotic order of convergence are not even robust [13]. The algorithms can diverge rapidly and the search fails. We will now study a class of robust algorithms known as interval algorithms. These algorithms can easily be used as iterative operators in indexing processes.

Ridders method [42] is based on linearizing the original function by multiplying it with an exponential function. Standard regula falsi is then applied to this modified function. This method always conserve bracketing and has in theory at least quadratic convergence (φ > 2.0). But the method evaluate the function twice which in practice lowers the convergence to $\varphi=\sqrt{2}$. The method is particularly good at finding roots of non-monotonic functions. Unfortunately the convergence of this method is not guaranteed.

$$i(x) = \frac{(x - f(i_1))(x - f(i_2))\, i_3}{(f(i_3) - f(i_1))(f(i_3) - f(i_2))} + \frac{(x - f(i_2))(x - f(i_3))\, i_1}{(f(i_1) - f(i_2))(f(i_1) - f(i_3))} + \frac{(x - f(i_3))(x - f(i_1))\, i_2}{(f(i_2) - f(i_3))(f(i_2) - f(i_1))}$$

Brents method with Inverse quadratic interpolation

The Van Wijngaard-Dekker-Brent method from 1960 is a well-established algorithm that uses inverse quadratic interpolation to achieve quadratic convergence. The method was later developed by Brent [4] and has in its present form, guaranteed convergence and robustness. While regula falsi and the secant method assume approximately linear behaviour between two prior root estimates, inverse quadratic interpolation (see fig. above) uses the three prior root estimates i_1, i_2, i_3 to form a function i(x). Setting x=0 gives the next estimate for the root of f(x)=0. The method has very high convergence for smooth functions. The number of significant numbers in the root *doubles* for each iteration.

Although regula falsi and similar methods work satisfactorily in most practical cases, none of them guarantees convergence in less than *O(n)*, whereas Brent's method converge within about $O(log(n)^2)$.

```
Operator:    VAN WIJNGAARDEN-DEKKER-BRENT
Input:       i1,i2,i3 (prev. estimates)
Output:      i1,i2,i3

if i2 and i3 has same sign then
   i3=i1 and d:=i2-i1 and e:=d
if |f(i3)| < |f(i2)| then
   i1=i2 and i2=i3 and i3=i1
Let imid=0.5*(i3-i2)
s:= f(i2)/f(i1)
if i1=i3
   then  p:=2.0*imid*s and q:=1.0-s
   else
      q:=f(i1)/f(i3) and
      r:=f(i2)/f(i3) and
      p:=s*(2.0*imid*q*(q-r)-(i2-i1)*(r-1.0))
      q:=(q-1.0)*(r-1.0)*(s-1.0)
if p is positive then q:=-q
e:=d
d:=p/q
i1=i2
i2= |i2+d|
```

Figure 3. Simplified pseudo-code for an operator based on the Brent algorithm

The algorithm assumes that the interval $[i_2,i_3]$ brackets the root and that $|f(i_2)| \leq |f(i_3)|$ which is true for monotonic functions. i_1 may coincide with i_2. Throughout the algorithm, i_2 is the best estimate. When $f(i_2)=0$, the algorithm has located the root (i_2). The algorithm fails if $|2.0*p| > \min(3.0*i_{mid}*q, |e*q|)$ since the interpolating parabola has a limited possibilities to approximate functions. The algorithm has low convergence if $|e| < 1$ or $|f(i_1)| > |f(i_2)|$. When q approaches zero the inverse quadratic interpolation can give very bad estimates and there is a risk for division with zero.

$$q = \left(\frac{f(i_1)}{f(i_3)} - 1\right)\left(\frac{f(i_2)}{f(i_3)} - 1\right)\left(\frac{f(i_2)}{f(i_1)} - 1\right) \approx 0$$

This situation is highly unlikely since it only occurs if the time interval between any two of the timepoint estimates $f(i_1)$, $f(i_2)$, $f(i_3)$ is small, i.e. the timepoint is located in a local burst of data. Since the Brent operator is only applied in very large search spaces (see adaptive algorithm below), the burst of data would have to be very large. Brents method guards against this problem by checking where the interpolation would land before carrying out the critical division. The implementation of iterative super-linear indexing operators require attention to round-off errors, convergence, and use of register variables for previous estimates. The reader is referred to “Numerical recipes in C” [41] for a detailed algorithm.

4. Adaptive computational indexing method

Complex methods with high order of converge *can* be slower than binary search. The speed of the presented methods depend on the size of the search space and the shape of the time index function. This is why we propose an adaptive search method.

The adaptive indexing method is not a single indexing algorithm, but rather a set of strategies that can be applied depending on information about the stored data. The key is that all bracket based algorithms can be mixed in a flexible way. If one method fails to prune the search space, another method can be applied for the same bracket. The idea is to adapt the convergence rate φ according to the specific constant K and the size of the current bracket $|\mathbf{B_{n+1}}| = \mathbf{K}\, |\mathbf{B_n}|^{\varphi}$. The extended Brent's algorithm uses a similar idea, but it does not take into account the size of the search space.

There is an actual increase in computational effort when using the inverse quadratic interpolation which must be justified by a faster convergence. Furthermore, the Brent's method must be adapted for search in a discrete domain. It is not meaningful to find the timepoint at index 1,34. As long as bracketing is guaranteed we can use integers for index arithmetics.

An example of an adaptive operator is the following combination of regula falsi, Brents method and bisection search:

$$\mathbf{H(B)} = \begin{cases} \textit{regula falsi} \text{ on B if B is near-linear} \\ \textit{brent on} \text{ B if super-linear convergence} \\ \text{possible and the search space is large} \\ \text{otherwise, } \textit{bisection} \text{ on B} \end{cases}$$

This iterative operator will have the following properties:

- Super-linear convergence for smooth time index functions. *O(log(log(n)))*.
- Convergence is guaranteed to never be slower than *O(log(n))*
- Only one cell reference per iteration
- Constant space algorithm

The time index function is evaluated at the midpoint of the current bracket. If the search space is small we use bisection. If the function seems to be linear or near-linear based on this information then regula falsi is attempted. If the time index function is non-linear we try Brent's method. If Brent's method has low convergence or fails we use the basic bisection operation. Note that the bracket mid point is only calculated and evaluated once (one cell reference). The halt test is when the bracket contains only one index.

4.1 A cost-benefit model for iterative operators

The definition of a cost-benefit model is a general way to accomplish operator selection in the indexing process. The cost of each iterative operator is expressed in terms of the current bracket size and previous estimates and timepoints. The benefit is defined as the expected relative reduction in search space. To be efficient in a practical system, the cost-benefit model must be tightly integrated with the code of the indexing algorithm. With a cost-benefit model a variety of specialized iterative operators could be defined to handle patological cases in an efficient manner.

5. Implementational issues

The current implementation of the storage structure and the computational indexing method consist of approximately five hundred lines of C-code. Special attention had to be given to the coding the of super-linear algorithms in order to conserve bracketing in all possible situations.

We have regarded the storage structure as an array to timepoints and values but to be useful in a real main memory database some additional aspects must be considered: The storage structure should support dynamic allocation and growth. Furthermore, many applications will continuously insert new time stamped data in the storage structure. Eventually the system will require a mechanism to dispose, archive, or compress the oldest items in the database.

This temporal purge mechanism should remove all data with time stamp less than t_{limit} and leave room for new data. The simulation applications may also want to undo (roll-back) all updates done after a certain point in time. This situation typically occurs when a simulation fails or gets instable and the simulation must restarted from an earlier time point. There is no need to save data from the instable part of the simulation.

The dynamical growth, temporal purge operation and temporal rollback can easily be accomplished by maintaining the time stamps in several arrays organized in a circular buffer structure as in fig 5. The capacity of the storage structure is expanded by allocating a new fixed size array and inserting a reference to it in the circular buffer. The temporal purge and temporal rollback is accomplished through manipulation of three pointers. There is no overhead associated with this structure at insertion time but some extra address calculations must be performed when an element is accessed. An array index is treated as a virtual index which is translated into a base address of a fixed size array and an offset within the array.

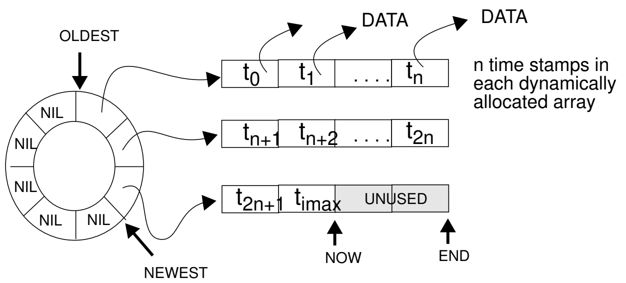

Figure 4. Dynamical time index arrays

There is more to a temporal database than just storage structures. To integrate the new access structure in an object-relational main memory database [16] we need to support logging, reference counting, types and boxing of data items. To keep the insertion operation fast we prefer to delay all overhead until access time. The storage structure will store literal "dirty" data which will not reference counted until it is accessed.

When data is accessed, the information will be temporarily boxed which allows type inference and locking. This is preferable when the file-activity ratio [35] is low.

To integrate the computational indexes with a query optimizer, the optimizer must be given access to information on the cost of accessing data in the storage structure and the cost of boxing the data, reference counting, locking and adding type information.

6. Performance evaluation

We measured three aspects of the performance of the storage structure: insertion, deletion and search. The tests were performed on a Sun SPARC-Center workstation cluster with 2.5 GB virtual memory. We will refer to the reading of a timepoint of a particular index as one cell reference [49] and we wish to measure the average number of cell references that will occur when searching for an index. In main memory, this efficiency measure is rather accurate, whereas in disk based systems the cell references require different amounts of time. The duration of a cell reference depend on if a new disk seek or a track reading must be issued or if the data is already in main memory.

There are several factors involved in the performance of indexing methods. The database was populated using a number of simulated sensors(1-20000 sensors). Each sensor generated independent sequences of time stamped values using uniform and non-uniform distributions for the time interval between generated values. The number of time stamped tuples was varied between 200k to 1000 M tuples. By varying the distributions we affected the slope (average frequency of generated data) and the smoothness of the time index function (variations in the frequency of generated data). For a wide range of parameter settings, the generated time series empirically confirmed the existence of a plausible region (Fig. 2).

It was decided that index compression should be ignored initially, since existing algorithms for compression would make the indexes non-monotonic. The speed of insertion into the temporal storage structure was as expected very high: more than three million time stamped values per second. The insertion routine written i C was compiled into 23 (Sun SPARC) machine instructions but this number could be further reduced if the insertion routine was hand-coded (10 machine instructions). This is more than four thousand times faster than using standard hashtables in the main memory database [7]. Note that the array based storage structure do not have the overhead associated with reference counting and the type system at insertion time.

Also logging of insertion operations can be avoided since persistence is achieved by restarting the simulation. Deletion in the forms of rollback and temporal purge had very high performance since these operations only involve changing a set of pointers.

We also measured the performance of traditional search techniques in order to compare them to the adaptive super-linear indexing techniques. Figure 6 shows the result of the measurements.

For large data sets the super-linear methods used fewer cell references than binary search. The access time when using the adaptive iterative indexing operators was constant for all practical sizes of search spaces. This is confirmed by theoretical analysis of the convergence of super-linear methods for simulation data. If cpu-time usage where to be used instead of cell reference count, the algorithms would still display the same performance behaviour but with a different scale.

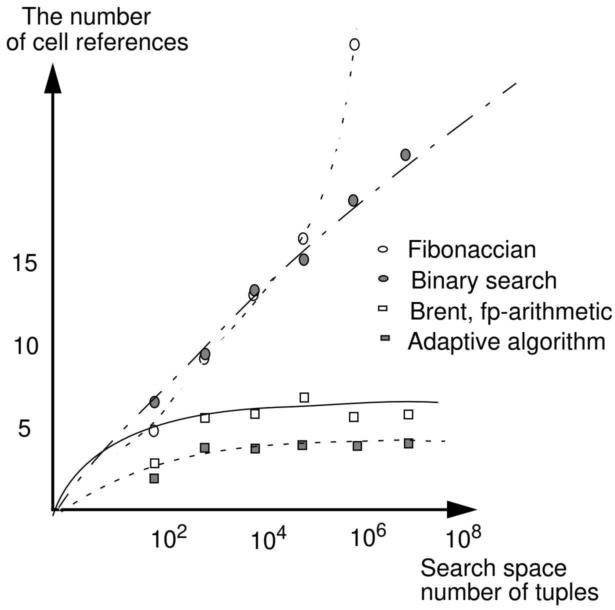

Figure 5. Number of cell references of indexing methods for typical simulation data

By estimating the number of memory operations and arithmetic operations required by a super-linear algorithm, we found that the computational effort is about 4-5 times more than the traditional binary search. Note that the adaptive algorithm only uses the super-linear methods while the search space is large. This results in a break-even point which is in the vicinity of 10^6 - 10^7 time stamped values depending on the implementation of the algorithms and the shape of the time index function.

7. Discussion

Interpolation search is not faster than binary search according to [30]. This is true in the general case when we have no information about the time distribution. But in simulation and sensing applications we do have some knowledge of the shape of the time index functions and we can take advantage of this information. Typical time index functions adhere to a plausible region (Fig. 2) due to the central limit theorem.

Simulation applications can during a time interval generate equidistant point. The time index function will then consists of piecewise linear sections of equidistant points (Fig. 2). These linear sections can be encoded by representing the end-points of the linear section and the time increment in that equidistant section. The search space would then be reduced to the end-points of all linear sections and the searching would be done in two phases: locating the correct linear section and then performing regula falsi in that section. The speed-up will not be large since the adaptive algorithm will attempt linear interpolation as soon as the current bracket is within a piecewise linear sequence of timepoints. This explains why the adaptive algorithm outperforms the pure implementation of Brent's algorithm.

The iterative indexing methods exclude the use of existing lossy key compression techniques as used in R-trees [22]. The indexing methods require key compression that produce exact keys when decompressed. An alternative approach is to use compressed key in the indexing method. The indexing methods can work on compressed indexes the keys retain their monotonicity when they are compressed.

This paper presented the search methods as a specialized temporal indexing methods, but they are equally useful in other applications where the domain of the keys has linear ordering and new data arrive with keys in accordance with the ordering. Examples of other applications are control applications and logging of database events.

The iterative indexing methods described in this paper are not very efficient when used in disk based systems since each cell reference has a high probability of resulting in a disk seek operation. To adapt the computational indexes for disk based systems, we must reduce the number of pages read from disk. One way to accomplish this is to record previously accessed timepoints and indexes and then incrementally synthesize an algebraic computational index. Since the time index function is monotonic, we can use the earliest (first) timepoint in all previously accessed disk pages to build an monotonic algebraic function that approximately correspond to the *inverse* of the time index function with a certain accuracy (e.g. page size on disk).

8. Summary

The numerical solution of non-linear equations and equation systems is a well-established area, in which a variety of methods has been developed. We have shown how equation solving methods can be used to compute indexes in simulation data. One of these methods is bisection search, which in computer science is known as binary search. In our work, we have tried and tested other new numerical techniques as indexing methods. In particular we have focused on robust quasi-Newton methods designed for efficient numerical solution of non-linear equations in one dimension. These methods make it possible to avoid overhead at insertion time which is particularly good if the file-activity ratio is low. We have developed and implemented an adaptive iterative indexing method based on bisection, regula falsi, and Brents methods. The advantages are:

- Practical efficiency.
- Quadratic convergence: time complexity is *O(log(log(size)))* in practical applications. The number of cell references is fewer (or equal in the worst case) than in binary search. The mathematical analysis of convergence in non-linear iterative algorithms does not generally take account of the practical efficiency [39]. Normally good iterative algorithms converge far better than existing theorems can prove, which is to our advantage.
- Near optimal performance of insertion, which is of major importance in simulation.
- Efficient usage of storage. No additional structures needed.

This approach is only efficient for main memory data structures. A disk based system depend on the locality of accesses. Another limitation is that insertion is only possible with monotonically increasing time stamps, and only the newest and the oldest data can be deleted.

We have also proposed the use of inverse interpolation as an *algebraic* computational indexing method. This type of computional indexes are also applicable in disk based systems.

8.1 Future work

The paper only deals with the efficiency of the storage structures. If we can combine this with knowledge of data access patterns we could get even higher performance. For example, aggregated data could be pre-calculated [32][28]. This would avoid repetitive aggregate computations at access time. Further research in the incremental synthesis of algebraic indexes would be highly interesting. We will study discrete wavelet transform [8] as a compact way to model the time index function.

Acknowledgment

I would like to thank the Swedish National Board for Industrial and Technical Development, NUTEK, for financial support. Many thanks to members of the Division of Fluid Power Technology and the department of Numerical Analysis at Linköping University. The AMOS main memory databases was provided by EDSLAB at Linköping University. Jonas Karlsson and Martin Sköld at EDSLAB has also provided many helpful suggestions.

References

[1] Ahn,I.: *Towards an Implementation of Database Management Systems With Temporal Support*, Proc. IEEE Data Engineering Conference, February 1986

[2] Ben-Zvi,J.: *The time relational model*, PhD thesis, Computer Science Dept. UCLA 1982

[3] Bottenbruch,H.: J *ACM* 9(1962), 214

[4] Brent,R.P.: *Algorithms for Minimization without Derivates*, Englewood Cliffs, NJ: Prentice-Hall 1973

[5] Brinkhoff,T.,Kriegel,H.P.,Schneider: *Multi-Step Processing of Spatial Joins*, Proc. ACM-SIGMOD Intl. Conf. on Management of Data, Minneapolis, 1994

[6] Bus, J.C.P., Dekker, T.J.: *Two efficient algorithms with guaranteed convergence for finding a zero of a function,* ACM Trans. Math. Software. 1(1975) pp. 330-345

[7] Carlsson,E.,Falkenroth,E.T.: *SIAMOS Electrical circuit simulation in AMOS*, RTSLAB Memo 1995, Department of Computer Science, Linkoping University, Sweden

[8] Graps, A.: *An Introduction to Wavelet*, IEEE Computational Science and Engineering 2(2), 1995

[9] Comer,D.: *The Ubiquitous B-tree*, Computing Surveys, 11(2) 1979

[10] Conte, S.D., de Boor, C.J.: *Elementary Numerical Analysis: An algorithmic approach*, 3rd ed, McGraw-Hill, New York

[11] Dadam,P.,Lum,V.,Werner,H.D.: *Integration of time versions into relational database system*, Proc. VLDB, Singpore, August 1984

[12] Dahlqvist, G., Björck, A.:*Numerical Methods*, Prentice-Hall, Englewood Cliffs, N.J.

[13] Dew,P.M., James, K.R.: *Introduction to Numerical Computation in Pascal*, The MacMillan Press Ltd, London 1983

[14] Dreyer, W., Dittrich, A.K., Schmidt, D.: *Research Perspectives for Time Series Management Systems*, SIGMOD Record, 23(1),1994

[15] Elmasri,R.,Wuu,G.T.J.,Kim,Y-J.: *The Time Index: An Access Structure for Temporal Data,* Proceedings of the 16th VLDB Conference, Brisbane, Australia 1990

[16] Fahl, G., Risch, T.,Skold, M.: *An architecture for Active Mediators*, Proc. Intl. workshop on Next Generation Information Technologies and Systems,Haifa, Israel, 1993

[17] Ferguson, D.E.: *C ACM* 3(1960),648

[18] Gadia, S., Yeung, C.: *A Generalized Model for a Temporal Relational Database*, Proc. ACM SIGMOD Conference, June 1988

[19] Gardan,Y.: *Mathematics and CAD, Vol 1: Numerical methods for CAD*, Kogan Page Ltd, Pentonville Road, London N1 9JN

[20] Gunadhi, H., Segev, A.: *Physical design of Temporal Databases*, Dept. Computing Sci. Res. and Development, Lawrence Berkeley Lab., CA, Tech. Rep. LBL-24 578, January 1988

[21] Gunadhi,S.,Segev,A.: *Efficient Indexing Methods for Temporal Relations*, Dept. Computing Sciences Res. and Development,Lawrence Berkeley Lab., Univ. of California, Tech. Rep. LBL-28798, March 1990

[22] Gutman,A.: R-Trees, *A Dynamic Index Structure for Spatial Searching*, Proc. ACM-SIGMOD Intl. Conf. on Management of Data, Boston, June 1984

[23] Hellerstein, J.M., Naughton, J.F., Pfeffer, A.: *Generalized Search Trees for Database Systems*, Proc. VLDB, Zurich, Switzerland, 1995

[24] Iverson,K.E.: *A Programming Language*, Wiley, 1964, pp.141

[25] Jansson, A.:*Fluid Power System Design: A Simulation Approach*, PhD thesis, Dissertation No. 356,Division of Fluid Power Technology, Dept of Mech. Eng., Linkoping University, Sweden

[26] Jensen,C.S.,Clifford,J., Gadia, S.K., Segev, A., Snodgrass, R.T.: *A Glossary of Temporal Database Concepts*, SIGMOD Record, 21(1992)

[27] Johns, P.B., Briens, M.A.:*Use of the transmission-line modelling (t.l.m.) method to solve non-linear lumped networks,* The Radio and Electronic Engineer, Vol. 50 No. 1/2 1980 pp. 59-70.

[28] Kline, N., Snodgrass, R.T.:*Computing Temporal Aggregates*,Proc. Intl. Conf. Data Engineering,Taipei, Taiwan, March 1995

[29] Knuth, D.E.: *C ACM* 6(1963), pp.556-558

[30] Knuth, D.E.: *The art of computer programming, Vol. 3 Sorting and searching* , Addison Westley Publishing, 1973.

[31] Kriegler, H.P.: *Performance Comparison of Index Structures for Multi-Key Retrieval*, Proc. ACM-SIGMOD Conf. Management of Data, May 1984

[32] Lehner, W., Ruf, T., Teschke, M.: *Optimizing Database Access Performance in Scientific Applications Without Compromising Logical Data Independence*, Proc.2nd Intl. Conf. on Applications of Databases, San Jose, USA, 1995

[33] Lomet, D., Salzberg, B.:*Access Methods for Multiversion Data*,Proc. ACM SIGMOD Conference, June 1989

[34] Lum,V., Dadam, P., Erbe, R., Guenauer, J.: *Design DBMS support for the temporal dimension* , Proc. ACM Intl. Conf. Management Data, Boston, June 1984

[35] Martin, J.: *Computer Data-Base Organization*, Prentice-Hall Inc., Englewoor Cliffs, New Jersey

[36] Mauchly, J.:*Theory and Techniques for the Design of Electronic Digital Computers*,3(1946), 22.8-22.9, ed. G.W. Patterson

[37] Mediano, M.R., Casanova, M.A., Dreux, M.: V-Trees - A Storage Method for Long Vector Data, *Proc. VLDB*, Santiago, 1994

[38] Nascimento, M.A., Dunham, M.H., Elmasri, R.: Using Incremental trees for space efficient indexing of bitemporal databases, *Proc. of 2nd Intl. Conf. on Appl. of Databases*, San Jose, CA., Dec. 1994.

[39] Ostrowski, A.M.:*Solution of Equations and Systems of Equations*, 2nd ed., New York Academic Press, 1966

[40] Peterson, W.W.:Accessing for Random Access Storage, *IBM Journal of Research and Development* 1 (1957), pp. 130-146

[41] Press, W.H., Teukolsky, S.A., Wetterling, W.T. , Flannery, B.P.:Numerical recipies in C , The Art of Scientific Computing, 2nd ed., Cambridge University Press

[42] Ridder, C.F.J.:A New Algorithm for Computing a Single Root of a Real Continuous Function, IEEE Transactions on Circuits and Systems, Vol CAS-26, No.11 November 1979.

[43] Rotem, D., Segev, A.:Physical organization of temporal data, Proc IEEE Data Engineering Conference, 1987

[44] Scarborough, J.B.:Numerical Mathematical Analysis, 4th ed., John Hopkins Press 1958

[45] Segev, A., Gunadhi, H.:Event-join optimization in temporal relational databases, Proc. VLDB, August 1989

[46] Sellis,T., Roussopoulos, N., Faloutsos, C.: The R+-Tree: A Dynamic Index for Multi-Dimensional Dynamic Indexes, Proc, VLDB, Brighton, 1989

[47] Seshadri,P., Livny,M.: SEQ: A Model for sequence databases, Proc. Intl. Conf. Data Engineering,Taipei, Taiwan, March 1995

[48] Shen, H., Ooi, B.C., Lu, H.:The TP-Index: A Dynamic and Efficient Indexing Mechanism for Temporal Databases, Proc. Intl. Conf. Data Engineering,Houston, Texas, February 1994

[49] Slazberg, B., Tsotrasm V.J.: A Comparison of Access Methods for Time Evolving Data, Technical Report CATT-TR-94-81, Polytechnique University 1994

[50] Snodgrass, R.;Performance Evaluation of a Temporal Database Management System,Proc. ACM SIGMOD, Conf Management of Data, May 1987 pp.96-107.

[51] Snodgrass,R.: The Temporal Query Language TQUEL, ACM TODS, 12(2), June 1987

[52] Stonebraker, M.,Kolovson,C.: Indexing Technique for Historical Databases , Proc. IEEE Data Engineering Conference, February 1989

[53] Stonebraker, M.,Ahn,I.: Partitioned Storage for Temporal Databases, Information Systems, 13(4),1988

[54] Svensson, P., Andersson, M.: A Study of modified interpolation search in compressed, fully transposed, ordered files, Proc. 4th Intl. Working Conf. on Statistical and Scientific Database Management, Rome, Italy, June 1988.

[55] Tansel,A. et.al. (eds.):Temporal Databases: Theory, Design, and ImplementationDatabase Systems and Applications Series,Benjamin/Cummings Publ. Co., Redwood City, CA., USA

[56] Willard, D.E., S urprisingly efficient search algorithm for non-uniformly generated files, 21st Conf. on Communication, Control, and Computing, 1983.

[57] Wolniewicz, R., Graefe, G.:Algebraic optimization of Computations over Scientific Databases, Proc. VLDB, Dublin, Ireland, 1993

[58] Zemankova, M., Ioannidis, Y.E.:Scientific Databases - State of the art and Future Directions,Proc. VLDB, Santiago, Chile 1994