3 . Management, analysis and integration

Real time and spatiotemporal data indexing for sensor based databases

S. Servigne,
LIRIS: Research Center for Images and Information Systems, INSA, Bât. Blaise Pascal, 20 avenue Einstein, 69621 Villeurbanne Cedex, France

G. Noel,
LIRIS: Research Center for Images and Information Systems, INSA, Bât. Blaise Pascal, 20 avenue Einstein, 69621 Villeurbanne Cedex, France

ABSTRACT: The most important aspect of a system for emergency response is time. So it is necessary to organise fast storage of newly arriving data into databases, fast search of data, maintenance of time sequences, robustness of the systems. All of these processes have to be real time. Another important aspect for environmental risk monitoring for example, is to allow real-time queries based on spatiotemporal attributes. Fastest accesses to the most recent data are also necessary. In addition, real-time data collected into real time database from sensors require fast and efficient management to avoid main memory saturation. The paper discusses real-time spatiotemporal indexing schemes and real time management of main memory. Some possible solutions for real time spatiotemporal data indexing are presented to allow real time and fast data structuring. The proposed index methods support queries privileging recent data. One of the presented indexing solutions is dedicated to real time spatiotemporal data issued from fixed sensors. Another one is dedicated to real time spatiotemporal data collected from agile sensors. Finally, an outline solution concerning management of real time memory saturation according to the significance of data is presented.

1 INTRODUCTION.

Sensor networks are now common and usually used for many applications especially concerning risk monitoring. The monitoring of natural phenomena is indeed a key element of any situation which can provoke some disasters, such as flooding, volcanic eruptions, landslides and so on. Sensor network often share the same global architecture, as defined in Figure 1. Different sensors measure various data. From here on, they have to transmit these data to a monitoring system for processing. The transmission policies must take into account the nature of the sensors, their autonomy and their processing and storage capacities. A balance between energy saving and data accuracy must be reached.

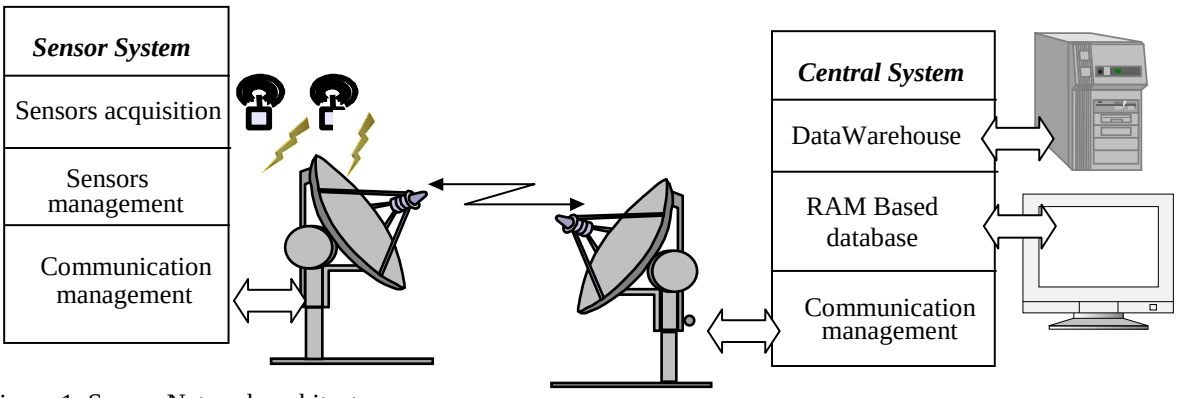


Figure 1. Sensor Network architecture

When data reach the monitoring system, data first go through a main-memory sub-system (database,...) for short-term analysis. Main-memory is faster than disk-based memory and therefore allows faster processing of recent data. This is particularly interesting when the monitoring system is linked to a risk management system. After the main-memory, data can be stored into a secondary memory (disk-based memory, a data warehouse...) for longer term-analysis.

All of this means that the system architecture may change according to various elements. As a matter of fact, the type of sensors used may change the network structure. Data location (in the network, in a database...) has an impact on data querying. The use of a database means that data have to be correctly indexed...

First, an example of a risk monitoring system for environmental phenomena is presented to better explain the need of real time querying and structuring with spatiotemporal criteria. Real time spatiotemporal query requires specific data indexing method. After a state of the art on data structuring according to real time and spatiotemporal specifications, some possible solutions for real time spatiotemporal data indexing are presented.

2 A NATURAL PHENOMENON MONITORING SYSTEM

After a presentation of the general system architecture, a focus is made on an existing system: a volcanic activity monitoring system. Data location and data updating in sensor networks, and also data storage are then tackled. Systems have to offer various and new real time queries for natural phenomenon monitoring. These issues are finally overviewed.

2.1 *General architecture*

Different monitoring systems are used to collect and process data issued by sensor networks. Among these, the Earthworm system, developed by the USGS (USGS, 2006) appears as a good example. It was designed to replace older systems for seismic activity notification. While the first version of the specifications did not take into account the storage and use of older data, the next ones tried to meet these new requirements. The main project split up into two parts: the real-time notification (Automatic Earthworm) and the longer term processing of data (Interactive Earthworm). Different databases were used for the second part.

The Automatic Earthworm is a message passing system. The different components broadcast announces labelled according to the kind of message sent and the source of the message. That way, the different listening systems can choose to pick the relevant messages. It must be noted that even thought the system has been defined as passing message; it can be implemented on a single computer. In that case, a part of the memory can be shared among different processing modules.

The Interactive Earthworm uses a database for storing older data. That is, data that have exited the Automatic Earthworm system and data that may be eligible for medium to long term

analysis. Different systems can access those data through querying or using specific messages. However, even though the specifications were set for the Earthworm architectures, different variations have been developed for specific activities.

As a whole, the Earthworm system was primarily created for notification purpose but has been improved so as to add long term analysis capacities, thanks to a database. This system has been used in many natural phenomenon monitoring systems, including volcanic activity monitoring systems.

2.2 *Instance of system: a volcanic activity monitoring system*

Data collected by the monitoring network of Popocatépetl volcano (seen in Figure 2. Puebla, Mexico) is heterogeneous, coming from various types of sensors. While the sensors are mainly spatially fixed, it should be noted that some data-processing could generate aggregates, which can be regarded as *agile* or even mobile data (epicentre locations). A sensor is said *agile* if it has the capacity of change of spatial location between two data acquisition phase from a specific sensor. A sensor that continually changes it location is labelled as mobile. A sensor that changes sometimes its location is considered as agile.

Developments in sensor technologies have lead to the use of agile or even mobile sensors to obtain data on specific geographical sectors. However, even though sensors are able to move, measurements are usually considered as more important as location changes.

As for now, collected data are indexed through their timestamps and sensor identifiers. Following the Earthworm specifications, data are first sent to an in-memory turnpike system for current data notification, pre-processing and aggregation and toward a database for short to medium-term analysis. For longer-term data analysis, the database sends the measurements toward a data warehouse. We shall focus on the database, with the idea that data can later on be sent toward a data warehouse, possibly using a compatible indexing system.

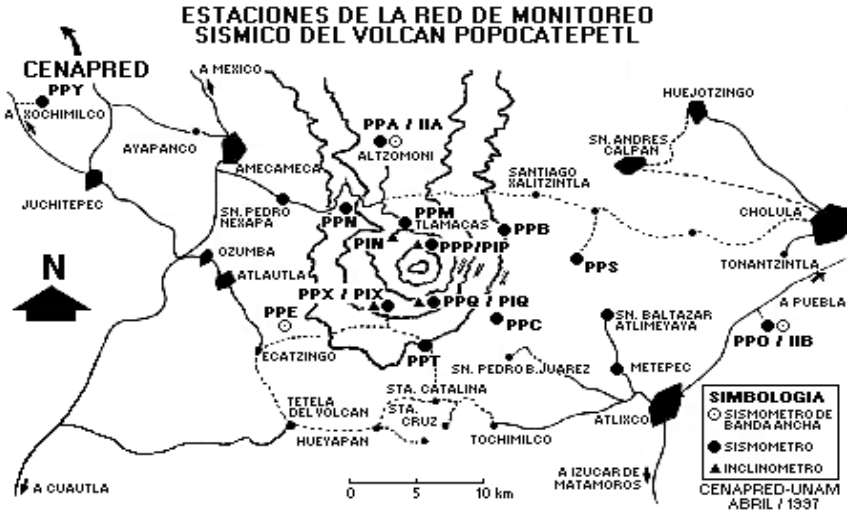


Figure 2. Measuring stations around the Popocatépetl volcano

2.3 *Data location and data updating in sensor networks*

Due to the potentially high number of sensors and their calculation, storage, and transmission limits, data location and updating schemes can be impacted. In networks constituted with a majority of micro sensors, the continuous transmission of data is usually unrealistic, due to the transmission cost. On this matter, different studies tend to agree on the fact that before transmission, data aggregation at sensor level is advisable, despite the scanty calculation capabilities of the sensors (Intanagonwiwat, 2001) (Ratnasamy, 2002). However, aggregating data leads to temporal delays between data measurement and data availability for the users.

For the same reasons of data cost, several studies focused on data location. Three main approaches can be defined (Eiman, 2003).

The first one, the *local approach*, aims at storing data at the local sensor level. Thus, it minimizes updating costs while increasing querying cost. Furthermore, so as to keep tracks of past records, it is necessary to send data to a storage point.

The second approach, the *external approach*, often found in some Geographical Information Systems, uses storage devices (databases, data warehouses) that are outside of the sensor network. So, updating cost is obviously higher but querying cost is noticeably lower. Currently, most decision support systems and phenomenon monitoring systems rely on such solutions, combining the benefits of an in-memory storage system for the most recent data and a data warehouse for older data. Moreover, this approach offers a relative safety against sensors failures (Satnam, 2001). If a sensor fails suddenly, during a critical period, the data previously collected by this sensor, transmitted to the centralized database, are still available for the users.

Various approaches coexist between these two extremes, proposing solutions for data replication between specific nodes of the network. However, these schemes also have their own constraints. Data-centric storage patterns are used to spatially determine specific nodes in the network. Therefore, data can be accessed from these replicates. By distributing data between different network nodes, it is possible to achieve a compromise between updating / querying costs. However, sensor mobility can lead data relocating problems, limiting the real-time efficiency of such techniques.

We can conclude that natural phenomenon monitoring, in which the number of sensor is limited and the environment can turn out to be particularly hostile, should focus on external approaches, at least until better communication protocols arise. External approaches limit the risk of loss data due to sensor failure, a common problem for some natural phenomenon monitoring systems. However, the problem of data storage is not completely solved yet.

2.4 *Data storage*

As stated earlier, the key idea is to store the maximum of information into main memory for the most recent data, and to flush ancient data into an archive (disk, data warehouse).This does not explain how, nor where data should be kept. Three storage levels should be noted (Laurini 2005).

Data are kept, at least for some time at the sensor level. It would be too expensive to directly send data to be indexed. Therefore most sensors aggregate data before a transmission. This means that most recent data, unfortunately unavailable to the analysis system are kept at the sensor level. Some recent systems tend to favour this kind of systems, even for keeping older data.

Current analysis systems rely on in-memory data. Therefore data must be sent toward a central point, where it is kept in main-memory (directly or in main-memory databases). Main-memory is faster than dissk and is interesting for short term analysis. For longer term analysis, data are sent toward disks or secondary-memory (data warehouses...).

Anyway, some common problems must be solved. If data are kept in a database, they must be indexed correctly, taking into account the mass of data issued from a sensor network and their real-time specificities.

Even then, another problem arises. As a matter of fact, when the memory is filled, memory saturation may bring the whole system down. Therefore solutions must be found to free database memory without loosing data. A specific real-time spatiotemporal database management must be coupled with a specific archiving management (Data Warehouse), as seen in Figure 3.

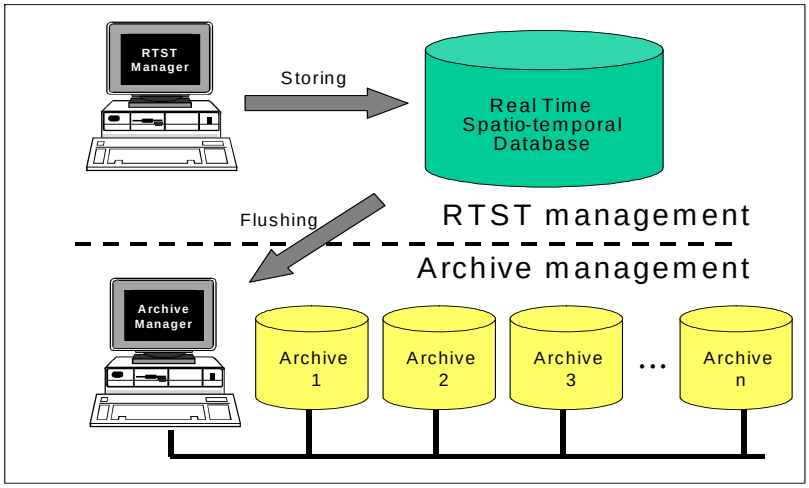


Figure 3. Structure of a real time system for geographic information with regular flushing of data into archive (RTST meaning real time spatiotemporal).

2.5 *Queries*

One point must be noted however. As a matter of fact, even though spatiotemporal data are definitely useful in monitoring systems, most actual implementations are sensor-identifier-based systems.

While current indexing patterns use timestamps and sensor identifiers, specialists' needs for queries based also on spatial attributes are increasing (Figure 4). Querying patterns usually focus on finding data from a specific sensor within a given temporal interval (ending at the present time). When the number of sensors increases, users often fetch data from specific reference sensors so as to estimate the global state of the monitored system before querying other more specific sensors to refine their estimation. Other kinds of queries are possible but less frequent.

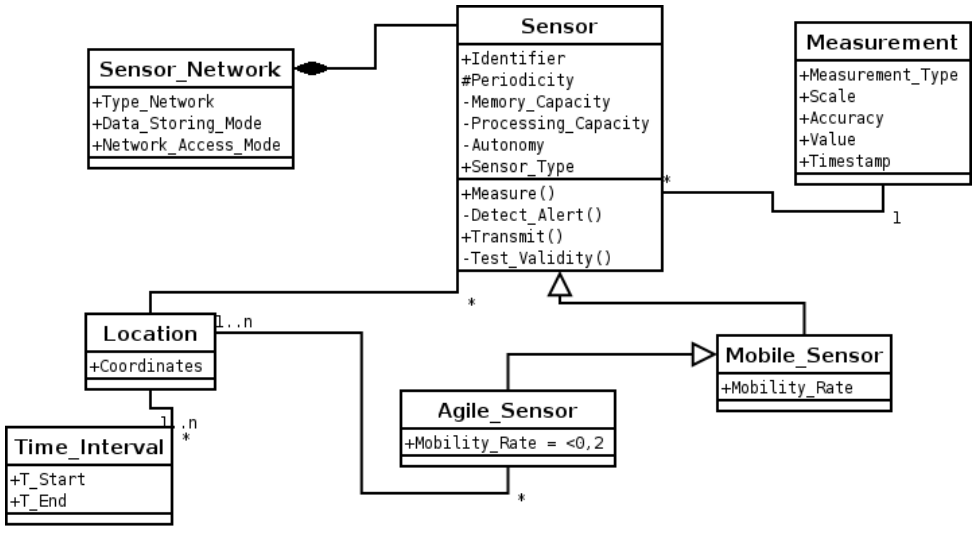


Figure 4. Sensor network UML data model

Sample common queries could be: “Fetch the data issued from sensor number ‘IIB’ in the last 5 minutes.” “Fetch the data from the sensors 'IPP' between the time T1 and T2.”

Due to the impact of GPS technologies and alike, specialists now wish for systems that can answer queries based on spatial components. **Therefore, future systems shall meet real-time and spatiotemporal requirements.** For example, scientists would like to add spatiotemporal constraints to answer queries such as “Fetch the data issued from the sensor at point <X, Y> issued in the last 5 minutes.”

The Popocatépetl monitoring system shows specificities that can be extended to other monitoring systems. A significant number of updates are sent from a sensor network toward a centralized in-memory database. Because of the number of sensors and their measurement frequencies, soft real-time constraints emerge. Measurements from a sensor at timestamp Ti must be committed, copied into the database, before a measurement with a timestamp Tj>Ti reaches the database to be processed.

For the same reason, under normal volcanic activity, update queries must be considered as more important than other lookup queries. While the other processes using the collected data run as background tasks, the updates must be carried out with higher priority.

The focus on the most recent data over the older data should also be noted. In other words, volcanologists are more interested by the current state of the volcano than by its former states (provided these ones are not typical of a significant activity phase).

Last specificity to be noted: the need to refer to sensor through spatial positions as well as through their identifiers. This shall lead to a finer management of sensor agility and an addition of new querying patterns while preserving older ones.

An access method for monitoring systems should meet requirements. Therefore, we focus on real-time indexing methods for spatiotemporal data into real-time main-memory databases to allow fast accesses to real-time data. After an overview concerning spatiotemporal indexes, two new methods for real time databases are detailed. Finally, a solution for real time memory saturation is also presented.

3 OVERVIEW OF REAL TIME SPATIOTEMPORAL DATA INDEXING METHODS

A state of the art on various existing methods is essential to better understand the pros and the cons and the main concepts of the situation.

3.1 *Soft real-time*

Regarding the real-time approach, the main idea is to answer queries within time constraints (Noel, 2004). It is possible to separate three kinds of constraints (Lam, 2001). The soft one, used in case of volcanic monitoring for instance, implies that transactions should be fulfilled within time limits, yet it is understandable that some transaction can not comply within the limits. The firm constraints, more restrictive, allow some transaction not to be fulfilled within the time limits, yet in this case the whole system can be slightly impaired. The hard constraints, finally impose that under no circumstances a transaction should miss a deadline. Otherwise, the system could come to a halt. Priorities are generally used to define which transaction is more important than another; which is not equivalent to define which one should occur before another. Different techniques can be used so as to assign priorities: Earliest Deadline First, Rate Monotonic and other variants (Lam, 2001).

Real-time computing is not similar to Fast-computing. Fast-computing does not prevent a low priority transaction to block high priority transactions (priority inversion) because they have already locked the access to some resources, data. Moreover, for databases, the current paradigm is to keep the index and even the whole base in main memory so as to reduce the number of slow disk accesses. Index Consistency Control (ICC) methods can then be used to make sure no priority inversion occurs while accessing the index (Haritsa, 2001).

Computers processing power increase regularly. That is a well known fact. Another well known fact is that memory cost drops regularly as well. However, what few people notice that memory access cost does not decrease in with processors improvements. Real-Time meant diskless data access for more than 20 years. However, since the late 90's researchers have reached the conclusion that this was not enough. Main memory data accesses are now

considered as a bottleneck. Some researchers have provided models for a better management of memory buffers. While classical indexing structures such as the B+tree were ill-considered in the early 90's, assessments of performance have shown that the processing capacity improvements could give them a new life in the Real-time community. At least if some changes were made to their initial structure. As such, the CSB+tree (Rao, 2000), a cache-conscious B-tree limits the number of pointers in the structure, optimizes the node length so as to fit the memory line length... It has proven a real improvement over older Real-time structures that were meant to outlast the B-trees. Some researchers are now working on extensions of the ideas set up for the CSB+tree so as to provide other indexing systems.

3.2 *Spatial indexing*

The classical spatial approaches in indexing often tend to linearise data so as to use known "fast" structures. Such is the case for quadtrees, kd-trees (Samet 1984; Ooi, 1997) or other methods for spatial objects, or more accurately spatial points. Kd-trees are related to binary trees. A reference point is taken, along with a reference dimension. Every other point that falls below the reference point for this dimension shall branch to the left, all points with higher values branch to the right. At the next level, a new reference point is taken in each branch and the next dimension is used as a reference. It is relatively fast, can be updated on-line but the final shape of the tree depends on the insertion order of data, which can lead to unbalanced trees.

Another widely accepted approach is to use rectangles, bounding structures to match the location of objects. The bounding rectangles can then be regrouped within bigger rectangles so as to create a balanced tree. The R-tree, and its sibling R*-tree (Ooi, 1997) are examples of this. While the R-trees allow to work with complex objects (approximated as rectangles and not points), their higher building and querying time make the use of lighter structures appealing to index points.

When it comes to Real-time concerns, few spatial indices remain. A special version of the R-tree, named CR-tree (Kim, 2001) uses compression to limit the node size in memory, so as to make it fit in a buffer line. Some works now try to mix this approach with the patterns developed for the CSB+tree so as to further reduce the memory access cost to spatial data.

3.3 *Temporal indexing*

Regarding temporal approach, it is important to note that different notions of time can be used for databases (Ooi, 1997). The Transaction Time allows users to perform "rollbacks" so as to find past-values. It does not allow to modify previously entered values, or to enter future values. One can only append new data issued at the present moment. The Valid Time represents the time when a fact is considered true. It allows users to modify past data, and to enter future data. However, it does not allow rollbacks. The Bi-Temporal Time is a mix between the two others, allowing rollbacks, post-modifications and future updates.

There are mainly two ways of considering temporality. The latter is to consider that time is monotonous (time goes in one direction) and to use B-trees as index structures. An interesting variation of this is to consider that data flows constantly ; therefore it becomes possible to link the root of the tree more closely the last leaf, this leaf containing the most recent data. Such an idea has been developed in AP-trees (Gunahdi, 1993).

The other way of considering temporality is to consider time just as a spatial dimension and to use R-trees, with on one dimension the timestamps and on the other dimension the validity duration.

3.4 *Spatiotemporal indexing*

Spatiotemporal approaches have to face the variety of possible types of data: points, ranges, intervals (Wang, 2000). This leads to a distinction between three families of indexing trees: those that work with objects in continuous movement, those for discrete changes and finally those for continuous changes of movements. Another way of differentiating the families of

index has been brought by (Mockbel, 2003). They have focused on approaches aiming at indexing past locations, present ones and future ones.

Many trees have been developed to answer specific needs. Some trees tend to consider the temporal aspect as yet another spatial dimension, which has led to 3DR-trees (Theodoridis, 1996). However, these trees do not take into account the monotonicity of time and usually need to have a previous knowledge of data to index. Another family of trees makes a difference between spatial and temporal dimensions. In HR-tree (Nascimento, 1998), typical of this case, snapshots of spatial R-trees are linked in a time indexed balanced tree. The main problem of this kind of trees is the size of the tree. While nodes that do not change between two snapshots are shared among the R-trees, only minor changes force to duplicate some of the data. Furthermore, they tend not to be optimal for interval queries.

While most of these approaches focus on the data as a whole, some specific indices focus on particular aspect of system monitoring. As an example, structures such as the SETI (Chakkar, 2003) or SEB-tree (Song, 2003) divide the global space in sub-zones for a zone-based spatiotemporal indexing. Other structures such as the FNR-tree (Frentsos, 2003) or MON-tree (Almeida, 2005) are dedicated to indexing data from an existing road or access network. With such structures, the focus is no longer the data themselves but more conceptual entities that can be used for more efficient querying or monitoring.

Recently, researches have lead to different structures based on mobility. Objects in the physical world can often move. Monitoring the movements of these objects has proved an interesting issue, particularly for systems trying to predict the future location of objects. TPR-trees (Saltenis, 2000) use velocity vectors to estimate the future location of an object or its future expansion. While these structures can be efficient in forecasting locations, they are usually not perfect for keeping tracks of past data. Therefore an analysis based on specific past data can be impaired by a structure otherwise interesting for forecasting.

One aspect to note is the difference between mobility and agility. As a matter of fact, mobility is linked to the constant movement of objects. A car, bus or plane is an accurate example of what a mobility-based indexing structure usually has to deal with. On the other hand, agility is linked to a more restrictive notion. Data sources can move, but usually stay in the same location for long times. A portable measurement station, set up at a location for one week then set up elsewhere qualifies for the definition of agility. Most current monitoring systems rely on some kind of sensor agility. Sensor agility management is usually more important to these systems than mobility management. Even though newer sensors tend to be lighter and therefore more 'mobility-oriented', most systems as still widely based on fixed or agile sensors.

3.5 *Real time indexing and memory saturation management*

The faster development of processor technologies over memory technologies during the 90's has lead to great changes in main-memory and real-time databases. As a matter of fact, it has appeared that memory access should be considered as a major bottleneck to system performances. This has lead to the development of structures based on B+trees, such as the CSB+tree (Rao, 2000) or structures akin to the CSB+tree (Bohannon, 2001) (Raatikka, 2004). These structures focus on limiting the cost of memory access by a better use of memory buffers. In the spatial and spatiotemporal domains, new indexes have also been developed. The CR-tree (Kim, 2001) and other indexes (Sitzmann, 2002) (Min, 2004), based on modified R-trees limit heap of memory to store and to access data through compression algorithm or other processor intensive computations.

Even though these structures prove to be interesting assets for real-time indexing, they do not comply with all the specificities of sensor databases. The CR-trees are based on R-trees. Therefore, they tend to store data without regard to their origin. Once again, resolving a sensor specific query involves taking into account the whole dataset, thus slowing down the whole process. The specificities of real-time, sensor related data indexing are not within the parameters of the actual indexing solutions.

As a conclusion of this part, the need is real for real time spatiotemporal data indexing for fixed sensor database, agile sensor database, and it is also necessary to consider real time memory management. There is no existing method answering to the following specifications:

- storing spatiotemporal data collected in real-time from sensor network,
- real-time and spatiotemporal indexing to allow real time spatiotemporal queries and to facilitate rapid access to recent data
- allowing real time spatiotemporal queries taking into account sensor location (fixed sensor, agile sensor)
- real-time main memory management using data significance (application based) to elect data to store on long period.

In the next paragraph, two new methods of indexing for real time spatiotemporal databases are detailed giving a solution to the first three specifications detailed above. Paragraph 5 will detail a solution integrating the management of main memory saturation.

4 REAL TIME SPATIOTEMPORAL DATA INDEXING

One method, the PoTree is designed for structuring mass of spatiotemporal data collected from fixed sensors. The second method, the PasTree is dedicated to index spatiotemporal data issued from agile sensors.

4.1 *An indexing structure for a "fixed real-time sensor" database*

The following sections describe the PoTree, an indexing method (Noel 2004a). This centralized main-memory database spatiotemporal index can store real-time data issued from a fixed sensors network. It also allows real-time querying while focusing on the most recent data.

4.1.1 *Overview of a solution*

Some ideas can be used from the issues detailed in the previous chapters. First of all, the difference between temporal and spatial data can be used to segment the index tree. Then, a variation of B+-tree, the AP tree, has suggested a direct link between the root of the temporal data and the latest node. All of these ideas have been associated to provide a solution of a real time indexing structure for fixed sensor database, the PoTree.

The PoTree is based on the difference between temporal and spatial data, with a focus given to the latter. This way the notion of information sources is linked to a specific spatial location. It has been devised so as to deal with system like volcanic activity monitoring, which involves a number of different sensors with measurement frequencies going as high as 100 Hz. The spatial aspect is indexed through a Kd-tree, while the temporal aspect uses modified B+-trees (see Figure 5). This combination of structures and more than anything else the modifications of the B+-tree are the major innovations of the structure. As for now, mobility is not managed by the structure. However the specificities of both of these trees allow on-line and batch updates: it is possible to update the structure either on real-time or by using batch files.

The monitoring stations being immobile, this structure does not allow mobile sources of information. This way, every spatial location, akin to spatial object (sensor) is directly linked to a specific temporal tree. Requests shall first determine the spatial nodes concerned and later on determine the temporal nodes.

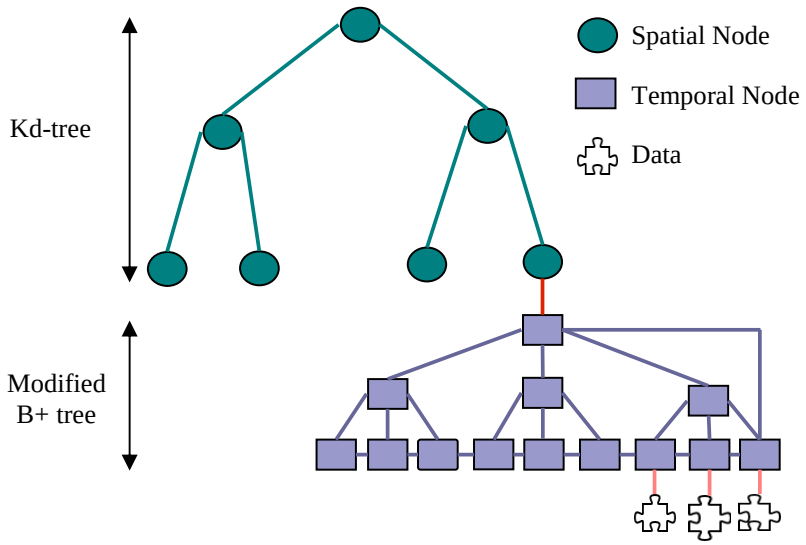


Figure 5. A possible tree structure for a real time spatiotemporal index for fixed sensor database

4.1.2 *Details and discussion*

Kd-trees are simple spatial structures, but they are not perfect structures. One of their main problem being the fact that they rely on the order of inserted data. If data are entered in different orders, the final trees may have different shapes. Another issue is the fact that they are not perfectly suited for mobility, which is not part of our needs in this section. However, ICC methods, originally designed for B-trees can easily be adapted to cover Kd-trees. Different tests have also proven that these trees behaved reasonably well compared to R-trees for small number of data (Paspalis, 2003). As each B+-tree is linked to one object it is possible to develop a secondary structure so as to access directly the temporal data of specific objects, without the need to first determine their location. This can be useful for the notion of hierarchy of information sources.

Furthermore, it has been noticed that the most recent data are considered of higher interest than the older one. It has also been noticed that inserts are generally held at rightmost of the structure, where are found the newest nodes. Therefore, the temporal tree has been modified to add a direct link between the root and the latest node. While maintaining this link requires minimum work for the system, a simple test prevents being forced to traverse the whole tree so as to append or to find the requested data. This direct link is useful to save processing time.

As most, if not all, of the updates take place to the rightmost part of the temporal tree, the fill factor of leaf nodes can be placed higher than usual. Deletes should be somehow rare under normal conditions, and updates that do not concern the newest data should be even rarer, unless the systems experiences lag time due to network problems between the sensors and the database. Therefore the split and merge procedures can be changed so that the nodes can be filled almost at their maximum capacity.

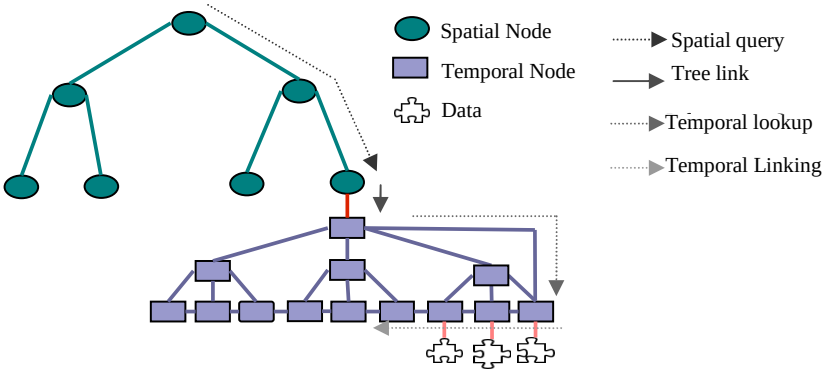


Figure 6. A Point / Interval Query

Figure 6 shows an example of point / interval query. In order to process queries, it is first needed to determine which information source it is relating to. Therefore, queries first browse the spatial sub-tree to determine the relevant node. Once this node has been determined, we can access to its temporal sub-tree. For interval queries, the next step is to look for the end of the interval. Optimally, it shall be on the last node of the sub-structure. From this point, it is possible to use the links between the leaf nodes to fetch all the data of the requested interval.

This configuration implies that this tree is more specifically designed for queries on the most recent data. Spatial range / temporal interval queries that do not end at the present time, do not take any advantage of the specificities of the tree.

4.1.3 *Experimental results*

Various tests have been conducted between the PoTree and R*-tree structures (thanks to Hadjieleftheriou's implementation, Hadjieleftheriou 2004). As a matter of fact, every indexing solution is designed to focus on some features, some applications. While some structures have been designed to store data related to mobile sensors (evolution of positions and alike, cf. TPR-tree (Saltenis, 2000)), or to store locations and timestamps of special events, so far the focus has not been on data issued from sensors. As a consequence the PoTree has been compared to the most widespread indexing structure in the spatiotemporal database world: the R*-tree.

Randomly generated data have been generated and sequentially issued to a fixed number of random points acting as information sources. Tests have been conducted changing the total number of data to index (1000-200000), the number of information sources (10-100) used and the portion of the base to scan for interval queries. The tests have been conducted on a 1.6 G Hz, 128 Mo RAM computer, running Linux. The programming language used was Java.

Due to the differentiation of spatial and temporal component, and due to the fact that data were coming from a finite set of spatial points, the PoTree built time has been greatly reduced compared to the R*-tree (Figure 7). Please note that for a better readability concerning time, scale on figure 7 uses seconds instead of milliseconds (Figure 8).

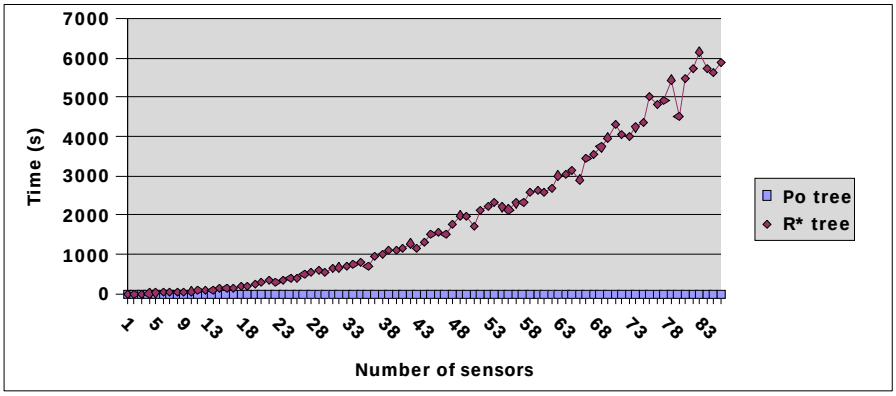


Figure 7: Comparing R*-tree and PoTree construction time

While 25 000 points stemming from 100 different locations were indexed in less than one second with the PoTree, it took nearly 45 seconds with a R*-tree. Other tests have shown that the construction time of the PoTree evolved linearly with the number of stations, the number of different spatial locations (Figure 8).

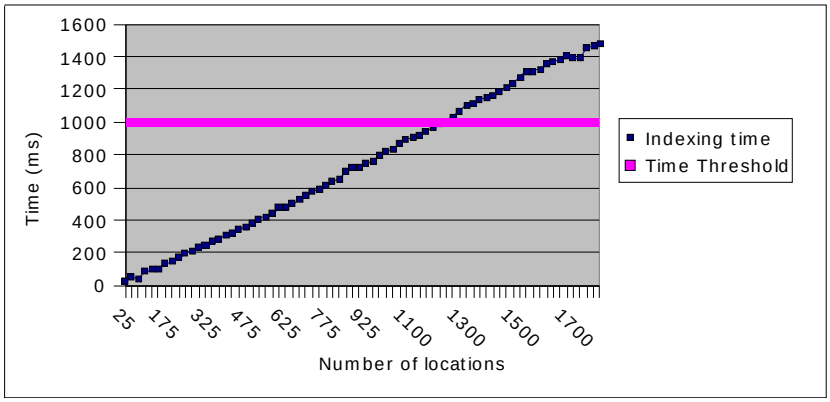


Figure 8. Influence of the number of station locations on Tree construction time

Various queries (Noel 2004b) have shown interesting properties as well. Interval queries took an advantage of the linking of the temporal nodes of the PoTree. For point-interval queries, the PoTree can be up to 8 times faster the R*-tree. While for interval queries the difference has shown much lighter, it still remains in favour of PoTree solution, as shown in Figure 9. On this figure, the last 10% of the collected data were fetched. The spatial range covered the whole of possible locations. It is visible than for a low number of data the R*-tree fares better, yet when the amount of data rises past 6000, it is the PoTree is more efficient.

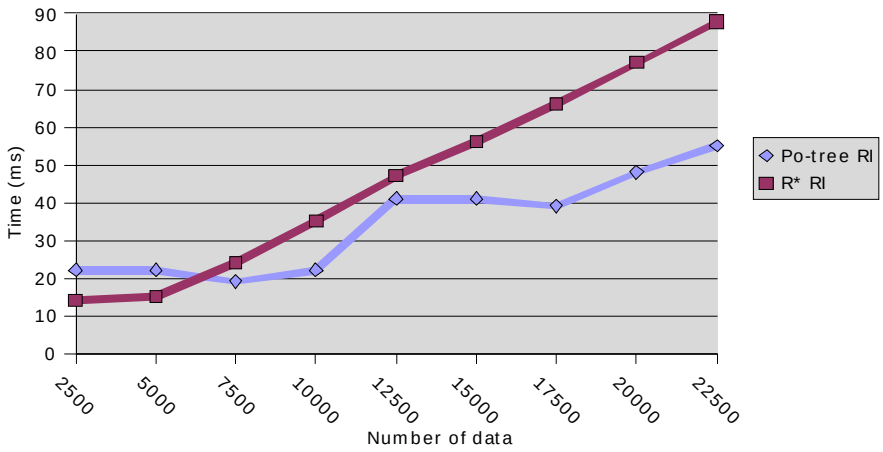


Figure 9. Range-Interval queries

Results obtained have shown that the PoTree is compatible with the constraints set by application cases concerning environmental risk monitoring: favouring the newest data, processing of masses of data in a given time, fixed set of spatial sources, possibility of real-time system use. Even though the mobility is not yet easily managed, the PoTree meets the initial specifications defined in 5.1.

The PoTree indexing method is more efficient than R*trees for spatial windows / time interval queries (see figure 13 below). The test data were data issued by 68 sensors, covering the whole studied area. The queried was based on the last ten percent of all sensors. This can be explained by the use of sensor specific sub-trees to limit data access cost.

4.2 *Spatiotemporal real time indexing structure with sensors agility (location changes) managing*

The PoTree offers interesting results for data issued by fixed sensor networks, but it lacks finer agility management. The next sections describe the PasTree, another main-memory database indexing structure. It also manages real-time data and real-time querying. However it also offers

sensor agility management and multi-dimensional access to the data. Queries can be based on spatiotemporal aspects or on sensor identifiers.

4.2.1 *Overview of a solution*

While the structure previously presented uses two structures to index spatiotemporal data, it does not allow sensor location changes. If a location was to change, a new temporal sub-tree would have to be created. This could be troublesome if a specific location becomes irrelevant, or in order to index some specific data. For example, seismic epicentres are usually determined by specialized processes but can be considered as data to be indexed. Those epicentres can be considered as data collected from evolving sensors. So, a new tree is needed. The PasTree has to deal with location changes, yet remain focused on prioritizing the newest data and update transactions.

Sensors can move from time to time. The spatial sub-tree should be able to track these modifications. Different approaches exist, yet we shall aim at introducing a multi version approach. A given node records the presence of past sensors (with an end-time) and of actual sensors. So as to offer other querying options, quadtrees could be used instead of Kd-trees, as seen in figure 10 and 11.

The temporal sub-trees should also keep tracks of the location changes. Each of these sub-trees is related to a specific sensor. Keeping track of their location allows following the movement of sensors through a time interval without querying the spatial sub-tree. The temporal sub-trees of the PoTree can be suitably adjusted to differentiate two kinds of entries: measurement data and location changes.

A tertiary structure, based on B+ tree keeps record of the sensor IDs. As a matter of fact, some queries do not need spatiotemporal properties. Scientists are used to use sensor identifiers, and do not always rely on spatiotemporal properties. Therefore this structure is directly linked to the temporal, sensor related sub-trees without using the spatial sub-tree.

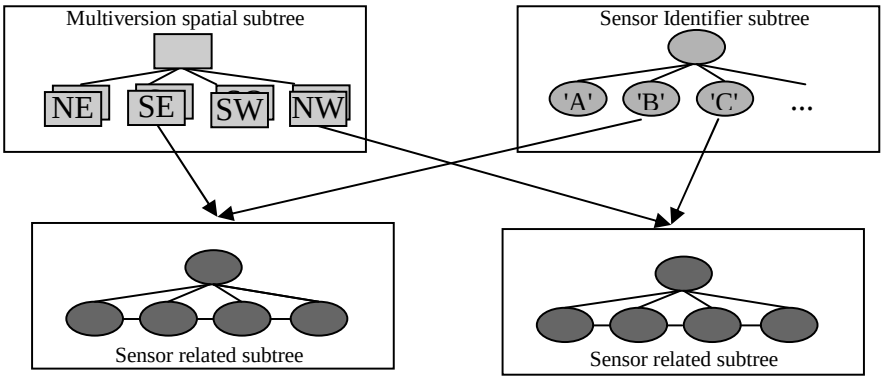


Figure 10. A possible multi-tree structure for a real time spatiotemporal index for agile sensor database

4.2.2 *Details and discussion*

The PasTree suffers from data duplication, yet it also allows for more request types than the PoTree. As a matter of fact, queries can be based on the sensor ID as well as on spatiotemporal properties. It allows users to follow a specific sensor through its location changes or to have a look at different sensors passing through a region during a lapse of time. It stills focuses on update transactions and on the newest data.

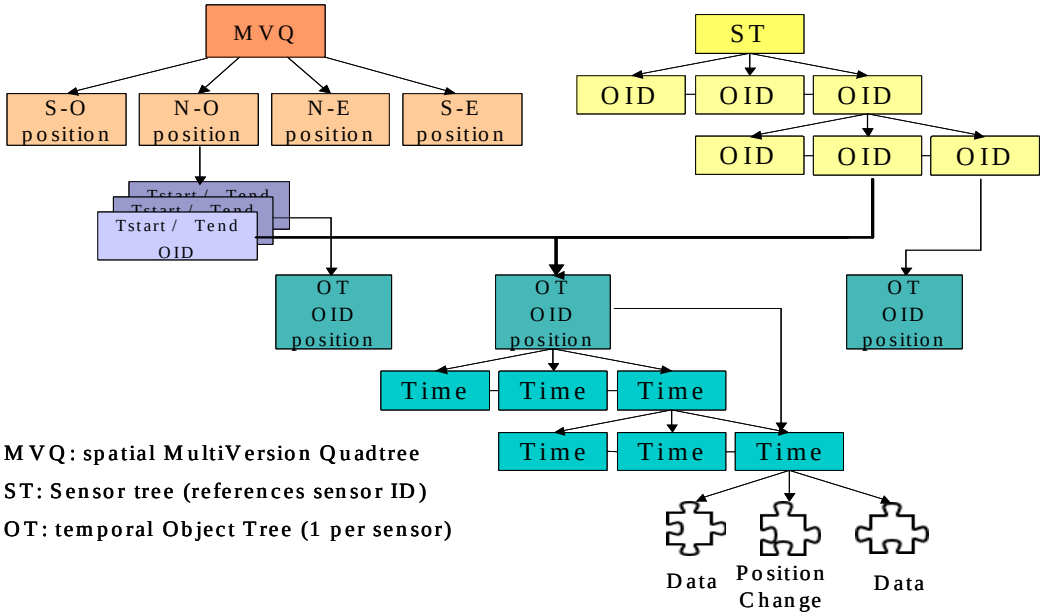


Figure 11. Detailed structure of the PasTree

4.2.3 *Experimental results*

The PasTree was implemented in Java (Sun VM) on a 1.3 Ghz Athlon XP based system with 512 Mo of memory. Different tests have been carried out on the PasTree. Randomly generated data, as well as real-world data (from the K-Net) were used.

The PasTree has been tested against the R*Tree. Once again, this choice is linked to the lack of purely sensor-based indexing structures focusing on data and not the spatiotemporal properties of the sensor. It was also tested against the PoTree with static sensors. As the PasTree does not specifically aims at managing mobility, it has not been tested against mobility-centred access methods, such as the TPR-tree.

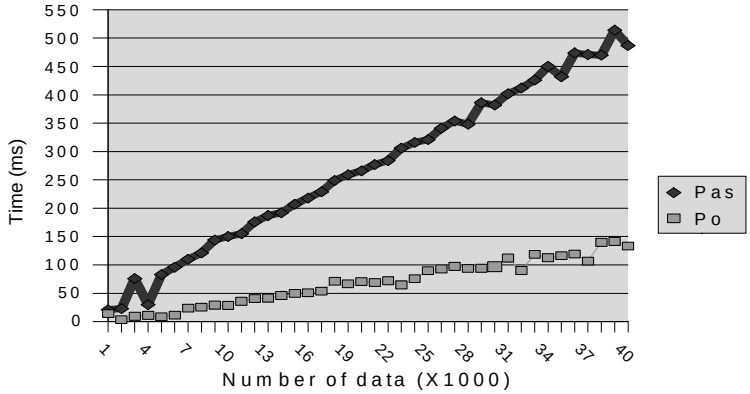


Figure 12. PoTree / PasTree building time related to the number of data to index (68 sensors)

Figure 12 shows the impact of the number of data on the PasTree building time. This test has been carried out with real data issued by 68 sensors. A semi-linear raise in building time can be noted. As a matter of fact, during the updates, the spatial and identifier sub-trees did not change.

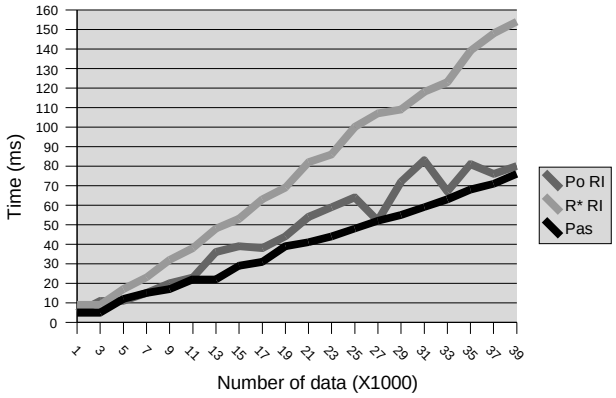
Only the sensor sub-trees had to be really updated. Even then, the structure made use of the direct link to the last node. This sub-tree needs only further processing when the last node has to split. The spatial sub-tree is only updated if agility becomes a factor. Otherwise, it is only used to reach the relevant sensor sub-tree (which takes $\theta(n \log Nl)$, with Nl the number of locations used).

Other tests have been carried out to understand the impact of agility on the building cost. Randomly generated data simulating data issued by 100 sensors with different agility levels have shown that the PasTree can process data issued by 600 fixed sensors and 300 agile sensors (agility of 0.5). The PoTree could have dealt with 1200 fixed sensors in similar conditions. However, it could not have dealt with agility.

Comparisons with the PoTree have shown that this structure was 3 to 4 times faster in building the index. However, the limitations on the answerable queries limit the potential of the PoTree. The various data access patterns provided by the PasTree are proposed at the cost of construction time.

Even then, the PasTree is still far more efficient than the R*tree. As a matter of fact, it uses the notion of information source, gathering data into specific sub-tree to limit updating costs.

Figure 13. Spatial window/Time interval query solving time related to the number of data within the base



The PasTree is, as the PoTree, more efficient than R*trees for spatial windows / time interval queries (figure 13). The test data were data issued by 68 real sensors, covering the whole studied area. The queried was based on the last ten percent of all sensors. This can be explained by the use of sensor specific sub-trees to limit data access cost.

From these different results it appears that the PasTree is a suitable database indexing method for data issued from a set of real-time sensors. It offers spatiotemporal as well as sensor-identifier-based data access patterns. Furthermore, it can manage sensor agility. It uses more resources than the PoTree in processing queries. However, it also adds new features, linked to sensor agility and users needs to alleviate the cost of using the PasTree.

5 OUTLINE OF A METHOD OF INDEXING FOR MANAGEMENT OF MEMORY SATURATION: StH

The StH method that we outlined is an indexing method for spatiotemporal real time database in main memory. This index is able to manage real time measurements collected by a network of agile sensors.

5.1 *Specifications*

The StH is like the PasTree. It is dedicated to answer problems related to the real time spatiotemporal indexing of data resulting from a network of agile sensors. The StH method is focused on the initial role of sensors namely collecting data resulting from one sensor in only one indexing substructure. It aims at allowing the resolution of queries based on identifiers of sensors as well as on spatiotemporal attributes.

The StH is more particularly dedicated to the databases into main memories, connected to a data warehouse to store the least important data. In order to prevent memory saturation of the database, the StH index takes into account the need for "emptying" the database by selecting data to be transferred downwards the data warehouse according to the importance of data. The

importance of data is determined according to several criteria. The most recent data are regarded as priority, but it is advisable to consider other criteria. Variation of value between two data can impact the importance of the measurements. Small variation involves that the new measurement is less important.

Moreover, concerning environmental risk monitoring, it appeared that some process could include data corresponding to observations on particular areas in addition to the data resulting from sensors. Thus, in the case of the volcanic monitoring for example, the passage of one lava flow can be regarded as an observation relative to a zone, and not at a located point.

The StH index uses the global structure of the PasTree adding development of particular methods of management of memory saturation.

5.2 *Description of the structure of the StH index*

The total structure organisation of the PasTree index based on a set of substructures is used again (Figure 14) to define the StH index. The difference between the StH and the PasTree concerns the central substructure. The central substructure of the StH is related to sources of information (sensors), contains all the data relative to one sensor and is able to ensure a management of memory saturation. One of the two access structures of the PasTree is re-used just as it is: the tree based on sensor identifiers. The other access structure, the spatial tree, is modified.

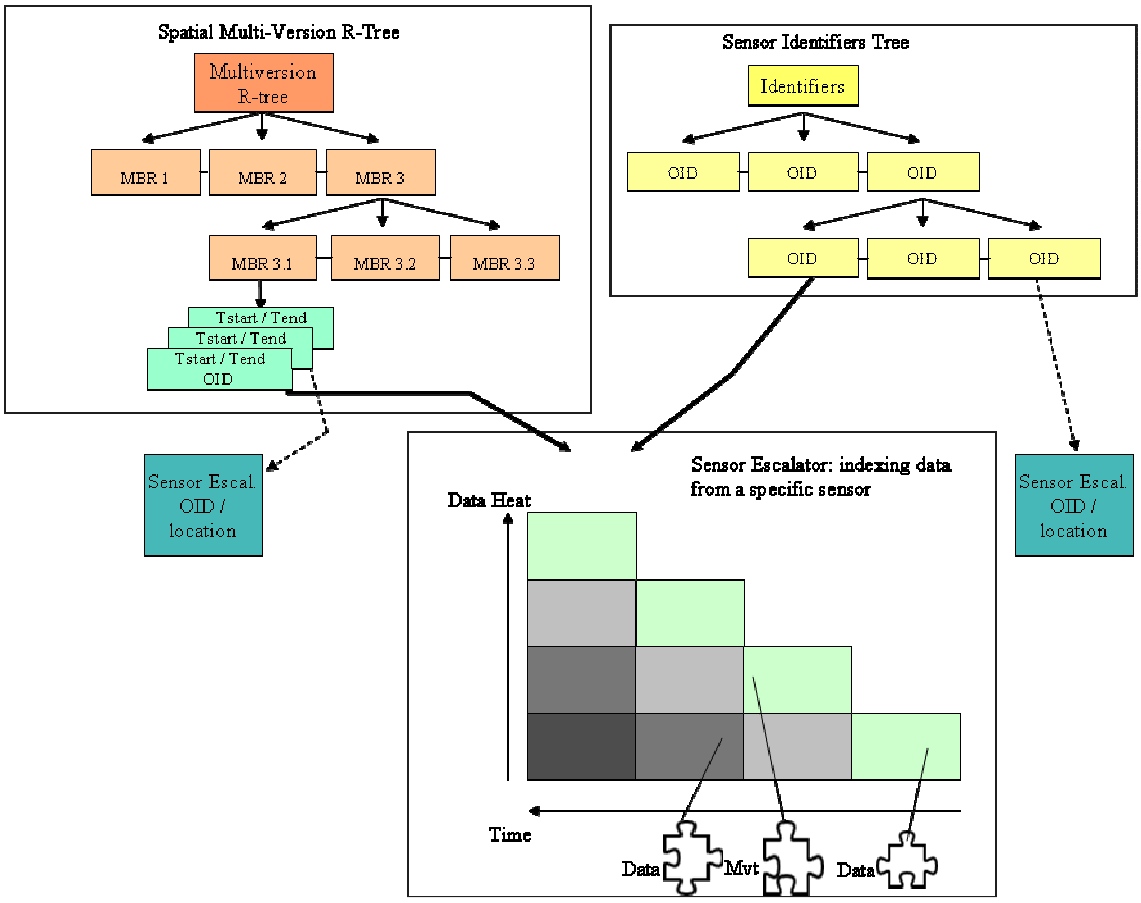


Figure 14. Structure of the StH index

As said before, the tree for sensor identifiers results from the PasTree index. This tree allows database queries based on sensors identifiers.

In the StH index, the spatial tree, based on a multiversion quadtree in the PasTree, is now based on a multi-version R-tree. This structure allows to record data resulting from specific located points (sensors) as well as data resulting from observations on areas (difficult to support by a quadtree). It also allows for intersection or nearest neighbours spatial queries.

The major change comes from the substructure based on sensors. It is no more a tree but it is now a dynamic staircase. This sensor staircase behaves like an escalator (figure 15). The structure is filled by the left and initializes procedure of dumping by the right. The staircase is composed of steps (cf. Figure 15). Each step is a pile of scales. In each scale, pointers towards the data are stored. The choice of the scale of storage is determined by the significance of data (according to the application). A function calculates the heat of data which characterizes the significance of the data. When a number of data by step is reached, the procedure of dumping is carried out. During the dumping procedure (Figure 16), the lowest scale of each step (corresponding to the coldest data) "is flushed" towards a data warehouse and removed from the StH structure. A new step is then created (on the left of the structure) to collect the new data. The structure is dumped gradually, preserving the most significant data for a longer time.

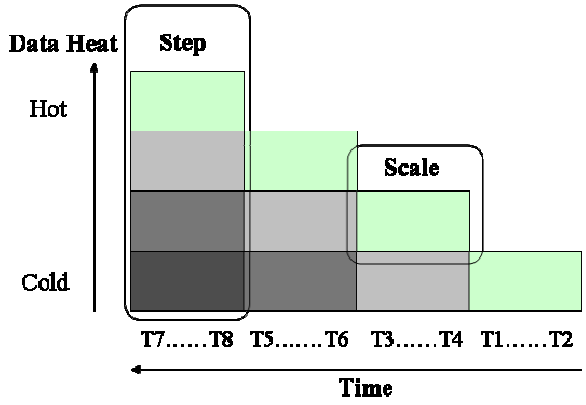


Figure 15. Structure of the StH escalator of sensor

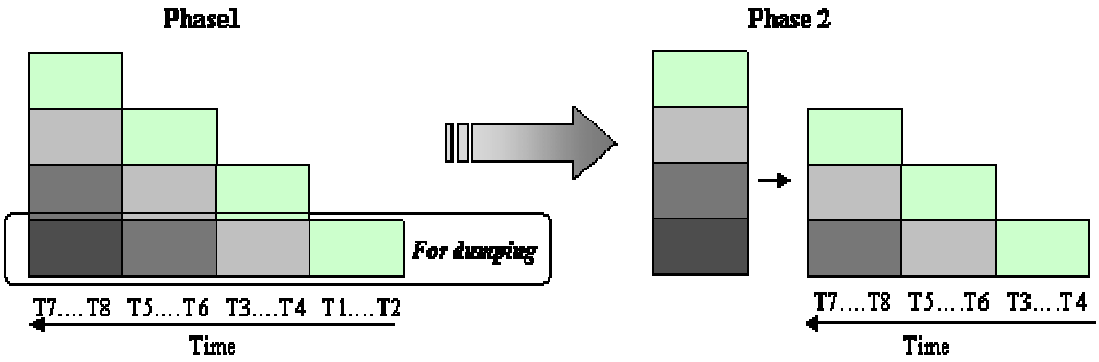


Figure 16. StH index dumping

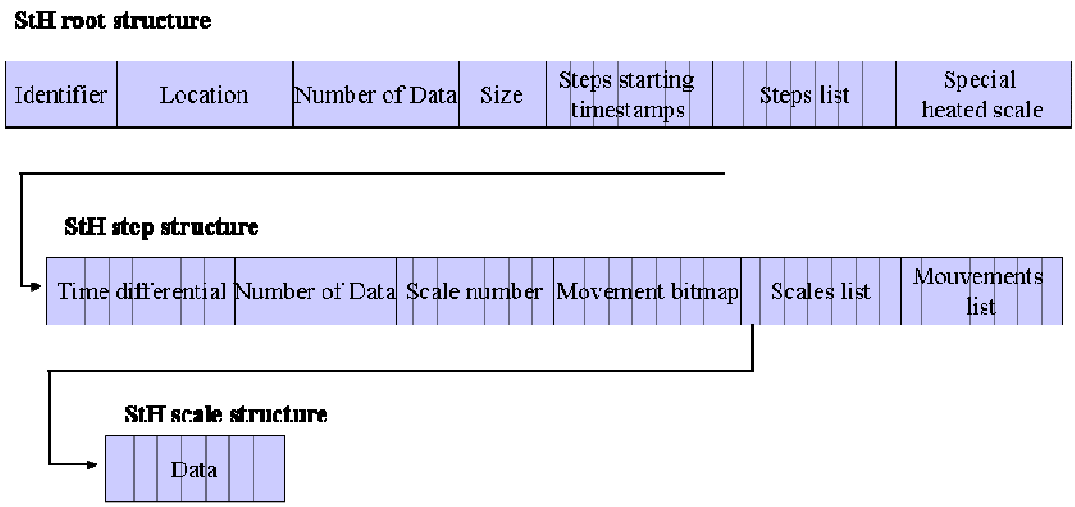


Figure 17. Data structure of the escalator of the StH index

Concerning the global structure of the StH index, several attributes allow the resolution of the most current queries (Figure 17, root). In addition to the identifier of the sensor associated with one staircase, the number of data contained in the staircase, the memory size and the last known position of the sensor, a pointer on a table containing the temporal markers of each step is stored. This table is used to determine the step to question so as to answer a query. The root of the StH preserves also a pointer on a table containing the headings of different steps. Bound to the staircase is also a special step, which contains only one scale. In this scale are preserved the data explicitly qualified "hot data" by the user. These data are copied from the staircase and are preserved explicitly in this structure. Only an explicit declaration of the user can empty this special step, created in order to force the conservation of certain data.

A pointer on a table of differential time is associated to each step (Figure 17). The difference in time since the origin of creation of the step is preserved there. It is consequently possible to use shorter temporal records in order to limit the size of the structure. A notation of the differentials on two bytes instead of four is used. Concerning the step, a pointer on a table of bitmaps is stored. A bitmap is associated to each step. They are used in order to determine in which scale is a particular data as well as a number of records of a particular measure in the scale. Another bitmap is also preserved indicating the changes of locations of the sensor. If a change of location takes place, the bit corresponding to the data is placed at the value TRUE. The step also contains a pointer towards a table of pointers on scales. Finally, it contains a bond towards a list of locations corresponding to the movements carried out during the period covered by the step. The scales (Figure 17, scale) are finally made up only of measurement records.

Finding a particular data is resolved by counting the number of data between the beginning of the scale and the data to be reached. This involves accessing the differential time record so as to determine the record number to reach. From here on, the bitmaps table can be accessed. A first access is used to determine which bitmap is set at the specific record number. This determines which specific scale holds the data. Another access to this specific bitmap determines the data number within the scale. Movement and location queries are based on the same methods with the movement bitmap and list.

It is necessary to note the importance to use a suitable function to determine heat of data. If the function is too much complex, the cost will be too high. Badly defined, the function will not offer a homogeneous distribution of data during the phases of normal system activities. This will lead to bad performances of functions of "emptying". It is quite obvious that at the time of particular activities, function must classify a greater number of hotter data.

Actually, the structure of the StH is currently under experimentation and validation. The experimental phase already emphasized the role of the function of heat attribution, which must be adapted to each case of specific application.

6 CONCLUSION

Time is crucial for emergency response and risk monitoring. Risk monitoring is based on collections of sensor data. In this paper, a discussion concerning real time spatiotemporal indexing and real time main memory management for sensor databases is presented. As an example, a global architecture of environmental risk monitoring system is detailed. The difficulties concerned real time data management in database and also spatiotemporal data management. After a state of the art on data structuring according to real time and spatiotemporal specifications, some possible solutions for real time spatiotemporal data indexing are presented to allow real time, fast data structuring and queries. One of the presented indexing solutions is dedicated to real time spatiotemporal data issued from fixed sensors. Another one is dedicated to real time spatiotemporal data collected from agile sensors. This latter solution offers multidimensional accesses: according to purely spatiotemporal criteria or sensor identifiers. In addition, the last indexing solution allows managing saturation of main memory according to the significance of data which depend on applications.

The indexing solutions detailed are new and efficient as they permit to:

- store spatiotemporal data collected in real-time from sensor network,

- index spatiotemporal data in real time allowing real time spatiotemporal queries and facilitating rapid access to recent data (most of existing methods do not privilege last entered data)
- allow real time spatiotemporal queries taking into account sensor location and move (fixed sensor, agile sensor)
- manage in real time main memory saturation using data significance (application based) to elect data to store on long period.

7 REFERENCES

ALMEIDA V.T., GUTING R.H. 2005. Indexing the Trajectories of Moving Objects in Networks, in: *GeoInformatica*, vol. 9(1), pp33-60

BARBARÁ D. 1999. Mobile Computing and Databases - A Survey. In: *IEEE Transactions on Knowledge and Data Engineering*, vol. 11 (1), pp. 108-117.

BEHR FJ. 1995 *Mobile GIS: Contributing to Corporate Benefits*. DA/DSM Seminar, 20 November 1995, Rome, Italy, http://www.graphservice.de/papers/mobile_g.htm

BOHANNON P., McILROY P., RASTOGI R.. « Main-Memory Index Structures with Fixed-Size Partial Keys », *In SIGMOD Conference* , 2001.

BUELHER K., MCKEE L. (eds). 1996. *The OpenGIS Guide, Introduction to Interoperable Geoprocessing*. Open GIS Consortium. http://www.opengis.org

CHAKKA V.P., EVERSPAUGH A., PATEL J.M. 2003. *Indexing large Data sets with SETI*, Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003), Asilomar, january 2003

EIMAN E. 2003. Research *Directions in Sensor Data Streams: Solutions and Challenges*, technical report DCIS-TR-527, Rutgers University,

FRENTSOS. 2003. Indexing Objects Moving on Fixed Networks, Proceedings of 8th Int. Symp. On Spatial and Temporal Database (SSTD)

GUNADHI H., SEGEV A. 1993. *Efficient indexing methods for temporal relation*, In IEEE Transactions on knowledge and Data Engineering, 5(3), 496-509

GUTING RH, BOHLEN MH, ERWIG M., JENSEN CS, LORENTZOS NA, SCHNEIDER M, VAZIRGIANNIS M. 2000. *A Foundation for Representing and Querying Moving Objects*. ACM Transactions on Database Systems, Vol. 25 (1) pp. 1-42.

HADJIELEFTHERIOU M. 2006. Spatial Index Library [Online], viewable at: http://www.cs.ucr.edu/~marioh/spatialindex/, (last consulted 01/11/2006)

HARITSA J.R., SESHADRI S. 2001. *Real-time index concurrency control*. In Real Time Database System – Architecture and Techniques, Kluwer Academic Publishers, Boston, ISBN: 0-7923-7218-2, 60-74

Intanagonwiwat C., Estrin D., Govindam R. et al. 2001. Impact of Network Density on Data Aggregation in Wireless Sensor Networks, in Proceedings of the International Conference on Distributed Computing Systems, Vienna, 2001

KIM K., CHA S.K., KWON K. 2001. Optimizing Multidimensional Index Trees for Main Memory Access, Proceedings of 2001 ACM SIGMOD Int. Conf. On Mnagement of Data, USA May 2001, pp. 139-150

LAM K.Y., KUO T.W.. 2001. Real time database systems: an overview of systems characteristics and issues. In Real Time Database System – Architecture and Techniques, Kluwer Academic Publishers, Boston, ISBN: 0-7923-7218-2, 4-16

LAURINI R., SERVIGNE S., NOEL G. Soft Real-time GIS for Disaster Management, GI4DM, In: Proceedings of International Symposium on Geo-information for Disaster Management. Delft, The Netherlands March 21-23, 2005. 10 p

LONG M., MURPHY R., PARKER L. 2003. *Distributed Multi-Agent Diagnosis and Recovery from Sensor Failures*, IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS), Vol. 3, pp. 2506-2513

MIN Y.S., YANG C.Y., YOO J.S., SHIM J.M., SONG S.I., « PCR-Tree: An Enhanced Cache Conscious Multi-dimensional Index Structures », *proceedings of DEXA 2004*, pp. 212-221

MOKBEL M., GHANEM T.M., AREF W.G. 2003. *Spatiotemporal Access Methods*, IEEE Data Engineering Bulletin, Vol 26, n°2, pp. 40-49.

NASCIMENTO M. A., SILVA J. R. O. 1998. *Towards Historical R-trees*, In: *Proceedings of ACM Symposium on Applied Computing* (ACM-SAC) Atlanta, USA, p. 235-240

NOEL G., SERVIGNE S., LAURINI R. 2004a. *Real-time spatiotemporal data indexing structure*, Proceedings of 2004 AGILE 7th conference on Geographic Information Science, Heraklion, 2004, pp. 261-268.

NOEL G., SERVIGNE S., LAURINI R. 2004b. *The Po-tree: a soft real-time spatiotemporal data indexing structure*, Proceedings of 11th SDH International Symposium on Spatial Data Handling, Leicester, 2004, 10p.

NOEL G., SERVIGNE S., LAURINI R. 2005. Spatial and Temporal Information Structuring for Natural risk monitoring, GISPlanet'2005, Lisbonne 30 mai-2 Juin 2005. 10 p.

OOI B.C., TAN K.L. 1997. *Spatial Databases*. In Indexing Techniques for Advanced Database Systems, Kluwer Academic Publishers, Boston, ISBN 0-7923-9985-4, 39-75

PASPALIS N. 2003*Implementation of Range searching Data-Structures and Algorithms* [Online], viewable at: http://www.cs.ucsb.edu/~nearchos/cs235/cs235.html, (last consulted 11/20/03)

PRICHARD J., FORTIER P. 1997. *Real-Time SQL*. In: *Second International Workshop on Real-Time Databases* September 18-19, 1997 Burlington, Vermont, USA, pp. 289-310.

RAO J., ROSS K. 2000. Makinb B+ trees Cache Conscious in Main Memory, Proceedings of ACM SIGMOD 2000, pp. 475-486

RAATIKKA V., « Cache-Conscious Index Structures for Main-Memory Databases », *Master's Thesis of university of Helsinki*, Departement of computer Sciences, 2004

RATNASAMY S., ESTRIN D., GOVINDAN R., et al. 2002. Data-Centric Storage in Sensornets, in Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, 2002

SAMET H. The Quadtree and Related Hierarchical Data Structures ",1984 , ACM Computer Survey , Vol 16 (2) , pp187-260

SALTENIS S., JENSEN C., LEUTENEGGER S., LOPEZ M. 2000. Indexing the Positions of Continuously Moving Objects, In Proceedings of ACM SIGMOD 2000

SATNAM A., AGOGINO A.M., MORJARIA M., A. 2001. Methodology for Intelligent Sensor Measurement, Validation, Fusion, and Fault Detection for Equipment Monitoring and Diagnostics, *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, , vol 15, n°4, 2001, 307-320

SERVIGNE S., TANZI T., NOEL G. Telegeomatic System and Real Time Spatiotemporal Database. Proceedings of Urban Data Management Systems, UDMS'06. Aalborg, Denmark, May 15-17, 2006. 12 p

SITZMANN I., STUCKEY P.J., « Compacting discriminator information for spatial trees », *proceedings of the Thirteenth Australasian Database Conference*, 2002, pp. 167-176

SONG Z., ROUSSOPOULOS N., (2003), *SEB-tree: an Approach to Index Continuously Moving Objects*, in Mobile Data Management (MDM), January 2003, pp. 340-344

THEODORIDIS Y., VAZIRGIANNIS M., SELLIS T. (1996) *Spatiotemporal indexing for large multimedia application*, In Proceedings of the 3rd IEEE conference on multimedia computing and systems (ICMCS)

WANG X., ZHOU X., LU S. (2000) *Spatiotemporal Data Modeling and Management: A Survey*, In Proceedings of the 36th International Conference on Technology of Object-Oriented

USGS, Earthworm Overview [Online], viewable at http://folkworm.ceri.memphis.edu/ew-doc/OVERVIEW/1_History.htm, (last consulted 01/10/06)

WOLFSON O. (1998) *Moving Objects Databases: Issues and Solutions*. In: the Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM98), Capri, (Italy), July 1-3, 1998, pp. 111-122.