periodica polytechnica

Electrical Engineering
54/3-4 (2010) 123–128
doi: 10.3311/pp.ee.2010-3-4.06
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

In-memory preprocessing of streaming sensory data – a partitioned relational database approach

József Marton / *Sándor* Gajdos

Received 2008-10-03

Abstract

In this paper we present a database architecture and an application area and method in detail. As sensory data stream in the database it is efficient to preprocess them in-memory before integrating in the repository in order to save storage I/O cost. After data are integrated, it is important to allow efficient quering based on data retrieval profile. This can also be supported by the presented database architecture by partitioning the database upon different criteria. It is mandatory to hide the internal partitioned architectural details from higher layers, so options to allow transparent quering options are also presented. We have implemented a test system and experimental results are also given.

Keywords

Partitioned database · streaming data preprocessing ·

József Marton
Sándor Gajdos
Database Research Labs, Department of Telecommunication and Mediainformatics, BME, H-1117 Budapest, Magyar tudósok körútja 2., Hungary

1 Introduction

As the application area of sensors broadens the amount of streaming sensory data to be processed is also increasing. Analytical systems working on heterogeneous sensory data require efficient preprocessing techniques because they require an integrated view of different data sources in order to calculate data in a short time period. In this paper, we present a partitioned database approach, i.e. a composite of memory-resident and disc-resident partition. This approach can provide two to five-times performance improvement in the data preprocessing stage. It can adapt to hardware resources efficiently, i.e. adding more CPU cores and random access memory increases performance in a near-linear fashion. This is especially the case when data streams to be integrated are independent of each other ? which is a common premise. This paper is organized as follows. In Section 2 we outline where do sensory data originate from and why do they need preprocessing. Then we move to data and database partitioning in general, and describe why it can be useful. In Section 3 we introduce the partitioned database approach and describe the relation of partitions to each other and their view from outside. We also address options to allow for transparent data retrieval. After showing the traditional, off-line way of preprocessing in Section 4 we move to the proposed way that is based on the partitioned database approach. We also give evaluation results to show the performance gain that could be achieved. In Section 5 we summarize our results.

2 Sensory data, preprocessing and partitioning

This section is further divided into three subsections regarding sensory data, their preprocessing as well as data and database partitioning in general.

2.1 Sensory data

The majority of data is distributed as they are produced by individual sensors or by a collection of sensors, i.e. sensor network distributed spatially and connected by communication lines. Data is collected over a broad area and streams in at a much higher rate than before [1]. Transmission of data might be done using several different approaches. Two such approaches

are transmission over dedicated measurement lines and through peer-to-peer network. Physical connectivity medium can be, for instance, some copper lines or wireless radio channels.

Sensory data, in most of the cases, is time-series [2] data i.e. as time approaches data from more and more recent measurements are published. The more recent measurement data has the exact same semantics compared to the very first piece of data that emerged from the sensor.

For example, stock exchange quotes can be seen as streaming sensory data, as data (i.e. quote) are produced from time to time and the more recent pieces of data have the same semantics as older pieces of data. Data from different kind of sensors, or from similar sensors observing different subject (e.g. temperature measurements from different reactor blocks of a power station) can be, in most of the cases, preprocessed and transferred independently of each other. Independent preprocessing, however, does not mean that pieces of data can be preprocessed on their own. Data might get context for interpretation from other sensory data (e.g. changes in neutron flux and temperature of the reactor zone are correlated for a given family of nuclear reactors).

2.2 Purpose of preprocessing

The main goal of preprocessing is to provide coherent view of data from different sources and enable more efficient analysis by reducing the complexity of data [3]. Coherent view is mandatory in order to allow for comparison and correlation of data over time or between data of different semantics.

To achieve its goal, preprocessing might involve measurement unit conversion, scaling or code space translation.

Code space translation might allow for more compact storage, more efficient cross-referencing at retrieval-time (because of simpler primary or foreign keys). This is why code space translation is also widely used in data warehousing applications. As an undoubtedly useful side effect of code space translation erroneous input data can be identified and data owner can be informed about that.

2.3 Data and database partitioning

Partitioning data or even a database means physical or logical separation of stored data based on various criteria. Partitioning might also involve partial mirroring of data. Partitioning might be done such a way that different partitions reside on different storage media [4].

Partitioning criteria might be based on e.g. some hierarchy or freshness of data (i.e. in some applications the more fresh data tend to be needed in more detail and more often). In case of partial mirroring separate partition might be created for separate application (e.g. alert logging of data that crossed some critical value, or anonymized data to satisfy regulations).

In case of solely separation of data, transparent data access need to be provided, that is, from the users' (or applications') point of view the same access path should work to access data regardless of the actual partition it is stored in.

Why do we need to partition our data? Active portion of data might be small compared to the whole database size. Identifying and separating this active database portion makes it cheaper to achieve the needed data retrieval performance. This is especially the case when active data portion is stored on faster media than historical data, which can even be stored on massive array of idle disks (MAID) [4] or tape drives.

Partitioning also eases data management. Business rules define how long historical data need to be kept online. In most cases these business rules define a sliding window. As the window slides over some data partition (i.e. data age out), that partition can be archived and put off-line without affecting continuous access to the active database partition (i.e. arrival and integration of fresh data and data retrieval for analysis are not affected).

3 Partitioned database

In this section we present the partitioned database (DB) that consists of a memory-resident and a disc-resident database partition. We also outline the linkage of partitions to each other as well as options to hide partitioned scheme from users. In the last subsection, we introduce two products that can be configured to operate as partitioned database.

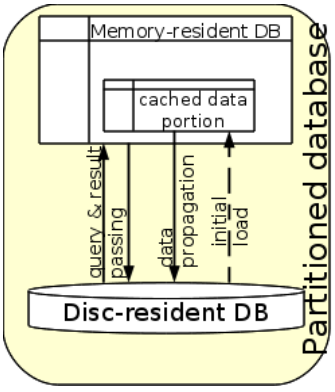

Fig. 1. Architecture of the partitioned database showing IMDB and DRDB parts. Arrows indicate the cases when data transfer or query serving might occur.

3.1 Architecture

In-Memory Database (IMDB) or Main-Memory Database is a database management system where the primary copy of managed data reside in the physical memory [5] that allows for random access. In contrast, Disc-Resident Database (DRDB) stores the primary copy of managed data on disc with block-level access.

IMDB systems may also involve using of discs to store secondary copies (e.g. backup) or to ensure data durability, whereas DRDB systems may be backed up to disc or tape and in case of partitioned storage they might involve multiple layers of storage as described where we were talking about the relation of historical data and MAID devices.

DRDB systems also use physical memory for buffering, but data structures are not converted to suit its random access characteristic. This, in fact, leads us to the key difference between IMDB and DRDB systems. DRDB systems, their algorithms and data structures were designed with block-level storage organization in mind whereas In-Memory Database Management Systems' algorithms and data structures [6]-[8] were designed with random access storage characteristic and CPU cache utilization in mind.

Our approach to partitioned database consists of two parts: in-memory and disc-resident partition as shown on Fig. 1. In-memory partition stores frequently accessed subset of data (e.g. fresh measurement data) along with configuration and lookup tables which drive processing and interpretation of the data. Disc-resident partition stores all the data of in-memory partition as well as less frequently accessed data in order to provide long term storage.

3.2 Migration of data

In the previous section we have mentioned that data structures differ in IMDB and DRDB systems to suite primary storage media. As data transfer occurs between partitions data representation needs to be changed, i.e. data need to be migrated. Arrows on Fig. 1 indicate data transfer cases between the two database partitions which we describe below.

When starting up a partitioned database instance, disc-resident partition already has data on its primary storage medium (i.e. on the disc). In contrast, IMDB partition needs to be loaded into the physical memory. This initial load might occur based on the data stored in the DRDB partition. Upon loading from the DRDB into the IMDB part data representation needs to be changed which demands high resources. In contrast, initial load can happen using memory image file of a checkpoint of the IMDB partition. This way initial load-time data structure changes are not needed anymore. Recently, application area of database management systems tends to expand: as they accommodate files and even act as file systems [9] both methods of initial load can be backed by disc-resident database.

As we have defined IMDB partition to hold only some subset of data available in the disc-resident partition all the changes in the IMDB partition need to be reflected in the disc-resident database. This process is called data propagation. If direct changes in the DRDB partition would be allowed, they would also need to be propagated toward the IMDB part.

Query and result passing between the two partitions is the third case of data transfer which leads us to the next topic, namely, transparency options. Result passing involve data structure migration on the partition boundary to allow for integration of sub-results computed in each parts.

3.3 Options for transparent data access

It is mandatory to hide internal architectural details of the partitioned database from components in higher layers. Basically this demands for a single port to connect to. All internal command and data flow should be controlled inside.

Two different approaches are possible. According to the first approach, the partitioned system has a separate query dispatcher component. This is called the query routing approach. The second approach, namely execution fallback, direct queries to the in-memory database management partition and, if needed, execution is redirected to the query execution engine of the disc-resident database.

The separate dispatcher component has two main roles. Receiving a query, it should analyze it in order to recognize individual elements and route them to the appropriate database partition to compute sub-results. When sub-results arrive they need to be merged and returned to the user application. By applying such dispatcher logic, queries which can be served using only the IMDB or DRDB partition, would slow down.

Should incoming requests be directed to the IMDB partition for serving, in a properly partitioned system (i.e. frequently accessed data reside in IMDB partition) high percentage of queries can be served in just one step. In case IMDB can not serve the query because lack of input data, query can be passed to the DRDB part.

Both query routing and execution fallback bases on logical description of the partitioning scheme. This logical description can be as simple as describing which data tables and of which time interval are available in the IMDB partition, or it can be more complex even involving table access statistics. Database management systems already gather statistics to support their cost-based query optimizers. By extending scope of these statistical metadata, new possibilities arise.

3.4 Implementations

In this section, we briefly introduce two in-memory database products that can be configured to operate as a partitioned database.

Oracle provides its in-memory database product TimesTen to store and manage data close to the application clients, i.e. store them in the main memory in suitable data structures and provide management routines in the clients' address space to minimize process boundary crosses. Recent TimesTen versions claim to be PL/SQL compatible with Oracle Database. TimesTen can operate over an Oracle Database to provide an in-memory cache and can manage basic partitioning of data: some tables (fully or partially) can be brought in the cache whereas others can be left in the underlying Oracle Database. TimesTen provides a single gate to access the whole database: if some operation can not be served in TimesTen, it is passed on to Oracle Database, and result is transferred to the client. Though this single gate access is provided, to achieve optimal performance, the system needs to consider the partitioned scheme of the database built upon TimesTen and Oracle Database.

IBM's soliDB in-memory relational database management system can also operate as a cache over a disk-resident database,

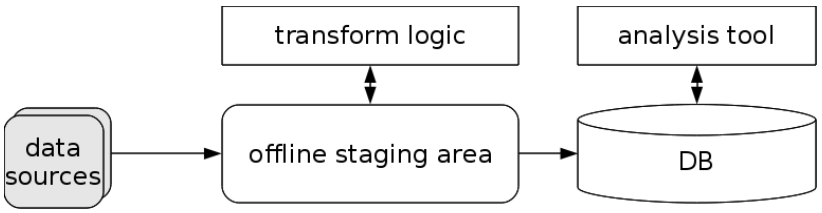

Fig. 2. Traditional off-line way of preprocessing utilizing off-line staging area. This process takes place just before integration of data into the database that publishes it for analysis tools.

called soliDB Universal Cache. While TimesTen can only operate over an Oracle Database, soliDB Universal Cache can be backed by different disk-resident database management systems including particular versions of IBM DB2, Oracle Database, Microsoft SQL Server and Sybase. soliDB, as an in-memory cache can also act as a single SQL gate to the partitioned database's data, regardless of the location of the table (i.e. whether it is in the cache or in the disk-resident database). IBM's soliDB also enables to transform the schema that is to be cached.

4 Preprocessing: traditional and proposed way

In this section we are going to discuss where preprocessing should be done and why to do it there. We then proceed to the traditional, off-line way of preprocessing and the way utilizing the partitioned database.

4.1 Where to preprocess?

Preprocessing can be done at three different levels: in the sensory device, just before integrating data into the repository or in analysis-time.

Sensory device might be configurable to allow for preprocessing, but this makes the device more expensive and the resulting system more error prone (and thus more expensive to manage). Sensory devices should be kept as simple and as stupid as possible to avoid involving extra costs.

Should preprocessing be done in analysis-time, analytical computations would take more time. Furthermore, the exact same calculation (i.e. preprocessing function) would be applied each time analytical computation involves using a particular data element. This is not affordable, as analytical computations usually take long time, and in order to shorten this process analytical systems tend to pre-calculate well defined aggregations.

The third option for preprocessing is to do it just before integrating data into the database. This is a more centralized option compared to the sensory device level as in the first option, and occurs only once per data element, which contrasts the second option. This is the point in the lifetime of data where preprocessing occurs in the traditional preprocessing workflow as well as in the proposed way.

4.2 Off-line preprocessing

Traditional way of preprocessing involves some off-line staging area just before integrating data into the database. As it is shown on Fig. 2 data are extracted from data sources and then transferred to the staging area. Initiated by some trigger events (e.g. all related data of a specified time frame have arrived) or time schedule, transformation logic applies preprocessing function before data enter the database.

Latency caused by the off-line preprocessing process depends on particular trigger events. On the other hand, this way ensures that data in the database show a consistent view for further processing.

Off-line transformations have at least two drawbacks. Temporary storing of data at some staging area not only wastes I/O and communication bandwidth but also decreases usefulness as data loose their freshness. That is why we move to near real-time transformation and integration of data into the database (repository) using the proposed partitioned database scheme.

4.3 Proposed preprocessing path

In the previous section we have discussed the traditional way of preprocessing sensory data and showed two drawbacks. Our proposed preprocessing path addresses these issues. Furthermore, near real-time preprocessing demands for high computational capacity which can be provided using several different approaches. Our proposal follow a resource-sensitive way.

We have already described the architecture of the proposed two-tier database backend in Section 3.1. and on Fig. 1. On the top of the disc-resident database layer operates a memory resident database. This means that working set of the current data and a recent portion of the historical data reside in the physical memory. In this proposed path, raw data (i.e. data that is candidate for preprocessing) enter directly the memory resident partition.

Analysis tools (and possibly other applications) connect to the partitioned database as a whole and thus they see a coherent view of the database, i.e. they can neglect the partitioning scheme applied. On the other hand, transformation logic is aware of the partitioning scheme and connects directly to the memory resident database partition, as it would be suboptimal if transformation logic neglected the fact that raw data reside in physical memory, i.e. it is stored in the memory resident database partition.

Fig. 3 shows the proposed relation of data sources, transformation logic and analysis tools to the partitioned database. Sensory data enter directly the in-memory database partition which, basically, acts as an in-memory staging area integrated into the (partitioned) database management system. Transformation logic also connects directly to the IMDB partition. This way raw data as well as lookup tables that might be needed for the transformation are available in the memory address space of the application driving the preprocessing. Accessing data al-

Tab. 1. Experimental result utilizing a partitioned and a solely dic-resident database setup. In this case, introducing the partitioned database setup, we have reached 2-times performance gain with ca. 10% average CPU time spare.

	Record rate (s^{-1})	Record rate (min^{-1})	CPU utilization (average)	Overall performance
Partitioned database setup (Oracle 10gR2 + TimesTen 7)	2500	150000	92%	22,1%
Disc-resident database setup (Oracle 10gR2)	1200	72000	90%	10,6%

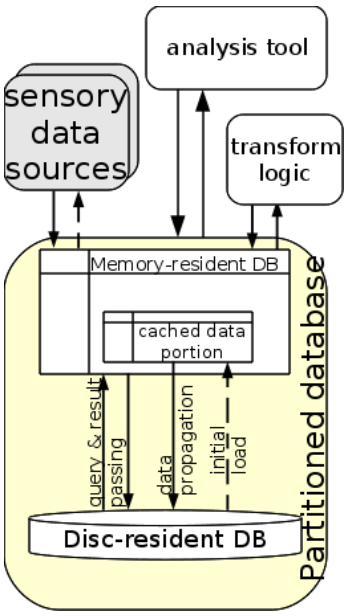

Fig. 3. Analysis tools connect to the partitioned database, whereas fresh data from data sources arrive directly in the in-memory database partition to allow for efficient preprocessing by transformation tools

ready in address space minimizes process boundary crosses, and thus increases performance. It is guaranteed by the partitioning scheme that no additional I/O cost arise during the transformation. Excluding additional I/O cost means that cost of preprocessing phase solely depends on the function applied.

Result of the transformation is appended to the repository of data and can be further used, e.g. for instant monitoring and prediction.

4.4 Experimental results

Test system has been implemented to measure performance gain of the proposed partitioned database architecture.

During the performance evaluation, we have used random generated test data that met the following specification. Incoming data records were textual and were composed of a timestamp of seconds precision (19 characters), quality code (10 characters), data source identifier (40 characters) and the measured value itself (decimal floating point number). Data arrival rate was specified as 680,000min^{-1} and most of the queries involved fresh data. Historical data needed to be retained on-line for 1 to 5 years depending on data source identifier.

We have implemented the partitioned database using Oracle Database 10g Release 2 disc-resident database management system and Oracle TimesTen 7 in-memory database management system. As it was stated in Section 3.1, the disc-resident database partition holds all the data (except raw sensory data), and the memory resident partition holds a copy of the fresh data. In our particular case, fresh data was defined as a moving window on the timestamp of data records. Raw sensory data enter the memory resident database and it is not propagated to the disc-resident database until it has been processed by the transformation logic.

Experimental results are shown in Table 1. Experiments were carried out on a consumer-class PC with AMD Athlon64 3000+ (1809MHz) CPU and and 7200rpm UltraATA 100 disc. We found that using the proposed scheme we were able to process data at 22% of the specified rate. To provide comparable measurement, we have replaced the partitioned database in our implementation by a solely disc-resident database tier. We found that a naive implementation were able to work at a performance of 5%, and a tuned version worked at a performance of 11% utilizing the same hardware. In both of the cases, CPU worked with ca. 10% performance spare.

5 Summary

In this paper we have focused on a specialized database architecture that allows resource sensitive and efficient preprocessing of streaming sensory data.

After an introductory part we have discussed where do sensory data come from, why do data need preprocessing and how data partitioning can contribute to efficient data management. We then moved on to discussion of the proposed partitioned database scheme. We discussed its architecture and showed that in three cases data representation changes were needed between partitions. This partitioning scheme needed to show a transparent view for the user applications, so we described two ways of transparent data retrieval: query routing and execution fallback.

Having discussed all the tools and concepts we needed, we moved to discussion of our proposed preprocessing path utilizing the partitioned database scheme, and showed where do performance gain come from. We also presented experimental results where we were able to achieve 440% performance gain using the proposed partitioning scheme.

References

1 **Cormode G, Garofalakis M**, *Streaming in a Connected World: Querying and Tracking Distributed Data Streams*, Tutorial in 32nd International Conference on Very Large Data Bases. Seoul, Korea, (Summer September 12).

2 **Michelberger P, Szeidl L, Várlaki P**, *Alkalmazott folyamatstatisztika és idősor-analízis (Applied process-statistics and time series analysis)*, Typotex, Budapest, 2001, ISBN 978-9-639132-44-3.

3 **Coaquira F, Acuna E**, *Applications of rough sets theory in data preprocessing for knowledge discovery*, World Congress on Engineering and Computer Science 2007, San Francisco, CA, USA., 2007.

4 **Hobbs L**, *Online Archiving with Oracle ILM & COPAN Systems' MAID Technology*, Oracle-Copan white paper, (February, 2008).

5 **Garcia-Molina H, Salem K**, *Main Memory Database Systems: An Overview*, IEEE Transactions on Knowledge and Data Engineering, **4**(6), (December, 1992).

6 **Lehman T J, Carey M J**, *Query Processing in Main Memory Database Management Systems*, Proceedings of the 1986 ACM SIGMOD international conference on Management of data, **15**(2), DOI http://doi.acm.org/10.1145/16894.16878.

7 **Analyti A, Pramanik S**, *Fast search in main memory databases*, Proceedings of the 1992 ACM SIGMOD international conference on Management of data, **21**(2), DOI 10.1145/141484.130317.

8 **Bohannon Ph, McIlroy P, Rastogi R**, *Main-Memory Index Structures with Fixed-Size Partial Keys*, Proceedings of the 2001 ACM SIGMOD international conference on Management of data SIGMOD '01, **30**(2), DOI http://doi.acm.org/10.1145/375663.375681.

9 **Rajamani R**, *Oracle Database 11g: Secure Files – An Oracle White Paper*, (June 2007).