IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 15, 2017, accepted May 21, 2017, date of publication May 25, 2017, date of current version June 27, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2708080

A New Method for Time-Series Big Data Effective Storage

MAHMOUDREZA TAHMASSEBPOUR, (Senior Member, IEEE)
Department of Information Technology Engineering, Islamic Azad University–North Tehran Branch, Tehran 1969633651, Iran

ABSTRACT Today, one of the main challenges of big data research is the processing of big time-series data. Moreover, time data analysis is of considerable importance, because previous trends are useful for predicting the future. Due to the considerable delay when the volume of the data increases, the presence of redundancy, and the innate lack of time-series structures, the traditional relational data model does not seem to be adequately able to analyze time data. Moreover, many traditional data structures do not support time operators, which results in an inefficient access to time data. Therefore, relational database management systems have difficulty in dealing with big data—it may require massively parallel software that runs on many servers. This has led us to implement Chronos Software, an in-memory background-based time database for key-value pairs; this software was implemented using C++ language. An independent design has been suggested through appropriately using temporal algorithms, parallelism algorithms, and methods of data storage in RAM. Our results indicate that the employment of RAM for storing the data and of the Timeline Index algorithm for getting access to the time background of the keys in Chronos translate into an increase of about 40%–90% in the efficiency as compared with other databases, such as MySQL and MongoDB.

INDEX TERMS Timeline index, NoSQL database, Chronos database, big data, key-value database.

I. INTRODUCTION

Due to the rapid growth of data sets and the volume of information which need to be stored, a comprehensive management is required to preserve such massive amounts of information. Data production during recent years has grown to such a volume that relational banks can no longer control such data; moreover, today's data is by no means similar to the traditional data. Meanwhile, time datamanagement has assumed growing importance in many applications; among the database systems that already support the temporal dimension of data, few present temporal operators which are even of poor performance qualities. As a unified data structure, the *Timeline Index* efficiently supports time operators such as temporal aggregation, time travel, and temporal join. Owing to its independence of the physical sequence of data, the *Timeline Index* allows for flexibility in physical design.

II. BIG DATA

Big data refers to such a large or complex collection of data that it is difficult to handle using the present database management tools or the traditional data-processing applications. Relational database management systems and visualization packages cannot deal with big data—this may necessitate employing massively parallel software running on a large number of servers. The limit of big data depends on capabilities of the managing organization and on the applications used for data analysis. The relevant challenges include extraction, storage, search, sharing, transfer, analysis, and visualization of data [2].

III. REVIEW OF LITERATURE ON-TIME DATA

Today, temporal data management has become an important aspect of most databases. Modern database systems create a new version of an object instead of a local update. Having paid the extra costs of these new versions, the users expect rich capabilities for querying and analyzing the data. For example, users may want to compare the present status of their investment to one year ago, which is an instance of time travel—querying a historical version of the database. Though the time data management has been extensively researched into, the corresponding temporal database technology has not received the same level of attention from the industry. Clearly, the problem is not lack of market demand or interest; indeed, customers are very much interested in receiving enriched time features; for example, financial and sales–and distribution software developers implement temporal operators as part

of the application logic—such features are not supported by relational database products.

The problem of slow introduction of temporal features into the industry can be put down to technical reasons. According to the literature, the work done so far on temporal data management is highly specialized; that is, the index structures and algorithms have only been given for a specific temporal function. Therefore, the implementation of a new data structure for each type of temporal query is hardly affordable even for global players in the market—the research outputs cannot be generalized to other time data [2].

IV. ADVANTAGES OF USING NoSQL INSTEAD OF SQL

NoSQL enjoys a few advantages over SQL. Most importantly, there is no need for a rigidly defined schema for the data being inserted into a NoSQL database—the input data can be altered at any time, to which the database adapts itself accordingly. The second advantage is the automatic division and smart detection of integration. In SQL, the designer should implement the database schema according to the multiple-server state considered in his/her design, but multiple servers do not cause any problem in NoSQL systems, thanks to such an intelligent and high-level system. The third advantage of a NoSQL system over a SQL system is that the NoSQL caches the data in memory in order to speed up the recovery of information; that is, a NoSQL system stores useful data in the cache, just like a processor saving similar resources in the cache. Therefore, speeding the design and execution of databases, NoSQL is essentially useful for today and future data processing.

V. STATEMENT OF THE PROBLEM AND SIGNIFICANCE OF RESEARCH

One of the recent problems of modern saving systems is their scalability. Today, saving systems continually move towards NoSQL, a new generation of simple and scalable saving systems. Simultaneously, the preserving of time data for commercial applications has become very important. Chronos, which has been designed on the basis of scalability, is a document-oriented database which allows data to be separately distributed over multiple servers.

Though the scalability of Chronos is the main advantageous reason for discarding relational databases, there are other important merits in Chronos, too. Chronos is based on the concept of replacing the model of a row with a more flexible model that is named document. Being document-oriented, Chronos enables the user to create internal documents and arrays. Moreover, it is a schema-less database.

VI. AIMS OF THE RESEARCH

A temporal database contains data that is related to future or the scheduling of an application. Many of applications are temporal, among which are financial and banking applications, and scheduling applications such as controlling flights in an airport, trains schedules, etc.

Taking the above-given definitions into account, we have built a temporal database aimed at i) creating a scalable system to store time-related data, ii) presenting an effective solution for storing time data with the use of the Timeline Index data structure [1], iii) managing time data, and improving the analysis of time-series data, and iv) creating a system quickly responsive to the delays in traditional database systems.

We have realized our goals using the Timeline Index which is a data structure used for efficient management of time data and query performance on data.

VII. THE INNOVATIVE ASPECT OF THE RESEARCH

Here, we have simultaneously made use of the following three characteristics of Chronos storage software: i) the software can store the information on RAM, and not on the disk, just similar to an in-memory database such as RAM Cloud, ii) it is specific to key-value data like Redis, and iii) is a background-based database like Influx DB.Though these three features are not unique to Chronos, there is currently no single software that possesses all the aforementioned three qualities together. Therefore, for the first time, we have made use of Chronos to store time-series big data. The implementation of the Timeline Index, an innovative algorithm, in Chronos is a unique characteristic of our new method of storing time-series big data.

VIII. HYPOTHESES

Processing of time-series data forms a large part of the big-data research. Through high-speed saving and processing of time-series data, Chronos can play a very important role in processing big data, which has been established by our results presented later in this article. Time-series data are often in the form of key-value pair data and are inherently background-based.

Taking these hypotheses into consideration, we have replaced traditional database systems with a NoSQL modern string according to key-value, and have made use of the Timeline Index data structure for storing and managing time data, of RAM as the storage instead of the hard disk, and of the temporal table algorithm. Therefore, we have implemented a new system for storing time-series big data in this article.

IX. RESEARCH STRUCTURE

In the Introduction, big data has been defined, and the importance of time data has been highlighted along with some brief notes on its background. Then, some advantages of NoSQL over SQL have been pointed out. Thereafter, we have explained the problem along with its significance necessitating studying such a problem as the topic of a substantial body of research. This has then been followed by the section on innovative aspects of our study and the hypotheses. In the following section, the Research Method, the new data structure used in this research has been explained, followed by the traits, principles, architectural design, and the algorithm of data management in the third section. Section four is allotted to the discussion of the data and findings. Section five evaluates and compares the accidental data created by the part codes related to each comparable database. Finally,

section six ends the article with our concluding remarks on the work.

X. RESEARCH METHOD

A. CHRONOS DATABASE

Chronos is a background-based in-memory database for key-value pairs which we have implemented and developed in C++ language using the Time Index algorithm [1] of the Timeline Index in order for the database to be practically and functionally presented. This database uses hash tables for storing data, and the Timeline Index data structure and the Time Index algorithm for preserving history tables. The time-background character of Chronos indicates that the values stored in this database for each key are not discarded after alteration and reform, but are kept in the time history for future access.

Moreover, the in-memory state of this database means that all the key-value pairs and their backgrounds are stored and preserved in the main memory; that is, though the data is lost if the application stops running, the in-memory character of fail-safe systems allows for much faster access to the data as compared to disk-based databases.

In relational databases, the task of managing the time dimension of data is given to the programmers, which usually results in inefficient and erroneous applications. Furthermore, each given application requires its own specific solution; therefore, it can be argued that there is an essential need for building appropriate temporal databases.

B. OVERVIEW OF THE TIME INDEX

The Time Index is a data structure used for efficient management of time data and query performance on the data. In this research, we focus only on the time of the system; that is, different versions of similar items are stored; the location is not commonly updated, but a new version of the item is created each time. Therefore, it is possible to compare the current data with the previous data. The management of time data has become more important with the development of the web. Though such functional applications are in continuously increasing use, their corresponding supporting data structures are either mainly inefficient or only suitable for a specific inquiry being necessary for the system. The Time Index is independent of the physical aspect of data, therefore providing us with the flexibility in the physical design (schema design). The time data Index produces a predictable performance which is better than that of the other methods we have evaluated here. The three types of the queries provided by the Timeline Index are the temporal aggregation, temporal join, and time travel.

XI. THE COMPONENTS OF THE NEW DESIGN OF CHRONOS DATABASE

Fig. 1 shows the components of our new design of Chronos software and the relations of its constituent parts.

The components of the Chronos database are as follow:

1) The standard template library (STL) includes the structures and standard basic definitions of C++ programming language like strings, maps, vectors, etc.—all the main necessary parts to build a software;
2) The longer includes the definitions which are essential for writing timed-log messages in the terminal output, which helps the user with the follow-up process of responding to the requests submitted to the program, and with handling possible errors occurring in the course of executing the software;
3) Mongoose is a library written in C language and an embedded light HTTP server which can respond to many HTTP requests in a short period of time despite its low-volume code. Mongoose library does not independently respond to HTTP requests, and requires the notification of a program accountability mechanism through defining a call-back function which refers the request to the HTTP server; this will be further explained later;
4) The threading building blocks (TBB), a library written by Intel Company, is used for writing multi-core and parallel applications. The structures presented in this library are very similar to those given in the STL library, except that the TBB structures such as Concurrent HashMap can process parallel requests, guaranteeing the protection of data health and a suitable speed of execution. Among other features of this library is the necessary set-up provided for performing compound atomic operators whose execution is spontaneous according to the rest of the system. Atomic operators are among the most basic segments of parallel systems used in this research;
5) The HTTP server is the part of Chronos software which analyzes the HTTP requests received from Mongoose via the Callback, and then, transforms it into a language understandable to the other parts of the software. Since the HTTP is a text-based protocol, we need to extract the type of request and other necessary information before performing the request, which is the task of the HTTP server. After analyzing the request, the HTTP server transfers it to the Request Handler explained later. Moreover, after the analysis, the server produces an output and transforms it into a user-understandable language and format, and then, sends it to the user;
6) The request handler is the practical starting point for processing a request made by the user. After receiving the analyzed request from the HTTP server, the request handler calls the desired process from the temporal table according to the type of the request

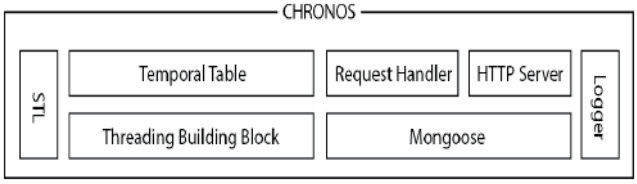

FIGURE 1. Our architectural design of Chronos database.

(get set, etc.), and then, returns the produced response to the HTTP server;

7) The temporal table is the core of Chronos software—where all the data is stored, and all the requested processes are performed. The temporal table is indeed an in-memory implementation of the Timeline Index algorithm. Since Chronos is designed for conditions with high access rates, this system requires very accurate access control to prevent the data from being destroyed; moreover, the accuracy of the access control becomes more crucial for an in-memory database to protect the data stored in the RAM. In the temporal table, this access control is achieved using the Concurrent HashMap structure data of the TBT library. This structured data is an implementation of the Open Hashing with partitioning, which allows for simultaneous accesses and reduction in the waiting time.

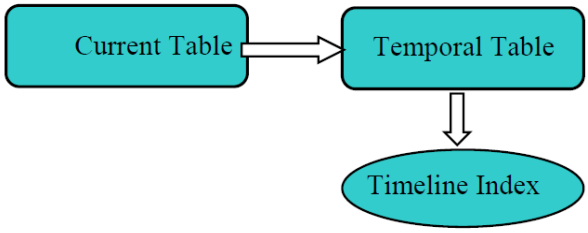

FIGURE 2. General Architecture of Timeline Index.

XII. GENERAL PRINCIPLES AND ARCHITECTURE OF THE TIMELINE INDEX

Fig. 2 shows how time data is managed [3]. For each table, the database keeps the current version of the table and separately preserves all the history of previous versions of the table in the structures. In other words, it is assumed that the current version is always the repeat timeline.

The current table allows for efficiently accessing the current situation of the database—this is the most important access for the user. The time features, for example, the time travel, are performed using the temporal table, and this is where the Timeline Index takes place—this is the index which accelerates the operation performed in the temporal table.

For each temporal table, there is exactly one timeline index. The temporal tables and timeline indices are the aims of this operation.

XIII. THE BUILDING OF THE TIMELINE INDEX

According to the design of the index presented in [1], defining an efficient production and also gradual updating is possible, even when the determined data is not in the order list. In [1], the BULK algorithm is first presented for specifically-ordered data and is then generalized to other cases. The repair and protection algorithms are based upon the counting sort algorithm [2].

The general time rank of this algorithm is linear compared to the temporal table, because it requires connecting only twice to each multiplex, once for counting the number of occurrences and once for writing the amount in the list of occurrences. The physical sequence of data is not relevant because the sequencing of the counter of the versions is done by a middle table.

Moreover, if the temporal table is arranged according to time, the index can be updated in a step-by-step fashion only through showing the new version and its corresponding event in the timeline index.

Finally, in contrast with the algorithms of the time data structures presented in [4]–[6], this algorithm lends its features to parallelism and distribution, based on the scan that states the difference in the versions.

The Timeline Index keeps the paths of all prints, updates, and omissions in the database. The time database requires tracking of the input validity with respect to the versions. Therefore, except the data kept in the database, the database keeps information such as 'from' and 'to' of each line referring to the time of the system related to the toppling. Two kinds of input exist, one for the opening of the field which means that it is valid for the current version, and the other for setting the field for example for those which are old.

XIV. ANALYSIS OF DATA AND FINDINGS

A. APPLICATION AND COMMANDS

Contrary to popular databases like Mongo DB and MySQL, each of which presents its particular exclusive user interface for recording, searching, and receiving data, Chronos is a powerful internal light HTTP server which presents an HTTP-based easy user interface. Use of a protocol like the HTTP makes it easier to produce and develop security and customer services. Furthermore, Chronos commands can be performed and tested even via a simple internet browser. The default port number on which Chronos HTTP server runs is 8008 which can be changed through entering argument –p on the command line.

The general format of all Chronos commands is as follows:
[host_address:port]/[command]?[query_parameters]

B. EVALUATION AND COMPARISON

To correctly evaluate the efficiency of Chronos software, we compare it to other databases. All the obtained time data were calculated using a computer featuring in 14.04 LTS Ubuntu operating system, Intel Core i5 processor with 2.6 GHz clock, 8 GB internal memory with 1600 GHz clock, and an external hard memory with 5400 RPM speed.

C. METHOD OF DATA TABLE CREATION

A suitable way for investigating the operation of databases is evaluating the sensitivity of their performance in lieu of increasing the number of data stored in them. It must be noted that time data is often high-frequency data, and Chronos software is no exception to the ability to support large volumes of data.

Each of the mentioned databases has been created with the tables of frequencies of 10 thousand (10k),

TABLE 1. Comparison of execution times of commands for inputting data in three types of databases.

Database	10K	100K	1M	10M
MongoDB	06s	59s	06m 21s	90m 05s
MySQL	04s	36s	05m 43s	61m 05s
Chronos	02s	20s	02m 35s	24m 53s

100 thousand (100k), 1 million (1M), and 10 million (10M) random data. Different methods were employed to create these tables in each of the databases under study owing to their different user interfaces, which will be briefly dealt with later.

D. CREATING RANDOM DATA IN CHRONOS

For example, Chronos supports HTTP requests for sending commands to users. Therefore, random data were entered by the following PHP code.

```
Function populate ($table, $num) {
$curl = curl_init ();
For ($i<$num; $i++) {
Curl-setup_array ($curl, array (
CURLOPT_RETURNTRANSFER=>1,
CURLOPT_URL=> 'http://localhost:8008/set?".
'key='. $table.
'&value='. (mt_rand () / mt_getrandmax ()),
));
Curl_exec ($curl);
}
Curl_close ($curl);
}
Populate ('t10k', 10000);
Populate ('t100k', 100000);
Populate ('t1m', 1000000);
Populate ('t10m', 10000000);
```

E. COMPARISON OF EXECUTION TIME

The times of commands for inputting the data have been summarized in Table 1. This data was obtained in MySQL using the Explain command, in MongoDB using the Mongo-Hacker software, and in Chronos using the Microtime function available in the PHP.

The results show that Chronos has a higher speed of data inputting compared to those of MySQL and MongoDB which are respectively 59% and 72%.

F. TIME OF COMMAND EXECUTION

Despite the importance of the speed of data inputting in a database, much more significance should be attached to the efficiency of the database in performing queries, for indeed the queries impose the highest load on the database. Therefore, it is highly important to investigate the efficiency of a database in performing queries.

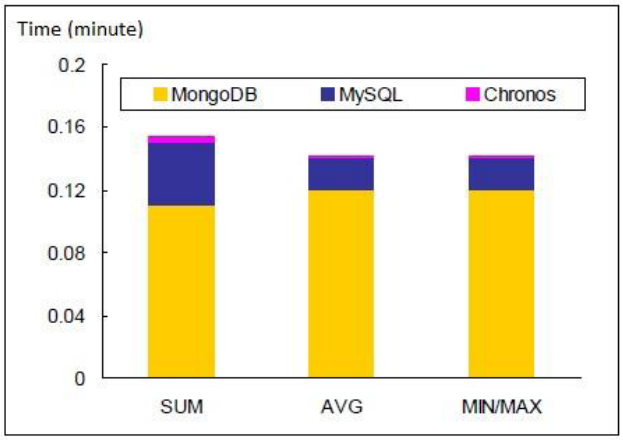

FIGURE 3. The times of execution of queries for the table with 10 thousand numbers.

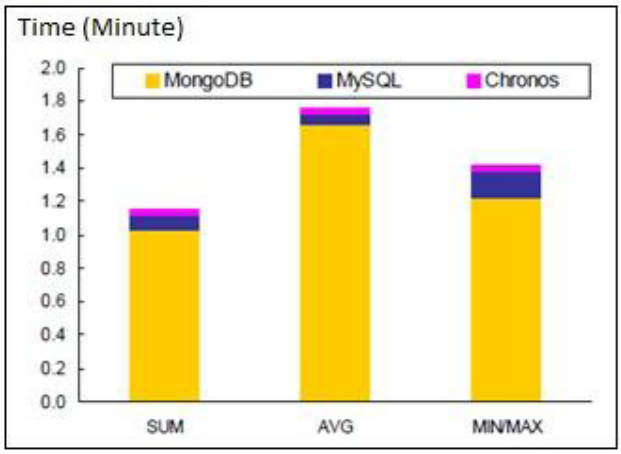

FIGURE 4. The time of execution of queries for the table with 100 thousand numbers.

Here, the efficiencies of previously mentioned software in performing the processed queries are compared, which is related to the time background of Table 1 (key).

XV. COMPARISON AND RESULTS

The time of execution of each of the SUM, AVG, and MIN/MAX commands at different frequencies for each database have been given in Fig. 3 to Fig. 6.

XVI. COMPARISON AND RESULTS

To present a new design for the storage of time data, we have directed our attention to factors affecting the storage, organization, and the speed-up, timely, and efficient analysis. With the development of the internet and the constantly growing volume of increasingly diverse data, previous storage systems are not capable enough to efficiently analyze the data, due to an increase in query time and a delay in the storage time. Therefore, it was a difficult task to state temporal questions using present languages like SQL. This all has necessitating lead us to research into a new design for a temporal storage software.

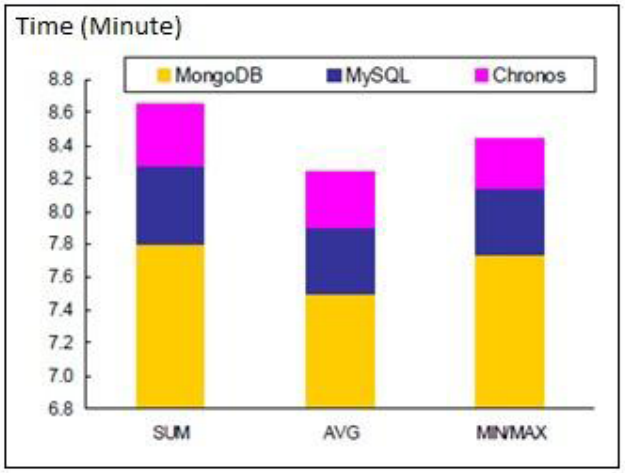

FIGURE 5. The time of execution of queries for the table with 1 million numbers.

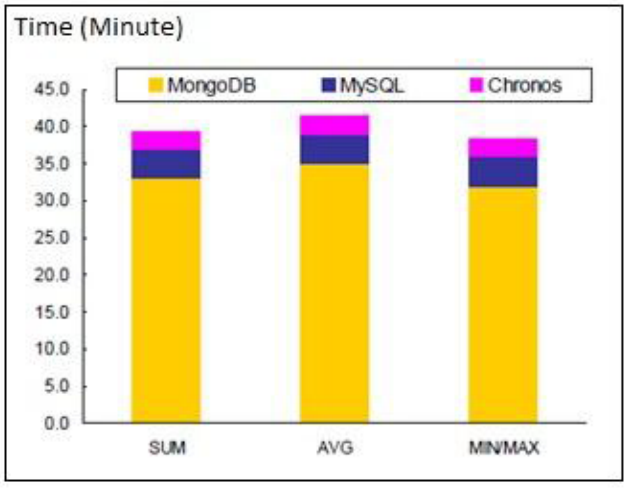

FIGURE 6. The times of execution of queries for the table with 10 million numbers.

Our research into a new design for storage of temporal data has led us i) to improve our understanding of concepts like big data, new generation databases, and their relation to big data, and of the effect of indexing on improving the performance of databases, ii) to review the performance of hardware components like RAM and hard disks, the use of parallelism algorithms that influence the speed factor, and the use of the Timeline Index data structure and iii) to learn to select an appropriate operating system for the implementation of different databases. Traditionally, there are two general methods for the implementation of temporal databases, one of which is to use the current implementations and to expand them, and the other is to convert the temporal language expressions to SQL using the present DBMS. Although the first method increases the efficiency of the system, the second method seems more practical even though it is more complex and redundant.

On the other hand, in designing a temporal database, the effect of time in all aspects must be taken into consideration. Among these effects are the management of temporal structures and the state of events, the variation of the relations, cursors and temporal views, the control of the database comprehensiveness, and the management of the present time.

Improvement of the query in temporal languages is much harder than the usual state. One reason is that the queries defined on temporal data are usually longer, and require more processing. The other reason is that improving temporal components is harder. These have been well researched into in order that we could arrive at a new design.

The idea of expanding databases such that they can deal with time data is not a new subject, but an exclusively independent design has so far been missing in the literature; we have built such a design, making use of temporal and parallelism algorithms along with the methods of data storage in RAM.

Our results indicate that employing RAM for storing data, and the time index algorithm for accessing the time background of the keys in Chronos software—though as minor and even not-technical-enough improvements, have to lead to the efficiency of the system increasing about 40% to 90% compared to the that of databases like MySQL and MongoDB which have been matured and kept by large companies.

REFERENCES

[1] M. Kaufmann, A. A. Manjili, P. M. Fischer, D. Kossmann, F. Farber, and N. May, "Timeline index: A unified data structure for processing queries on temporal data in SAP HANA," in *Proc. SIGMOD*, 2013, pp. 1173–1184.
[2] M. Tahmassebpour and A. M. Otaghvari, "Increase efficiency big data in intelligent transportation system with using IoT integration cloud," *J. Fundam. Appl. Sci.*, vol. 8, no. 3s, pp. 2443–2461, 2016.
[3] D. E. Knuth, "The art of computer programming," in *Volume 3: Sorting and Searching*. Reading, MA, USA: Addison-Wesley, 1998.
[4] G. Adel'son-Vel'skii and E. M. Landis, "An algorithm for the organization of information," in *Proc. USSR Acad. Sci.*, 1962, pp. 1259–1263.
[5] B. Becker *et al.*, "An asymptotically optimal multiversion B-tree," *VLDB J.*, vol. 5, no. 4, pp. 264–275, 1996.
[6] R. Elmasri, G. T. J. Wuu, and Y.-J. Kim, "The time index: An access structure for temporal data," in *Proc. VLDB*, 1990, pp. 1–12.

MAHMOUDREZA TAHMASSEBPOUR (SM'16) was born in Kerman, Iran, in 1975. He received the B.Sc. degree in electrical engineering from Islamic Azad University, Tehran, Iran, the M.Sc. degree in information technology engineering from the Iran University of Science and Technology, Iran, and the Ph.D. degree in computer science and information technology from the University of Malaya, Kuala Lumpur, Malaysia, in 2009.

In 2011, he joined the Educational Group of Iran, UAE, as a Lecturer. Since 2013, he has been with the Department of Electrical and Computer Engineering, University of Science and Culture, Tehran, where he is an Assistant Professor. Since 2016, he has been with the Department of Information Technology Engineering, Islamic Azad University–North Tehran Branch, Iran, where he is an Assistant Professor. He is a member of TPC as reviewer of some IEEE and international conferences and journals.