Developing of Node.js Module for Working with Effective Temporal Model in NoSQL Data Bases

Dimitar Pilev, Ventzislav Nikolov

Abstract— Ever growing user requirements to access information demands using of temporal database models. In such cases should be saved not only the data but the time intervals of their validity, as well. The subject of the article is developing of Node.js software module to process temporal data in the non-relational DBMS MongoDB. Asynchronous software model leads to extremely fast processing of large data amounts of temporal databases.

Index Terms— Temporal databases, Node.js, NoSQL database, MongoDB, asynchronous programming

1 Introduction

The continuous increase of the number of users utilizing the internet as a means of communication, payment, shopping etc. imposes the use of new technologies when implementing web-based applications. The applications that are being developed should process a large number of user queries towards the large amount of information stored in the databases.

One of the main requirements towards any contemporary web-application is related to its quick response time. The main objective is to decrease the response time of submitted user queries, regardless of their number and complexity.

At present there are a number of platforms for building web-based applications, such as the well-known Java EE [1], Apache PHP [2] and ASP.NET [3], as well as the new, but recently increasingly popular Node.js[4],[5].

The Node.js [6] platform is quickly gaining popularity amongst web developers thanks to the ease and speed of processing user queries. The use of the Node.js technology is increasing exponentially. Leading companies such as PayPal, LinkedIn, WalMart [7] have rewritten their applications using Node.js.

In a number of cases, the data base appears as the primary source of delay in processing user queries. Delay increases proportionately to the amount of processed data. This is the reason why leading companies such as Google, Facebook and Amazon are making the transition towards using a new type of databases called non-relational databases (NoSQL - Not only *Structured Query Language*) [8], [9].

The NoSQL database provides a mechanism for storage and restoration of data using a free, coordinated model of data unlike the relational model of DB. A non-relational database is an optimized depository containing information of the type key-value. Its designed purpose is to ease the processes of restoration and addition of information for the purpose of optimizing efficiency under the conditions of introduction and storage of large amounts of data.

- *Dr. Dimitar Pilev is currently lecturer at the Depatement of Computer Science, UCTM-Sofia, Bulgaria. E-mail: d_pilev@yahoo.com*
- *Dr. Ventzilav Nikolov is currently lecturer at the Depatement of Computer Science, UCTM-Sofia, Bulgaria. E-mail: nikolov_vnz@abv.bg*

Traditional models of DB and DBMS are capable of storing and processing only one present state of the modelled subject area. These are normally present states, whose old values are destroyed when change is necessary.

The constantly increasing demand of users for access to information imposes the use of temporal DB, where not only data is stored, but also the period of its validity. One major reason for non-reporting of the changing states of data over the course of time has been the lack of adequate maintenance of the temporal factor in DBMS. Until recently the existing DBMS would not allow temporal processing of data. Taking into consideration the extreme popularity of the problem, however, commercial DBMS such as Oracle and Teradata have already published [10], [11] new specifications of DBMS with temporal support. In order to implement temporal support, an additional surface layer has been developed in [12] to DBMS MySQL.

The purpose of the present development is to implement a module for the Node.js platform, allowing temporal support of data marked with effective time, stored in MongoDB database. The module allows the processing of queries related to addition, deletion, modification and searching of temporal information in the database.

2 Effective Temporal Model

The effective temporal model (ETM) is using effective time, representing a combination of valid and transactional time. The beginning of the effective period has been set at the current time at the moment of recording the cortege in DB (the beginning of the period marking the transactional time Ts), and its end has been set at the time marking the end of the period of the valid time Ve.

$$\textbf{\textit{insert}}\ (r, (a1, \dots, an), t'_e) = \begin{cases} r \cup \{(a_1, \dots, a_n \mid t_e)\}\ \textbf{\textit{if}}\ \nexists t_e\ ((a_1, \dots, a_n \in r) \\ \qquad \exists t_e\ ((a_1, \dots, a_n \mid t_e) \in r\ \nexists t \in \textbf{\textit{overlap}}(t_e, t'_e)) \\ r - \{(a_1, \dots, a_n \mid t_e)\} \cup \{(a_1, \dots, a_n \mid \textbf{\textit{change}}\textbf{ETP}(t_e, t'_e)\}\ \textit{if} \\ \qquad \exists\ t_e((a_1, \dots, a_n \mid t_e) \in r \wedge \textit{meet}(t_e, t'_e)) \\ r\ \textit{in opposite case} \end{cases}$$

a) Data add

delete (r, (a_1, … , a_n)) =
r - {(a_1, … , a_n | t_e)} U {(a_1, … , a_n | ***change*ETP**(t_e, t'_e)} *if*
$\exists\ t_e((a_1, \dots, a_n \mid t_e) \in r \wedge CT \in t_e)$
r *in opposite case*

b) Data delete

update (r, (a1, … , an), t'_e) =
insert (delete (r, (a_1, … , a_n)), (a1, … , an), t'_e***)***

c) Data modify

Fig. 1 Data manipulation in relation

The addition of data to the relation under ETM shall be implemented when we want to register facts that are not recorded in DB ($a_1, \dots, a_n$), which are valid for a definite period of time. The effective time shall coincide with the valid time from the moment of recording the facts in DB onwards. This means that facts are valid in the modeled reality and have been recorded in DB. Data recording should return a new updated version of the relation.

The following functions for work with effective temporal intervals have been defined in the model in order to present the semantics of data updating.

The function $meets(t_e, t'_e)$ shall perform a test for meeting of two intervals:

meets (t_e, t'_e)
if (E_e = E'_s)
return true
else
return false

The function $overlap(t_e, t'_e)$ shall perform a test for partial overlap of two intervals:

overlap (t_e, t'_e)
if ($E'_s < E_e$)
return true
else
return false

There are three cases when adding new data in the relation (Fig. 1a).

Case one: Let the values of attributes ($a_1, \dots, a_n$) , which do not depend on time, be unrecorded in the relation until now, or comprise part of its previous states (Fig. 1a). In that case there is no record where the effective period of time overlaps with the new period. In that situation we add a new cortege in the relation.

Case two: If the values of attributes ($a_1, \dots, a_n$) have been recorded in the relation and a record exists, where its period of validity t_e meets the new t'_e - the effective period, which is used to mark the cortege shall be updated to [E_s, E'_e).

Case three: If the values of attributes ($a_1, \dots, a_n$) are part of the current state of the relation (the effective period, through which data in the cortege is marked, is still active), then data modification is required, rather than data addition.

In order to update the values of data ($a_1, \dots, a_n$), which is part of the current state of the relation, the following functions is used:

changeETP(t_e, t'_e). This function changes the effective time period, through which data in the cortege is marked in the same manner regardless whether the old and the new periods meet or overlap(t_e and t'_e).

changeETP(t_e,t'_e)
if (***meet*** (t_e, t'_e) V ***overlap*** (t_e, t'_e))
return [E_s E'_e)
else
return 0

The deletion of a certain cortege consists of its logical removal from the current state of the relation (Fig. 16) The logical removal of the cortege is performed by correcting the time period, through which non-temporal attributes ($a_1, \dots, a_n$) are marked. Such correction is performed by setting the time for ending the effective time period, which time is current at the moment of deletion. For this purpose we use the function $delETP(t_e)$.

del (t_e)
if (CT ∈ t_e*)*
return [E_s CT)
else
return 0

If the values of attributes in the cortege ($a_1, \dots, a_n$) do not exist or, if they do exist but are not part of the current state of the relation, then deletion of the cortege would not produce an effect.

The modification of an existing cortege shall be performed through the "update" operation (Fig. 1c). It is defined as consecutive execution of the operations "delete" and "insert". The proposed ETM significantly reduces the volume of stored data. There is significant reduction in the excess of information stored in the DB. At the same time we can obtain a precise view of data validity over time.

3 Asynchronous Programming Model with Node.js

Node.js is a platform built on the basis of Google V8 JavaScript [13] for easy building of quick and scalable network applications. Node.js uses an "event-driven" asynchronous input/output model of operation using a single process, which makes it easy and efficient for real-time applications.

Node.js allows the development of applications written in JavaScript, which are executed by the server. The platform contains the most popular operation systems such as Windows, Mac and Linux. The environment provides the opportunity to interact with input/output devices through its API written in C++. This allows the connection with other external libaries written in different languages, by means of JavaScript code.

Node.js uses module module architecture to simplify the creation of complex applications. It contains built-in asynchronous input/output libraries for working with files, sockets and HTTP communication.

With traditional and well-known platforms of Java EE and PHP queries are processed by creating a separate thread or process for each query. Unlike them, Node.js implements an entirely different technology by operating with a single thread using asynchronous processing of queries (Fig.2).

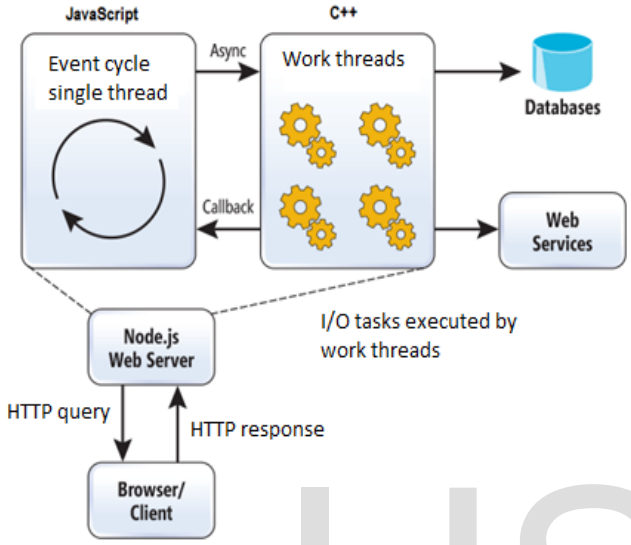

Fig. 2 Node.js event cycle

For this purpose the so-called event cycle is performed, which allows the processing of tenths of thousands simultaneous queries without concerns related to limited RAM memory and switching between separate threads.

The Node.js platform is only an environment, which means that the programmer must do everything by himself. By default there is neither HTTP, nor any other server. A single script contains the entire communication with users. This significantly reduces the number of resources used by the web application.

4 Data Model at MongoDB

MongoDB [14] is a document-oriented database management system, which stores structured information in JSON format [15] with dynamic scheme. This makes the integration of information in certain applications much easier and faster. MongoDB has gradually become one of the most popular non-relational DB, often referred to as NoSQL.

TABLE 1.
TERMINOLOGY USED WITH MONGODB

SQL	MongoDB
Database	Database
Table	collection
Row	Document
Column	Field
Index	Index
Connection of tables	Built-in documents
Primary key	Primary key

Table 1 shows the terminology used with MongoDB compared to relational.

Data in MongoDB is stored in the form of documents set in JSON format (Fig.3). Within each separate document, the value of a certain field may be of random type, including another document, data set or data sets.

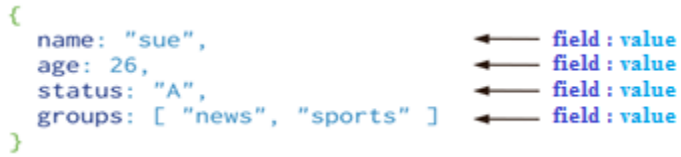

Fig.3 MongoDB document

MongoDB stores all documents in collections. A collection (Fig. 4) is a group of related documents having a set of shared common indexes. Collections are analogous to tables in relational databases.

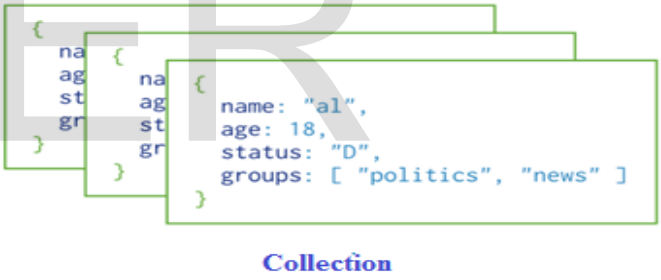

Fig.4 MongoDB collection

The advantage of MongoDB is that it stores information for

```
{
    "id": "1001",
    "customer":{
        "firstname": "Ann",
        "lastname" : "Smith"
    },
    "items":[
        {
            "description": "PC Dell Optiplex 760",
            "quantity": "1",
            "unitprice": "580",
            "amount": "580"
        },
        {
            "description": "HP laserjet 1020",
            "quantity": "2",
            "unitprice": "150",
            "amount": "300"
        }
    ],
    "payment":{
        "card": "visa",
        "ccnimber": "546258",
        "expiry": "04/2015"
    }
}
```

Fig.5 Storage of data regarding an issued invoice with MongoDB

a particular object in a single document, instead of dividing and storing it in several tables, which is typical for the relational DB model. Unlike relational DB, MongoDB allows data for a particular invoice to be stored in a single document (Fig. 5). A disadvantage of non-relational DB, however, may be the lack of, or partially minimum support of connections. The functionality of extracting information stored in several different collections must be performed by the application.

DBMS of MongoDB uses an asynchronous model for processing of queries submitted to the database, which significantly increases the efficiency of the system when working with large data sets. This characteristic makes it suitable for storage and processing of temporal data.

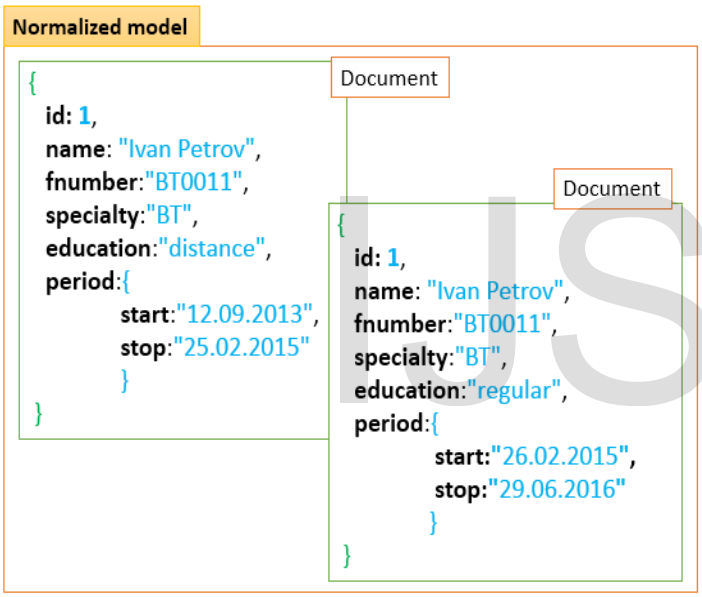

Fig.6 Storage of data under the temporal model supported by the module

5 Design and Implementation of Module for Working with ETM under MongoDB

During the development of the module, a normalized data model was selected where each document from the collection is marked with effective time. For this purpose an additional field (characteristic) "period" was added to document data, whicih sets the beginning and end of the effective time period marking the document (Fig.6).

The first step when developing a Node.js module is the creation of the file package.json. This file provides information about the functionality of the developed module.

Fig.7 shows the contents of the package.json file used to describe the developed module. The first two rows contain the mandatory fields for each Node.js module - name and version. Followed by description, keywords and name of developer. Node.js interface MongoDB is used in order to perform a connection MongoDB. The interface is included in the developed module as dependency (Fig.7, row 16).

```
1  {
2      "name": "mongodbETM",
3      "version": "0.0.1",
4      "description": "Temporal data model support for MongoDB",
5      "main":"./lib/index.js",
6      "repository": {},
7      "readme": "# Temporal support for MongoDB ",
8      "keywords":[
9          "Temporal MongoDB Support",
10         "Effective Temporal Database Model for MongoDb",
11         "Temporal Support for MongoDB Driver"
12     ],
13     "author":{
14         "name":"ME"
15     },
16     "dependencies": {
17       "mongodb": ">=2.0.0"
18     },
19     "engines": {
20         "node": "*"
21     },
22     "license": "MIT"
23 }
```

Fig.7 Contents of the package.json file, describing the developed module

The functionality of a module is distributed in the following functions:

- connect(mongoUrl, collection, callback) – for establishing a connection to the MongoDB server of DB. The submitted input data consists of the parameters of connection to MongoDB server, collection and callback function, whicih starts after the connection is established;
- insert(insObj, pe, callback) – for addition to the collection of data marked with effective time. The execution of the function requires data added to the collection (in the form of a document), the end of the time period marking the data, and a callback function.
- remove(removeObj, callback) – for deletion of data from the collection. Two parameters are set: criteria used to perform the deletion and a callback function typical for the asynchronous programming model;
- update(oldObj, newObj, pe, callback) – for modification of data in the collection. Criteria is set for update, new values, end of the marking time period and a callback function;
- find(query, options, callback) – for extraction of temporal data from the collection. The "query" parameter sets criteria for extraction of data from the collection. The second parameter "options" is optional and sets additonal settings such as sorting, restriction of the number of documents returned by the query etc. The callback function does the processing of the query result;
- close() – for terminating a connection to the MongoDB server of DB.

All operations related to updating and/or extraction of data are performed on a single collection.

```
var update = function(oldObj, newObj, pe, callback){
```

```
            remove(oldObj,function(res){
              if(res == 'Done' || res.result.ok==1)
                 insert(newObj,pe,callback);
              else
                callback('Cannot update collection!');
            });
        }
```

The implementation of the "update" function is in compliance with the requirements of ETM and the asynchronous programming model of Node.js In an ETM operation, "update" is restricted to consecutive execution of the operations "delete" and "insert" In order to observe the sequence of execution of both operations, theh addition of new data shall be performed in the callback function of the "remove" function. This guarantees that new values of data are going to be added only after the deletion of existing ones.

```
var etm = require("./lib/index.js");
var mongoUrl = 'mongodb://127.0.0.1:27017/test';
etm.connect(mongoUrl, 'proba', function(){
  console.log('Connected correctly to server');
  etm.insert({x:1,y:2}, new Date(2016,2,5),
        function(res){
          console.log(res.result);
          etm.find({}, function(docs){
            docs.toArray(function(err,data){
              console.log(data);
               etm.close();
            });
          });
        });
});
```

Fig.8 Usage of the developed module

6 Conclusion

The present article introduces a new platform for building network applications - Node.js. The environment allows quick and effective processing of user queries thanks to their asynchronous processing. A review is performed on the advantages and disadvantages of one of the most popular non-relational DBMS MongoDB. A module has been developed for the Node.js platform, which allows the processing of temporal data in an environment of MongoDB. The module allows the update and extraction of data marked with effective time. Asynchronous software model leads to extremely fast processing of large data amounts of temporal databases.

References

[1] Oracle, Java EE Documentation, available: http://www.oracle.com/technetwork/java/javaee/documentation/index.html
[2] The Apache Software Foundation, available: http://www.apache.org/
[3] PHP, PHP Documentation, available: http://php.net/docs.php
[4] ASP.NET, Get Started with ASP.NET, available: http://www.asp.net/get-started
[5] NodeJS, available: http://nodejs.org/
[6] JSConf Berlin,The European JavaScript Conference, Berlin, November 7 & 8, 2009, available: http://jsconf.eu/2009/
[7] WalMart, available: http://www.walmart.com/
[8] Pramod Sadalage, NoSQL Databases: An Overview, ThoughtWorks, October 2014
[9] Nosql Database, availeble:http://nosql-database.org/
[10] Database, O., Workspace Manager Developer's Guide. September 2010.
[11] Database, T., Temporal Table Support. Teradata Labs, 2012
[12] DimitarPilev, AnetaGeorgieva, Effective Time Temporal Database Model, International Journal on Information Technologies and Security, 2012. N2: 33-46, ISSN 1313-8251
[13] Google V8, V8 JavaScript Engine, available: https://code.google.com /p/v8/
[14] MongoDB, available: https://www.mongodb.org/
[15] Introducing JSON, available: http://json.org/