International Journal of Computer Science and Software Engineering (IJCSSE), Volume 4, Issue 3, March 2015
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 88-92

The art of fast web: Increasing Database Performance for RESTful Applications

Katalina Grigorova[1] and Kaloyan Mironov[2]

[1, 2] Department of Informatics and Information Technologies, University of Ruse, Ruse, Bulgaria

[1]kgrigorova@uni-ruse.bg, [2]mr.kaloyan@gmail.com

ABSTRACT

Using web based applications has increased considerably in recent years and is becoming the main way to use information services. Although emerging technologies, speed of a RESTful system has vulnerabilities and most often it is the database. This article traces and analyzes the best practices to increase efficiency in working with a database, and proposes a method that allows synchronized data caching.

Keywords: *REST, RESTful, Web Based Application, Database.*

1. INTRODUCTION

Use of the database is as an indispensable component of any modern application. In the world of RESTful services the database takes central place in the architecture of any REST-oriented system. Unfortunately, it exactly appears to be the weakest part of the system and requires good planning and implementation, in order to prevent future problems. The nature of web-based applications, and in particular RESTful, provides for the exchange of high traffic data between server and client, and the standards require this to be done quickly. Despite the rapid development of technology related to the processing and storage of data now the database remains the slowest component of REST architecture and if it have not received attention, may become a bottleneck. The article examines various techniques that aim to improve the speed of data transfer and to reduce the workload of the servers used. The caching is used as technology for rapid delivery of data in creation of a system that allows synchronized data caching

2. METHODS FOR DATABASE OPTIMIZATION

Although they are very convenient to store different types of information databases are not fast enough and since the people use them for web-based applications, always looking for ways to increase their productivity. Unlike standard desktop applications in RESTful services the database is subjected of "stress" test, because it should handle many concurrent requests. The article does not address the use of a specific database. The considered methods for optimization are applicable to all relational databases, and they are under study because currently are the most common in the creation of RESTful applications.
Over time, three main directions that help to optimize and improve the database performance can be identified. The process begins with an analysis of requirements, continues with the database design and ends with the creation of queries that allow fast retrieval of required information.

2.1 Preliminary analysis of the information

This phase of the database design is important for the entire system's performance. Knowing the information and the ways in which it is used, allows delineating logical groups of elements as well as the relationship between different groups. An important step of the information analysis is to determine exactly which data is necessary for the operation of the system and which can be omitted. This may significantly reduce the volume of information exchanged, it is limited only to the most needed, and that similarly decreases as well the time taken to transport data within the system.
Another important aspect of the data analysis is the differentiation them into logical units, which is done through the creation of separate tables. It is extremely important individual parameters of each logical unit to be correctly presented using notations for describing the types in the database. This requires experience and excellent knowledge of the data types in specific DBMS, and proper implementation significantly reduces the processing time of applications and increases database performance. A typical example of poor data definition and storage is a string of 5 characters in the variable of type Text, instead of Char (5). In the next two parts ways of presenting data

in tables that allow quick retrieval and creation of "fast" queries are discussed.

2.2. Tables design

A good design of the database tables may significantly improve the whole system's performance. Over time, three basic rules that is recommended to comply when create RESTful applications are established: the use of indices, creating static tables and sharding (horizontal partitioning the tables).

- Using fixed length (static) tables

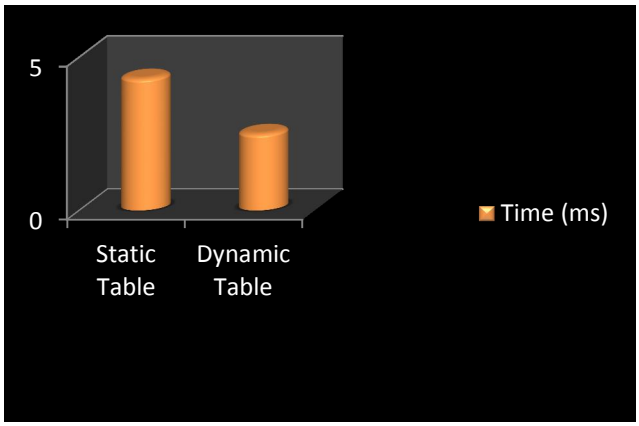

Fig. 1. Query execution in static and dynamic table

```
mysql> SELECT count(*) FROM `users` WHERE last_name LIKE 'az';
+----------+
| count(*) |
+----------+
|    63285 |
+----------+
1 row in set (0.25 sec)

mysql> ALTER TABLE `users` ADD INDEX ( `last_name` );
Query OK, 1009024 rows affected (17.57 sec)
Records: 1009024  Duplicates: 0  Warnings: 0

mysql> SELECT count(*) FROM `users` WHERE last_name LIKE 'az';
+----------+
| count(*) |
+----------+
|    63285 |
+----------+
1 row in set (0.06 sec)
```

Fig. 2. Performance after index adding

If each column in the table has a fixed length, the table is called static. The tables of this type are processed much faster because DB engine searches among the entries more quickly. If a specific record in the table should be read, then its location is calculated significantly faster thanks to the fixed values. Another advantage of static tables is that they can easily be cached and in the case of crash can easily be recovered. Fig. 1 shows the result of the implementation of the same query in a static and dynamic table.

- Using index fields

Use of indices could be applied not only for key fields [1]. Adding them helps to faster data retrieval and creates a tree structure of the elements of certain field in the table. Thus, the time of searching an element is reduced significantly. Fig. 2 shows the time difference in performing the same queries in the same tables at presence and absence of index fields.

- Sharding

This is a technique that allows a single table with a lot of records to be divided into separate, related tables with the smaller number of elements in each [2]. Searching for a specific entry in horizontally partitioned table becomes parallel in all individual shards of the table, and this considerably reduces the time for detection and delivery of data. An example of horizontally partitioned table is presented in Fig. 3.

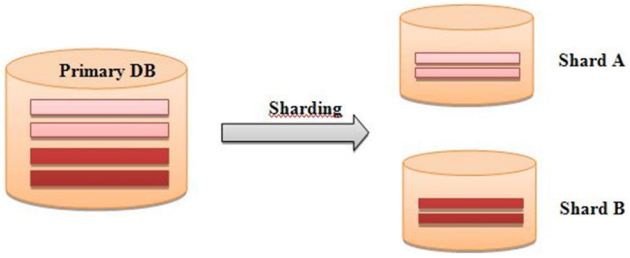

Fig. 3. Example of horizontal database sharding

2.3 Queries design

An important aspect of the rapid data delivery is the way in which they are extracted from the database. While design of tables provides a repository for fast delivery of information, the queries are those that determine what and how they will be processed. Design of fast and efficient queries is extremely important for the speed of service in every RESTful application [3]. Some of the best practices for creating queries are:

- Avoidance the use of functions in the query

The queries are commands, that "says" the server what to do. Bad, but often used in practice, is the addition of functions in the body of the query. In the performance of such a query, the function in it often requires more processing time than is necessary for the execution of the rest of the query, and this unnecessarily prolongs the extraction of data. Another important aspect that requires the use of static queries is caching the result. When using query without function in it, the server will cache the results of its implementation and in the case of recalling the query, the data will be "taken" from the cache, and will not be searched in the database.

- Using LIMIT when possible

Often, when performing a query to the database table only one result is expected. In such cases, it is recommended that in the query to use LIMIT1. Thereby increasing productivity, because, the server will stop searching the database when the first result is found, and this reduces the query execution time.

- Avoidance of SELECT *

Another common mistake when designing queries is retrieving all fields for a specific record by SELECT *. The relationship between the amount of data read from the table and the speed is proportional. Good practice is to extract only the information that is needed. This shortens the query execution time and reduces the volume of data to be transferred between servers.

2.4 Caching and use of dynamic memory

There are other "tricks" to improve the speed of the information layer in a web-based application. Some of them are very effective, but can not be applied in all cases of use.

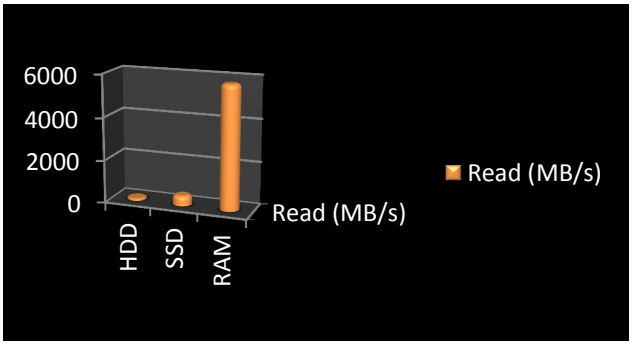

Fig. 4. Measurement reading speed of different memory types

The main problem that makes databases slow is the fact that information is stored on media that are relatively slow in reading data [4]. In most cases, these are standard HDD or SSD. The use of cash allows the result of a query to be stored in the dynamic (RAM) memory and when recall the query, the data are retrieved from there, which is much faster. Fig. 4 presents a comparison in speed of reading between HDD, SSD and RAM.

As can be seen from the figure, when operating in the memory, the speed is extremely high. Problem that exists when caching is the fact that information can be updated and what is stored in the cache can become obsolete.

Another option for use of the dynamic memory is the location of the entire database in RAM memory. This would solve the problem with speed, but is not practical, because for large databases high volume of memory is required. Another problem of this configuration is that the RAM memory is volatile, and in the failure, all information will be lost.

Solution to this problem is to create a caching mechanism that takes advantage of dynamic memory, monitors the actuality of cached data and is provided with an updated copy of non-volatile memory.

3. Concept for RAM-based, synchronized, key system for data caching

A system to cache the results of previously executed SQL queries is designed. While repeat the call, data is extracted from the fast dynamic memory without the need "to turn" back to the SQL server. What distinguishes the application of other existing solutions is the fact that the system monitors the actuality of cached information.

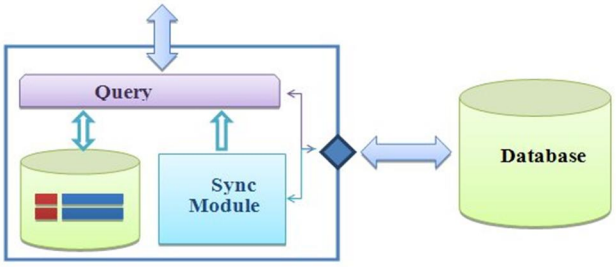

Fig. 5. In-Memory Synchronized Key-based Database Caching System

The Fig. 5 shows that the caching system is composed of three main modules: Cached results repository, Query processor and Synchronization module. Repository serves to keep the already implemented queries and works directly with dynamic memory, which makes writing and reading information from it very fast. Storing data in it is performed using hash table, and the unique key, which serves as an identifier of each record is used. To associate a record (result of a query) with the query itself, for the key is used MD5 encryption of SQL query. Thus a record of hash table has a key that contains the query, the second field of the hash is the result of the query execution. In this way the creation of an additional structure that associates keys with queries and results is avoided.

The query processor has several functions. First, it is assumed that the repository for cached results is empty. Upon query to the SQL server query processor first checks whether the requested information is available in the cache. If it is available, the data is forwarded to the user, thereby avoiding slow communication with SQL server. If the requested information is not available in the repository, SQL query is encrypted with MD5 algorithm and saved as a key in the hash table. The query is forwarded to SQL server, the response is recorded to already established key and forwarded to the user. Thus, the instance of a query that is not available, it is cached.

Synchronization module serves to maintain the relevance of the cached results. The module acts as a dynamic trigger and monitor changes in database fields that build the

cached results. Upon the occurrence of such changes, the module signals the query processor, which in turn recompiles the query and saves the actual outcome. In this way the actuality of cached data in the repository is guaranteed. Another major feature of the synchronization module is to create and maintain a current copy of the repository on the energy independent memory. Thus, in the failure, after restart the system, all cached results are loaded into the repository and dynamic memory.

4. Testing and analysis of results

The testing is designed to compare the time for delivery of information using RESTful application running with the caching system and in the same application, communicating directly with the SQL server. The tests are made by running identical SELECT queries, and the results of the two scenarios are compared. For objectivity of tests various complex queries that process data of a different number of tables are performed.
The testing is designed to compare the time for delivery of information using RESTful application running with the caching system and in the same application, communicating directly with the SQL server. The tests are made by running identical SELECT queries, and the results of the two scenarios are compared. For objectivity of tests various complex queries that process data of a different number of tables are performed.

Table 1: Benchmark query caching

Query type	TnC(t)	TwC(t)	*Performance*
Single table query	25904 ms	5886 ms	77.28%
Multiple table query	54721 ms	6182 ms	88.7%

Table 1 shows the average values of the SELECT query execution, comprising the results of one and several tables in a particular database. The measured parameters are the time of data delivery without caching - TnC(t), the time of data delivery with caching TwC(t), and the percentage representation of the level of optimization.
The reported experimental results are graphically presented in Fig. 6. From the image is seen significant improvement in the speed of information delivery on both types of queries. The average value, which improves the information delivery is 83%. What is interesting and is overseen by the experiments is that regardless of the TnC (t), TwC (t) maintains a relatively constant level. This is mainly due to the hardware configuration that supports caching application and values TwC (t) are directly linked to it.

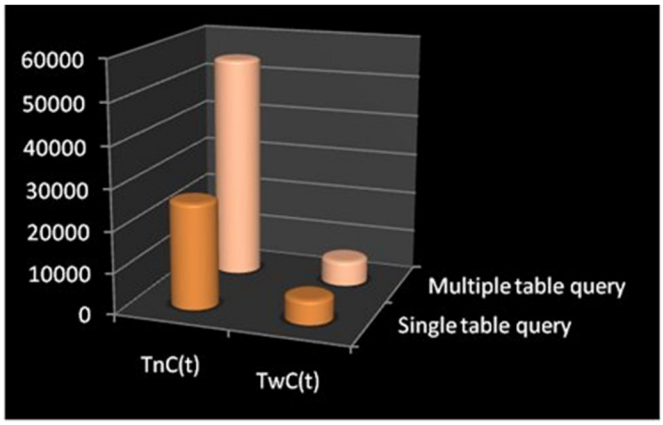

Fig. 6. Benchmark results

5. Conclusions

The use of caching in the context of RESTful and work with databases significantly contributes to improving the entire application's performance. The article discussed best practices for optimizing database with the aim of rapid retrieval of information, presented in a concrete solution for caching data that supports their relevance. The results of the experiments show the effectiveness of the developed application, and support the idea of using the dynamic mechanism for caching data.

REFERENCES

[1] Ullman, J. D, H. Garcia-Molina, Database Systems: The Complete Book, Prentice Hall, 2008, ISBN-10: 0131873253
[2] Roy, R., Shard – A Database Design, 2008
[3] An Oracle White Paper, Query Optimization in Oracle Database10g Release 2, 2005
[4] Lee, S., B. Moon, C. Park, Advances in Flash Memory SSD Technology for Enterprise Database Applications, EECS 2014

AUTHOR PROFILES:

Katalina Grigorova is an Assoc. Prof., Head of Department of Informatics and Information Technologies at University of Ruse, Bulgaria. She received MSc degree in Applied Mathematics from Moscow Power Engineering Institute and PhD degree in Computer Aided Manufacturing from University of Ruse. Her research interests include Business and Software Architectures Modeling, Business Process Modeling, Automated Software Engineering, Databases, Data Structures and Algorithms Design, Programming. Dr. Grigorova is a member of Association of Information systems (AIS) and its Bulgarian chapter BulAIS. She is a winner of IBM Faculty Award.

Kaloyan Mironov is a PhD student at Department of Informatics and Information Technologies at University of Ruse, Bulgaria. He received MSc degree in Software Engineering from University of Ruse in 2012. His research interests include Business Process Modeling, Computer Networks, Servers, Algorithms Design, Programming, and Robotics. He is a winner of The best paper award of scientific conference CompSysTech'2010.