The Creation of Scalable Tools for Solving Big Data Analysis Problems Based on the MongoDB Database

O.I. Vasilchuk[1], A.A. Nechitaylo[2], D.L. Savenkov[3], K.S. Vasilchuk[4]

[1]Volga Region State University of Service, 4, Gagarin st., Togliatti, Russia, 445677
[2]Samara National Research University, 34, Moskovskoye shosse, Samara, Russia, 443086
[3]Samara State University of Economics (SSEU), 141, Sovetskoi Armii st., Samara, Russia, 443090
[4]National Research University of Electronic Technology (MIET), Bld. 1, Shokin Square, Zelenograd, Moscow, Russia, 124498

Abstract. This article presents analyze of using MongoDB database to storing and effective data mining from open network sources. This paper attempts to use NoSQL instead of traditional SQL database in systems with strongly related information, comparing to relational and non relational approach in the performance and architecture.

Keywords: MongoDB, cloud services, architecture scaling.

1. Introduction

Modern data storage technologies provided a practical opportunity to accumulate huge amounts of information, which allowed a qualitative change in the attitude to the results of analysis of stored information. It became possible to move from a descriptive process of analyzing the results obtained over a certain period of time to predictive data processing technologies that make it possible to offer valid recommendations for the future. The using of relational databases (MySQL, PostgreSQL, Oracle Database and others) to solve large data storage problems becomes problematic. The main advantage of relational databases is the availability of techniques for maintaining data integrity, achieved by storing links between data elements. However, the storage and validation of these links requires additional time resources, which, with significant amounts and poor data structure, makes the use of relational databases difficult in some real-time systems.

As an alternative, there is a NoSQL database. One of the advantages of these databases in alternative storage formats and the links between them.

2. Alternative Data Storage for Unified Text Formats

2.1. The Problem Formulation

With the development of metaprogramming, the concept of reflection developed – the ability of the program to use and modify its structure[1]. Reflection, in the context of object-oriented programming, spread the technique where objects created by programs, based on knowledge of the structure of a class, are serialized into representations of given formats[2]. Most often this technique is used in the context of web programming, when data is serialized to xml, json and other formats for text data transmission.

This led to the task of filtering such data across different data fields of text formats, including using standard filters (for example, XPath[3]).

A common problem is the problem of data storage, the final form of which is some unified format (hereinafter referred to as UF)[4]. Since a web application user (usually a client application) works only with UF, and internal data views are not available for it, the filters available to the client are reduced to UF fields.

In classical relational databases, storing such formats is associated with the creation of several linked tables and subsequent cross queries. To speed up such queries, indexes are created for the corresponding keys.

Consider an example of a web service that provides access to the UF type JSON for some class "book":

Consider definition of scheme for UF:

```
{
  "definitions": {},
  "$schema": "http://json-schema.org/example/schema#",
  "$id": "http://itnt18.ru/itnt18.json",
  "type": "object",
  "properties": {
    "title": {
      "$id": "/properties/title",
      "type": "string",
      "title": "The Title Schema."},
    "description": {
      "$id": "/properties/description",
      "type": "string",
      "title": "The Description Schema."},
    "comments": {
      "$id": "/properties/comments",
      "type": "array",
      "items": {
        "$id": "/properties/comments/items",
        "type": "object",
        "properties": {
          "user": {
            "$id": "/properties/comments/items/properties/user",
            "type": "string",
            "title": "The User Schema." },
          "comment": {
            "$id": "/properties/comments/items/properties/comment",
            "type": "string",
            "title": "The Comment Schema." }}}}}
}
```

Listing 1. UF definition in JSON format

We assume that most often the book is filtered by title. And we assume that sometimes we just want to find a book, for example, for a search function on a web site, and sometimes

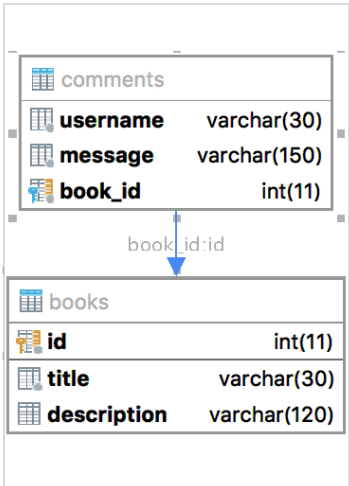

Figure 1. SQL data relations scheme

download a page with a book where besides the book and its description we also want to display all the comments.

It is common practice in the relational database to compile the following schema 1.

MongoDB use JSON-like format to keep data and can keep data in the output view.

2.2. Motivation for MongoDB

As an alternative to the classical approach, there are NoSQL databases. For tasks in which UF is involved, the most common use of document-based one.

As a working example, the authors used MongoDB. This choice is due to prevalence and testing in high load projects[5].

Based on benchmarking top NoSQL databases performance tests conducted by "End Point Corporation", the authors systematized MongoDB performance indicators for various hardware configurations, which were summarized in the table 1[6].

Table 1. Query time analysis results (operations per second)

Node numbers	Reading	Reading/Writing	Reading/Writing/Changing
1	2 149	1 278	1 261
2	2 588	1 441	1 480
4	2 752	1 801	1 754
8	2 165	2 195	2 028
16	7 782	1 230	1 114
32	6 983	2 335	2 263

The performance comparison experiment was conducted in the cloud services of "Amazon Web Services EC2", which provides an industrial platform for systems that require a horizontal extension of the architecture, such as distributed non-relational databases. In order to minimize the errors in measurements related to the current load of "Amazon Web Services EC2" services, each set of test scenarios was played three times, at least 24 hours apart, using newly created clusters with hardware configurations described in the table 2 of hardware configurations of cloud services "Amazon Web Services EC2":

Table 2. Hardware configuration of "Amazon Web Services EC2"

Node Class	Configuration	Application
i2.xlarge	30.5 GB RAM, 4 CPU, one SSD with 800GB	database nodes
c3.xlarge	7.5GB RAM, 4 CPU	database client nodes

As an operating system in the nodes used Ubuntu 14.04 LTS AMI in the HVM mode (virtual hardware virtual machine) virtualization, customized with Oracle Java 7. For each test, an empty database was used as the starting point. The client applications were programmed to enter randomly generated information into the database. After the database was finally populated, each of the test scenarios was executed sequentially. All clients performed requests in parallel, and then waited for all operations to be performed and the corresponding results obtained. The client software was supplemented by the installation of the YCSB free software package designed to analyze the performance of NoSQL databases[7]. The study of the company "End Point Corporation" allows you to determine the number of necessary nodes for placing MongoDB in the solution of certain business tasks, based on the anticipated load, when using a SSD drive in each machine-node. Nevertheless, despite its obvious competitive advantages, such as small size and weight, as well as the number of random IOPS that exceed by an order of magnitude the more common HDDs, the SSD is inferior to the latter by cost[8].

2.3. Preconditions

The SQL database with 5 millions entries for book instance and 10 millions for comments (two for each book entry) was used. And 5 millions entries for full (with comments inside view, also two comments on each book) book instance was used in MongoDB collection.

Three main scenarios were examined:

- Find by title
- Find by comment
- Find comments for the book

Since MongoDB stores the data immediately in the final UF view, it will be absolutely identical for searching the book and requesting an extended output.

Averaged values over 10 experiments are used to except the non-deterministic influence of external factors.

2.4. Results

Table 3. MongoDB and MySQL comparison (in seconds)

DB	MongoDB 3.4	MySql 5.4
Find by title	0.434527	1.646251
Find by comment	5.826707	1.387221
Find comments for the book	0.434527	4.360173

As can be seen from the results(Figure 2 and Table 3) of the research, the search query in the collection of the MongoBD is more effective than the search in the MySQL. The most effective scenario is to search for comments on the book, that is, additional information related to the main entity. The speed of work of MongoDB in this case surpasses MySQL almost 5 times. Nevertheless, it is important to understand that if the search target is additional information(comments here), the velocity of the query for MongoDB will be greatly worse then traditional relational databases.

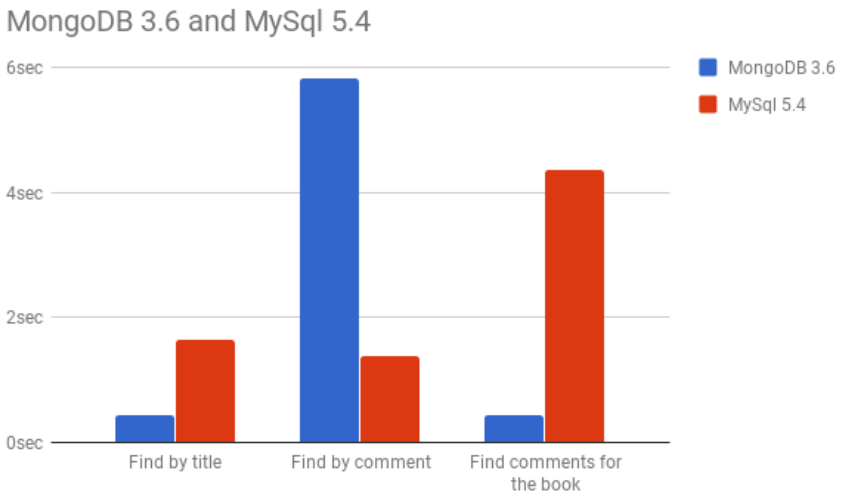

Figure 2. MongoDB and MySQL results comparison

3. Conclusion

The conducted research allowed to draw a conclusion about the expediency of using document-oriented databases for storing large amounts of data for indexing purposes with a small number of supported links or their absence. Feasibility is confirmed by the fact that the use of document-oriented databases for storing large amounts of data for indexing with a small number of supported links or their absence allows you to select the optimal configurations of computing systems in the framework of current and future business tasks, with the possibility of horizontal and vertical scaling, in conditions better performance than relational equivalents.

4. Reference

[1] Gregor Kiczales, J.R. The Art of the Metaobject Protocol / Jim des Rivires Gregor Kiczales, Daniel G. Bobrow. — MIT Press, 1991.
[2] JavaScript Reflect Global Object [Electronic resource]. — Access mode: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global Objects/Reflect (18.10.2017).
[3] XPath 3.1 Specification [Electronic resource]. — Access mode: https://www.w3.org/TR/xpath-31/ (18.10.2017).
[4] Charles Bell. Expert MySQL / Charles Bell. — APress, 2012.
[5] MongoDB Official site [Electronic resource]. — Access mode: https://www.mongodb.org (18.10.2017).
[6] Benchmarking Top NoSQL Databases [Electronic resource]. — Access mode: https://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL Benchmarks EndPoint.pdf (18.10.2017).
[7] Yahoo! Cloud System Benchmark (YCSB) [Electronic resource]. — Access mode: https://github.com/joshwilliams/YCSB (18.10.2017).
[8] Amazon EC2 Instance Types [Electronic resource]. — Access mode: https://aws.amazon.com/ru/ec2/instance-types/ (18.10.2017).