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] and K S Vasilchuk[4]

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

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 require 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 [5]. 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

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.

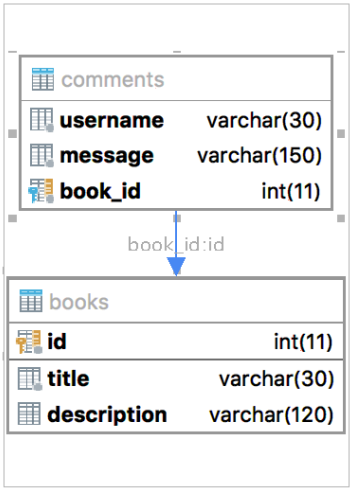

Figure 1. SQL data relations scheme.

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

2.2. Object-Relational mapping advantages

ORM or Object-Relational mapping is a programming technique designed to map database relations with object-oriented programming languages entities. It creates virtual objects database inside specific language representation.

The main the goal of technique is get rid of the need to write SQL queries to access database data. The dual way of data representation, relation and object-oriented, usually requires from programmers to write code for getting data from database in relation way, after transforming it to object-oriented, and transform back to relation data to safe changes. Relational databases operate over sets of tables with simple data representations, it leads to use SQL "JOIN" operation to get full object information. Since relational database management systems usually do not implement a relational representation of the physical link layer, the execution of several consecutive queries (referring to one "object-oriented" data structure) can be too expensive.

Relational database management systems work with good performance with global queries, affecting a large area of memory, but object-oriented access is more effective in the work with small amount of data, as it reduces semantic gap between the object and relational data representation[6].

Two way of data representation increases the complexity of object-oriented code to work with relational databases, it becomes more prone to errors.

2.3. Object-Relational mapping disadvantages

The most common problem with ORM as an abstraction over SQL is that it can't fully abstract realization details. Some of the program realizations of ORM works as SQL code generations tools, some of them do not use SQL equivalents at the external level.

The reason why abstraction make sense is simplification of code writing, but if you use ORM framework with knowing SQL as a requirement, it doubles programmers effort, for example the popular ORM framework Hibernate use HQL language SQL for complex requests, which is very

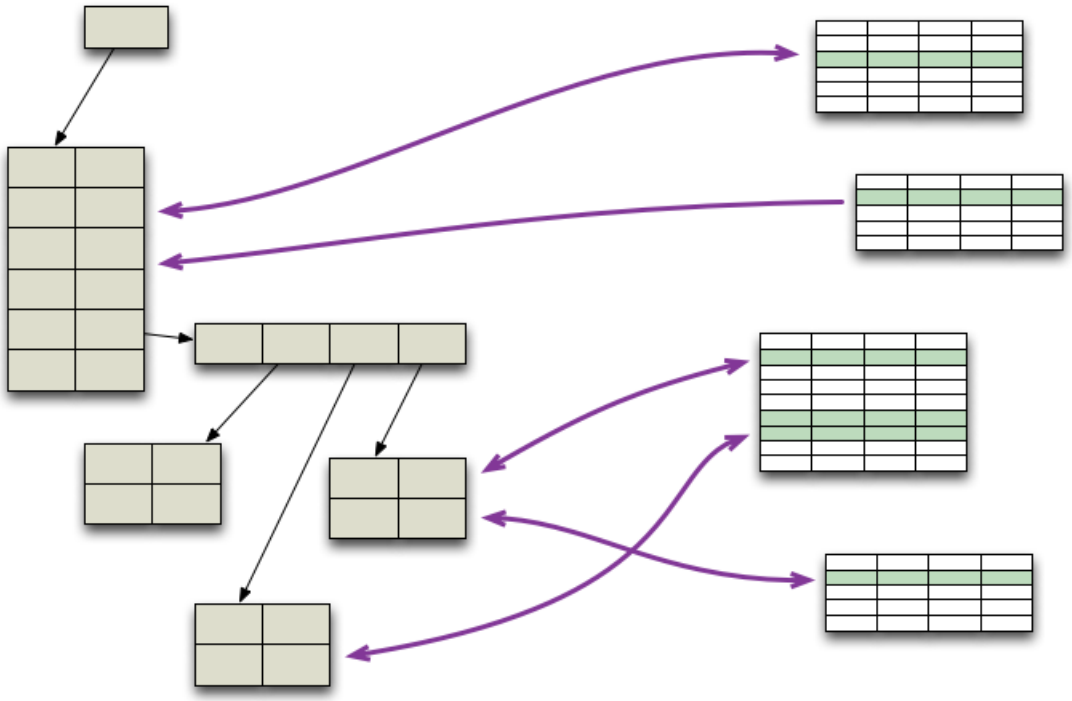

Figure 2. ORM data mapping.

semantically close to SQL. It brakes uniformity of the code abstraction when programmer needs a specific union of data processing.

Inefficiency is another common problem of ORM. If programmer need extract object data from relation database, ORM cannot know which of the object property are going to be used or changed, so it forced to extract all, it cause many requests instead few. The lack of context sensitivity means that ORM can't consolidate requests, which leads to the impossibility of data caching or other compensation mechanisms.

2.4. Object-Document mapping

ODM or Object-Document mapping is an alternative for ORM in document-oriented databases. The basic idea of ODM frameworks is the same, match data to the object, but we have few differences here.

Firstly, in ORM we should complete data for the object, and complete data for backward mapping inside database. In ODM there is no requirement for fully completeness of data. The document can be mapped partly to a database, without multiple table changing.

Secondly, ODM can make data mapping independently from the data source. It makes easier to operate over data in a program.

2.5. NoSQL databases

Relation databases usually based on ACID – Atomicity, Consistency, Isolation, Durability[7]. ACID is common requirement for transaction systems.

NoSQL databases usually based on BASE:

- basic availability – every request will be completed (successfully or not)
- soft state – The system state can be changed without any data changes due to data consistency
- eventual consistency – The data can be inconsistent for some time but will be consistent after a while.

It's obviously that NoSQL databases can not be used in any application. Some application are requires for transaction systems (like banking, ecommerce, etc), but at the same time usual ACID system doesn't suit for systems based on large data storages, like amazon.com and other. So NoSQL databases sacrifice data consistent to make more scalable system, to operate over large amount of data.

Also, NoSQL databases represent following features:

- Application of various types of storage facilities.
- Ability to develop a database without specifying schemas.
- Linear scalability (adding CPUs increases performance).
- Innovation: A lot of opportunities for data storage and processing

2.6. NoSQL databases common types

Unlike relation databases, NoSQL databases have various data schemas, implemented through the use different data structures.

Depending on the data schema and the approaches to distribution and replication, four types of storage can be distinguished: key-value store, document store, column database stores, graph databases.

Key-value storage

The key-value store is the simplest data store that uses the key to access the value. Such repositories are used to store media images, create specific file systems, as caches for objects, as systems well scalable by design. Examples of such storage facilities are Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB [8].

Bigtable-like databases (column database stores)

In this store, data is stored as a sparse matrix, the rows and columns of which are used as keys. A typical application of this type of database is web indexing, as well as tasks related to large data, with reduced requirements for data consistency. Examples of databases of this type are: Apache HBase, Apache Cassandra, Apache Accumulo, Hypertable, SimpleDB.

Column family stores and document-based repositories have similar usage scenarios: content management systems, blogs, event logging. The use of timestamp allows using this type of storage for the organization of counters, as well as the registration and processing of various data related to time.

The family column stores should not be confused with column stores. The latter are relational databases with separate storage of columns (in contrast to the more traditional line-by-line data storage)[9].

Document-based database management system

Document-oriented databases serve to store hierarchical data structures. They are used in content management systems, publishing, document search, and so on. Examples of this type of database are CouchDB, Couchbase, MarkLogic, MongoDB, eXist, Berkeley DB XML.

Databases based on graphs Graph databases are used for tasks in which data has a large number of links, for example, social networks, fraud detection. Examples: Neo4j, OrientDB, AllegroGraph, Blazegraph, InfiniteGraph, FlockDB, Titan [10].

Since the edges of a graph are materialized, that is, they are stored, traversing the graph does not require additional computation (like JOIN in SQL), but to find the initial vertex of the traversal requires the presence of indices. Graphical databases generally support ACID, and also have different query languages, like Gremlin and Cypher (Neo4j).

2.7. 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.

It has following advantages:

- It works with unstructured data, which make possible to add new data field with no additional cost.
- It available to compromise finding in performance-reliability.
- Working with ODM frameworks (Object-Document mapping) – alternative for ORM [11]. In cases of optional data it makes possible to map object without cross queries, make the mapping operation faster.
- The JavaScript support on server side.

As a working example, the authors used MongoDB. This choice is due to prevalence and testing in high load projects[12].

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[13].

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[14]. 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[15].

2.8. 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.

2.9. 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 3 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.

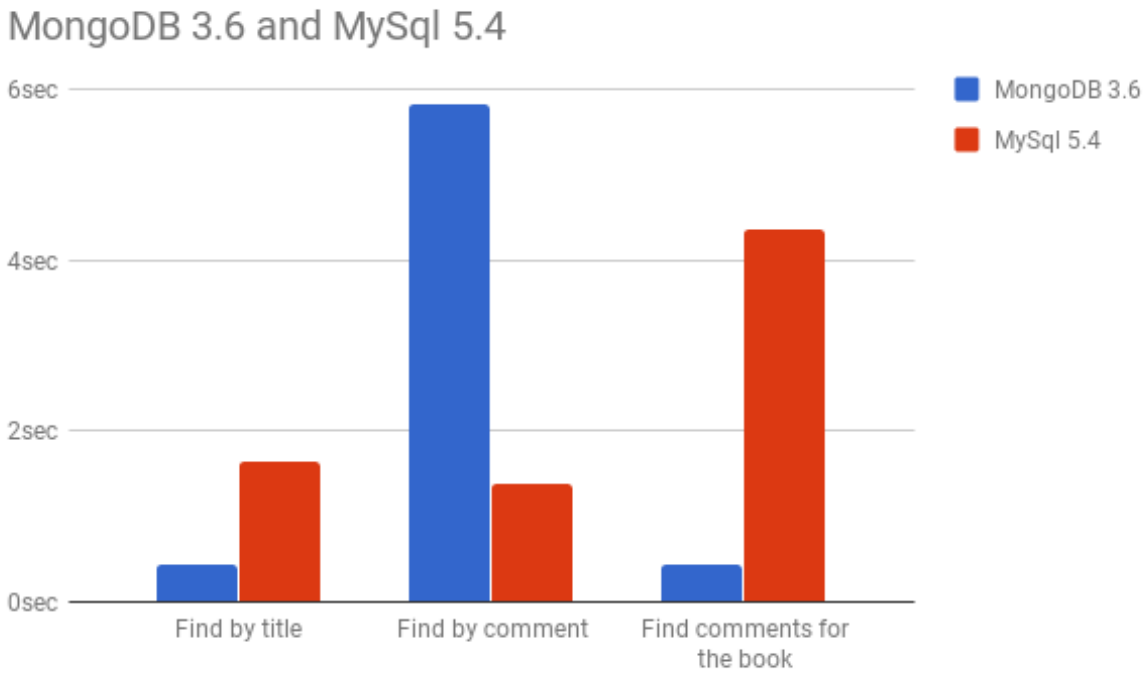

Figure 3. MongoDB and MySQL results comparison.

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] Kiczales G J d R, Bobrow D G 1991 *The Art of the Metaobject Protocol* (MIT Press)
[2] *JavaScript Reflect Global Object* (Access mode: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/GlobalObjects/Reflect) (18.10.2017)
[3] *XPath 3.1 Specification* (Access mode: https://www.w3.org/TR/xpath-31/) (18.10.2017)
[4] Bell C 2012 *Expert MySQL* (APress)
[5] Protsenko V I, Kazanskiy N L and Serafimovich P G 2015 *Computer Optics* **39(4)** 582-591 DOI: 10.18287/0134-2452-2015-39-4-582-591
[6] Loureno J R, Cabral B and Carreiro P 2015 Choosing the right nosql database for the job: a quality attribute evaluation *Journal Of Big Data* **2** 18
[7] Lake P and Crowther P 2013 Nosql databases Concise Guide to Databases *Undergraduate Topics in Computer Science* DOI: 10.1007/97814471560175
[8] Kazanskiy N L, Protsenko V I and Serafimovich P G 2014 *Computer Optics* **38(4)** 804-810
[9] Singh M and Kaur K 2015 Sql2neo: Moving health-care data from relational to graph databases *IEEE International Advance Computing Conference* 7154801

[10] Kazanskiy N L, Protsenko V I and Serafimovich P G 2017 *Procedia Engineering* **201** 817 DOI: 10.1016/j.proeng.2017.09.602
[11] Richardson L and Ruby S 2007 RESTful Web Services (Beijing: O'Reilly)
[12] *MongoDB Official site* (Access mode: https://www.mongodb.org) (18.10.2017)
[13] *Benchmarking NoSQL Top Databases* (Access mode: https://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL Benchmarks EndPoint.pdf) (18.10.2017)
[14] *Yahoo! Cloud System Benchmark* (Access mode: https://github.com/joshwilliams/YCSB) (18.10.2017)
[15] *Amazon EC2 Instance Types* (Access mode: https://aws.amazon.com/ru/ec2/instance-types/) (18.10.2017)