International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 7– No.8, September 2014 – www.ijais.org

Performance Evaluation of NoSQL Systems Using YCSB in a resource Austere Environment

Yusuf Abubakar
Department of Computer Science
Nuhu Bamalli Polytechnic, Zaria - Nigeria

ThankGod S. Adeyi
Department of Mathematics
Ahmadu bello University,Zaria-Nigeria

Ibrahim Gambo Auta
Waziri Umaru Federal Polytechnic
Birnin Kebbi

ABSTRACT
NoSQL is a database used to store high volume of data. NoSQL databases are horizontally scalable, distributed, open source and non-relational. High performance is a major concern for practically every data-driven system. NoSQL databases claim to deliver faster performance than the popular Relational database systems in various use cases, most notably those involving huge data. While this is always the case, it should be understood that not all NoSQL databases are created alike where performance is concerned. This being the case, IT professionals works hard to ensure that the database they select is optimized for the success of their application use cases. Such selection can be made in-house, based on tests with academic database benchmarks. We present the Yahoo! Cloud Serving Benchmark (YCSB) framework, with the goal of facilitating performance comparisons of the new generation of NoSQL databases in an environment where resources are limited. Unlike many previous benchmarks that considered a cluster or distributed system that NoSQL is known for, we limit out experiment to a single PC assuming a cluster with a single node or a distributed system with a single PC. We define a core set of benchmarks and report results for four widely used systems: MongoDB, ElasticSearch, Redis, and OrientDB implementation.

1. INTRODUCTION
While it is always the case that NoSQL stores claim to be faster in terms of performance than RDBMS systems in various use cases especially among the big data stores. It should be noted that not all NoSQL datastores are created alike where performance is concerned. System architects and IT managers are wise to compare NoSQL databases in their own environments using data and user interactions that are representative of their expected production workloads before deciding which NoSQL database to use for new application [11].

Most medium-sized enterprises run their databases on inexpensive off-the-shelf hardware; they need quick answers to complex queries [10]. Thus, it is important that the chosen database system and its tuning be optimal for the specific database size and design. Such choice can be made in-house, based on tests with academic database benchmarks. This paper focuses on measuring the performance of four NoSQL databases on a Cloud system with just a single node to give a direction in a situation where resources are limited and developers have only choice of a single computer system to deploy application whose requirement is best fit into a NoSQL data store.

2. BACKGROUND
In this section, we describe some background about the different data processing systems that we examine in this paper.

2.1 MongoDB
MongoDB [1] is a popular open-source NoSQL database. Some of its properties are a document-oriented storage layer, auto-sharding and asynchronous replication of data between servers and indexing in the form of B-trees. In MongoDB data is stored in collections and each collection is made up of documents. Collections and documents are loosely analogous to tables and records, respectively, found in relational databases. MongoDB does not require a rigid database schema for the documents. Specifically, documents in the same collection can made up of different structures. Another important feature of MongoDB is its support for autosharding. With sharding, data is partitioned between multiple nodes in an order-preserving manner. Sharding is similar to the horizontal partitioning technique that is used in parallel database systems. This feature enables horizontal scaling across multiple nodes. When some nodes have a disproportionate amount of data compared to the other nodes in the cluster, MongoDB redistributes the data automatically so that the load is equally distributed across the nodes/shards.

2.2 ElasticSearch
Elasticsearch is a horizontally-scalable, open source distributed database built on Apache's Lucene that delivers a full-featured search experience across terabytes of data with a simple yet powerful API. It is built to handle huge amounts of data volume with very high availability and to distribute itself across many machines to be fault-tolerant and scalable, all the while maintaining a simple but powerful API that allows applications from any language or framework access to the database [12].

Mapping is similar to a schema definition in SQL databases. A mapping is a crucial part of every index in Elasticsearch: it defines all document types within the index and how each document and its fields are saved, analyzed, and indexed. Elasticsearch can work with either implicit or explicit mapping [12].. If the Elasticsearch server has not been handed mapping before a document is inserted, the server will try to infer the type of the document based on the values in the fields of the document and add this type to the mapping. While implicit mapping might be an adequate solution in some cases, the use of explicit mapping provides an opportunity to create complex document types and to control how the Elasticsearch server analyzes each field. Explicit mapping allows the disabling of indexing of some fields in a document (by default the Elasticsearch server indexes all fields), which reduces the amount of the disk space needed

and increases the speed of adding new documents. This also provides a way to store the data that must not be searched but must be quickly accessible through indexed fields. For example, if we have a set of commits in a version control repository, we might want to index fields like author, date, commit message, etc., but remove the actual change sets from the index. While change sets remain to be instantly accessible through other fields, they neither take additional disk space nor increase the time required to index a document.

Elasticsearch provides its own query language based on JSON called Query DSL. A given search can be performed in Elasticsearch in two ways: in a form of a query or in a form of a filter. The main difference between them is that a query calculates and assigns each returned document with the relevance score, while a filter does not. For this reason, searching via filters is faster than via queries. The official documentation recommends using queries only in two situations: for full text searches or when the relevance of each result in the search is important. For simplicity, we will use term query to describe both queries and filters; however, our experience with Elasticsearch is limited to working only with filters, thus we do not report about use of queries.

The search in ElasticSearch is near real-time [12]. It means that although documents are indexed immediately after they are successfully added to an index, they will not appear in the search results until the index is refreshed. The Elasticsearch server does not refresh indices after each update, instead it uses a specified fixed time interval (the default value is 1 second) to perform this operation. Since refreshing is costly in terms of disk I/O, it might affect the speed of adding new documents [12]. Therefore, if you need to perform a large number of updates at once, you might want to temporally increase the default indexing interval value (or even disable auto-refresh) and then manually refresh indices after updates are completed.

ElasticSearch is a Restful server, so the main way of communication with it is through its REST API. Communication between the Elasticsearch server and a client is straight forward. In the majority of cases, a client opens a connection and submits a request, which is a JSON object, and receives a response, which is also a JSON object. The simplicity of this mode of communication places no restrictions on programming language used to implement clients or the platforms that they operate on; if a client can send HTTP requests, it can communicate with the Elasticsearch server. Moreover, there are libraries for different languages (e.g.,PyES for Python) that take care of some mechanics, and can provide better integration with the language.

2.3 OrientDB

OrientDB is an open source NoSQL database management system written in Java. It is a document-based database, but the relationships are managed as in graph databases with direct connections between records. It supports schema-less, schema-full and schema-mixed modes. It has a strong security profiling system based on users and roles and supports SQL as a query language. OrientDB uses a new indexing algorithm called MVRB-Tree, derived from the Red-Black Tree and from the B+Tree; this reportedly has benefits of having both fast insertions and fast lookups [13].

Features

a) Transactional: supports ACID Transactions. On crash it recovers pending documents.

b) GraphDB: native management of graphs. 100% compliant with TinkerPop Blueprints standard for Graph database.

c) SQL: supports SQL language with extensions to handle relationships without SQL join, manage trees and graphs of connected documents

d) Web ready: supports natively HTTP, RESTful protocol and JSON without use 3rd party libraries and components.

e) Run everywhere: the engine is 100% pure Java: runs on Linux, Windows and any system that supports Java technology.

f) Embeddable: local mode to use the database bypassing the Server. Perfect for scenarios where the database is embedded [13].

2.4 Redis

Redis [14] is an open source in-memory key-value store database that promises very fast performance, and more flexibility than the basic key-value structure. In Redis, a database is well-known by a number; the normal database is number 0. The number of databases can be configured but default is 16 databases [14]. Basically, a Redis database is a dictionary of key and value pairs. Nevertheless, apart from the classic key-value structure where value is a string and users are responsible to parse it at the application level, Redis offers more choices of data structures, where a value can be stored as: A string. A list of strings: Insertions at either the head or tail of the list are supported. Besides, querying for items near the two ends of the list is extremely fast, while querying for one in the middle of a long list is slower. A set of strings: This is a non-duplicated collection of strings which means adding the same string repeatedly yields only one single copy. Add and remove operations only take constant time (O (1)).

A sorted set of strings: Similar to set but in a sorted set, each string is associated with a score specified by clients. This score is used as the criteria for sorting and can be the same among multiple members of the set.

Redis databases can be replicated using the master-slave model. However, it does not support automatic failover, which means if the master crashes, a slave has to be manually promoted to replace it. A slave can have other slaves of its own, so it can also accept write requests, though a slave is in read-only mode by default. At the time being, sharding is not officially supported, although it is provided by some particular drivers [14].

3. RELATED WORK

Database benchmarking is an often discussed topic in the research area of relational databases. In this section we give an overview of the current research state of topics related to our intended goal to test performance of some NoSQL and SQL databases on a single machine.

[8] Proposed a workbench tool to efficiently run numerous benchmark tests to achieve high self-reliance results. Their tool interfaces with a workload generator, like the YCSB Client, to execute each run. They provided a good background for our work as they provide us an alternative benchmark tool to consider.

[9] argue that many micro and macro benchmarks do not model real workloads effectively. One approach they propose is to measure the performance of system operations, and compute the expected performance for a given application that uses some specified combination of those operations.

[10] Presents Transaction Processing Performance Council (TPC) database benchmark that measures the performance of ad-hoc Decision Support (DSS) queries. They present the benchmark and the steps that a non-expert must take to run the tests and report their own benchmark tests, comparing an open-source and a commercial database server running on off-the-shelf hardware when varying parameters that affect the performance of DSS queries. Their work is limited to a TPC databases only.

[5] Presents the Yahoo Cloud Serving Benchmark (YCSB) framework to facilitate performance comparisons of the new generation of cloud data serving systems. They define a core set of benchmarks and report results for four widely used systems: Cassandra, HBase, Yahoo!'s PNUTS, and asimple sharded MySQL implementation. They work do not address a situation where resources are limited and do not consider some popular NoSQL databases like Mongodb, Redis, e.tc in their experiment.

4. DETAILS OF THE BENCHMARK TOOL

We used an existing tool provided by Yahoo, called the YCSB Client, to execute these benchmarks. A key design goal of this tool is extensibility as it can be used to benchmark new cloud database systems. We have used this tool to measure the performance of four NoSQL systems, as we report in the next section. This tool is available under an open source license. It has ready adapters for different NoSQL Databases. YCSB tool allows benchmarking multiple systems and comparing them by creating "workloads". Using this tool, one can install multiple systems on the same hardware configuration, and run the same workloads against each system. The architecture of YCSB is as shown in figure 1.

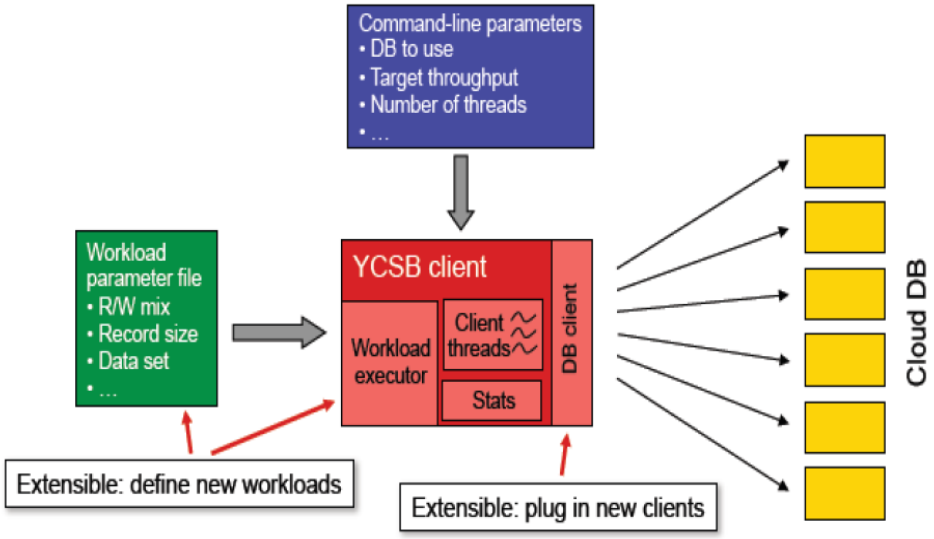

Figure 1: The YCSB Architecture [4]

4.1 Workloads

In this section we describe the set of workloads used for the experiment in this paper.

The YCSB framework contains a core set of workloads to evaluate different aspects of a system's performance, called the YCSB Core Package. In YCSB, a package is a collection of related workloads. The workload defines the data that will be loaded into the database during the loading phase, and the operations that will be executed against the data set during the transaction phase, and can be used to evaluate systems at one particular point in the performance space. A package, which includes multiple workloads, examines a broader slice of the performance space. While the core package examines several interesting performance axes, YCSB have not attempted to exhaustively examine the entire performance space. It is developed in such a way that users of can develop their own packages either by defining a new set of workload parameters, or if necessary by writing Java code. The following Workloads were considered in this report;

4.1.1 Workload A – 100% Insert

This workload writes 1000, 20000, 40000, 80000 and 100000 thousand one kilobyte (1KB) record into an empty database in each case recording the performance measurements in a text file.

4.1.2 Workload B – 100% Read

This is read intensive workload, i.e. it retrieves 1000, 20000, 40000, 80000 and 100000 thousand one kilobyte (1KB) record from a populated database in each case recording the performance measurements in a text file.

4.1.3 *Workload C – 100% Update*

Workload C changes a single field in a record size of 1000, 20000, 40000, 80000 and 100000 thousand one kilobyte (1KB) from a populated database in each case recording the performance measurements in a text file.

4.2 Overview of the Test

Each workload was validated with 10 client threads combined with overall records of 1000, 20000, 40000, 60000, 80000 and 100000; And each of these combinations repeated 20 times. YCSB by default creates 1k size records. The Execution time vs number of records using 10 client threads was measured. This utilizes all the cores of the test system and describes how increasing the number of records affects the average response time of a database operation.

5. RESULTS

In this section, we present benchmarking an experimental evaluation of the four NoSQL Systems using the various workload described in section 4.

Execution Time (INSERT Operation)

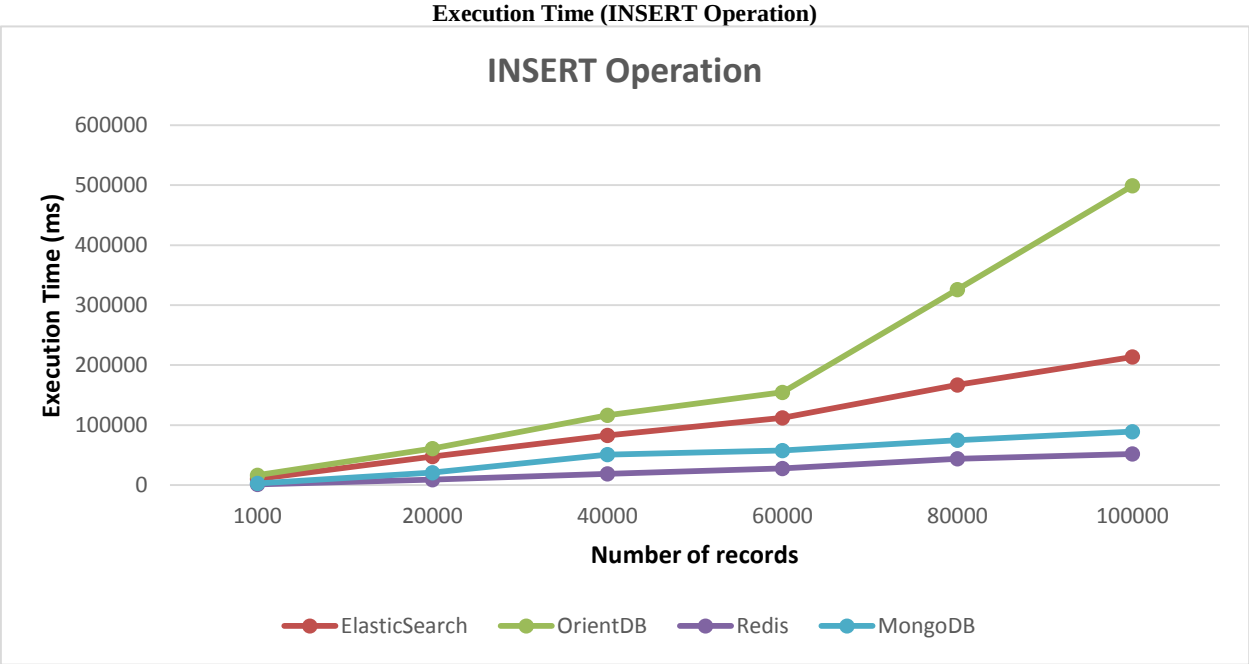

Figure 2: Graph of Execution time Against Number of records for Insert Operation.

Redis has the best performance, followed by MongoDB, ElasticSearch and OrientDB respectively based on the time they take to perform insertion. Redis is obviously optimized for writes and can perform them faster than reads even when the database is not heavily contended. Operations in Redis are fast enough because of its in-memory nature [6].

Execution Time (Update Operation)

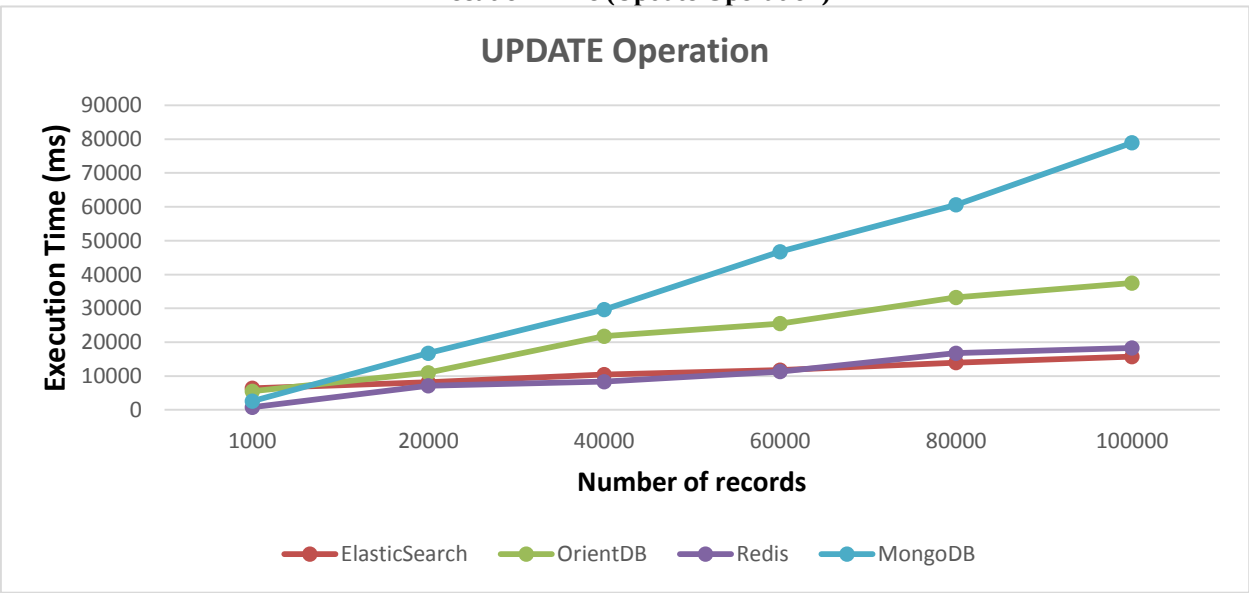

Figure 3: Graph of Execution time Against Number of records for Update Operation.

The above graph shows the speed of updating 1000, 20000, 40000, 60000, 80000 and 100000 records from the tested databases.

MongoDB is poorer followed by OrientDB for Update operation. Redis has the best performances especially at lower workload, but compete with ElasticSearch at higher workload (60000 records and above)

Execution Time (READ Operation)

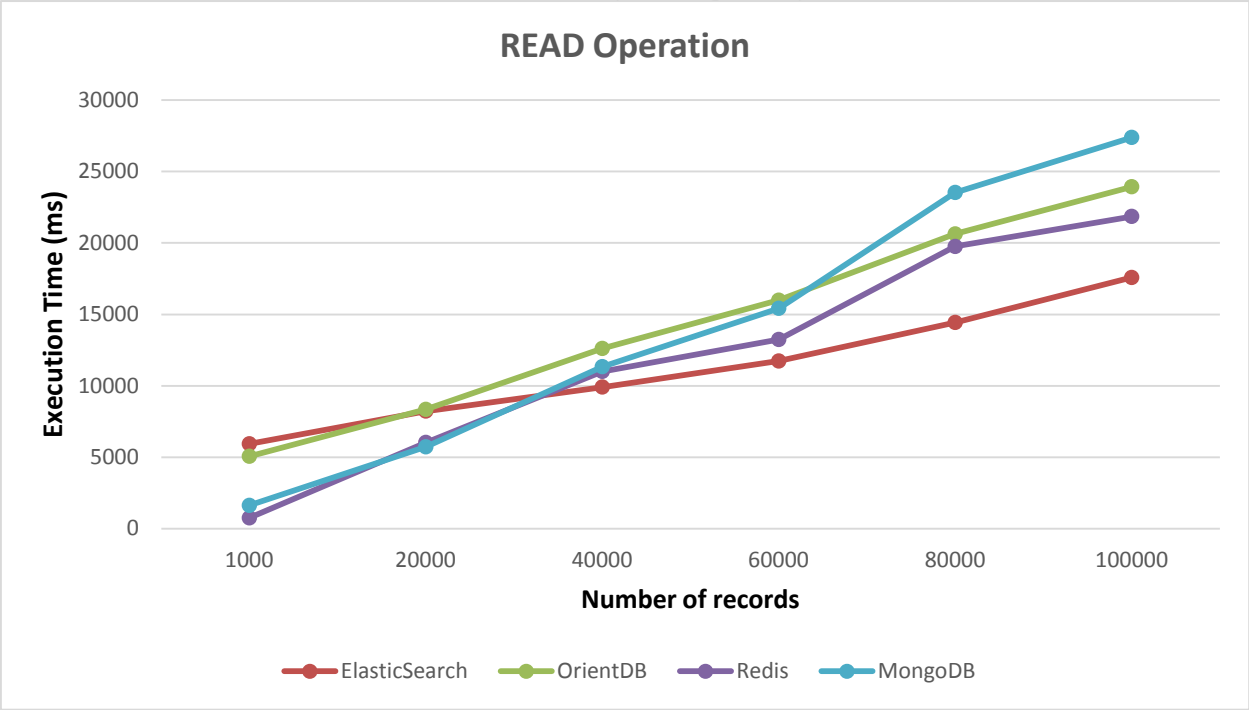

Figure 4: Graph of Execution time Against Number of records for Read Operation

The above graph shows the performance of the tested databases while reading 1000, 20000, 40000, 60000, 80000 and 100000 records.

The graph shows that the four database system tested have no consistent graphic pattern while performing the read operation.

6. CONCLUSION

IT professionals need to do their best to ensure that the database they select is appropriate and targeted for their application use cases as fast performance is important for nearly every data-driven system.. One of the ways to do this is to conduct a Benchmark test in the environment in which the database will run and under the expected data and concurrent user workloads. Benchmarks such as those contained in this paper can be useful as well in that they give database users a good idea of what the core strengths and weaknesses of the database they intend to use possesses.

7. REFERENCES

[1] MongoDB. http://www.mongodb.org/

[2] MongoDB – Replica Sets. http://www.mongodb.org/display/DOCS/Replica+Sets

[3] Sematext. Elasticsearch refresh interval vs indexing performance. http://bit.ly/1iZoPGc, July 2013.

[4] B. F. Cooper et al. PNUTS: Yahoo!'s hosted data serving platform. In VLDB, 2008.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[6] N. Hurst. (2010, March) Visual guide to NoSQL systems. [Online]. Available: http://blog.nahurst.com/visual-guide-to-nosql-systems.

[7] B. White et al. An integrated experimental environment for distributed systems and networks. in OSDI, 2002.

[8] P. Shivam et al. Cutting corners: Workbench automation for server benchmarking. In Proc. USENIX Annual Technical Conference, 2008.

[9] M. Seltzer, D. Krinsky, K. A. Smith, and X. Zhang. The case for application-specific benchmarking. In Proc. HotOS, 1999.

[10] A. Thanopoulou, P. Carreira, and H Galhards. Benchmarking with TPC-H on off-the-shelf Hardware. AN Experiments Report.

[11] Datastax. Benchmarking Top NoSQL Databases:A Performance Comparison for Architects and IT Managers. White Paper BY DATASTAX CORPORATION, FEBRUARY 2013

[12] Oleksii K., Olga B, Reid H., and Michael W.G. Mining Modern Repositories with Elasticsearch. Cheriton School of Computer Science University of Waterloo, Waterloo, ON, Canada

[13] OrientDB. http://en.wikipedia.org/wiki/OrientDB.

[14] Karl Seguin. The little redis book. Karl Seguin, 2010.