Baltic J. Modern Computing, Vol. 5 (2017), No. 2, 147-163
http://dx.doi.org/10.22364/bjmc.2017.5.2.01

Use of NoSQL Technology for Secure and Fast Search of Enterprise Data

Juris RĀTS

RIX Technologies
Blaumaņa 5a-3, Rīga, LV-1011, Latvia

juris.rats@rixtech.lv

Abstract. NoSQL solutions are used for search in large data volumes of unstructured data, mostly for the search of the public data (e.g. Google search). There are important differences between Google search and Enterprise search though. The latter should expose information strongly according to user's access rights. We create a persistence model that supports high performance search in large volumes of unstructured data in accordance with user access rights. The model is based on two level data store. The primary store is an SQL database and is used for the input and maintenance of the current data. The secondary store is an Elasticsearch database and is used for the search and retrieval of data. We describe the proposed model and provide the results of the performance tests. The 2000 ECM user strong workload on 1 million document database features 0.14 second average response time on a one node database configuration on a commodity server.

Keywords: NoSQL, Enterprise Content Management (ECM), Enterprise search, polyglot persistence, clustered processing

1. Introduction

Technology advance in a number of areas (computer networking, availability of cheap storage and processing power, ubiquity of data gathering devices etc.) allows to accumulate and save vast amounts of data that could possibly be used to everybody's good. Every day 2.5 quintillion bytes of new data are created, 90% of the data in the world today has been created in the last two years alone (Frank, 2012). This creates fundamental changes in methods of data processing and even shatters some seemingly unquestionable truths. Like: "This is a fundamental principle of every database – all the data must always be in a consistent state" (Strazdins, 2016). We are moving to the distributed, clustered solutions, where the ACID (i.e. Atomicity, Consistency, Isolation, Durability) principle costs too much. CAP theorem (Simon, 2012) has proved a distributed solution cannot support all of the three important features (Consistency, Availability and network Partitioning) and data consistency requirements are the ones usually compromised in favour of availability and tolerance for network partitioning. The BASE (Basically Available, Soft state, Eventual consistency) principle used for distributed systems talks about eventual consistency that allows for data replicas on cluster nodes to be out of sync for a short periods of time.

Another important shift is understanding there is no "one best choice" for all cases (Vorhies, 2015). A myriad of NoSQL solutions have been created lately that can be used to handle various types of business processes, like user session management (key-value stores), shopping carts (document or key-value databases), analytics (column databases), recommendations (graph or column databases) or social media analysis (key-value or document databases) (Vorhies, 2015). It is not uncommon to use several data persistency platforms (NoSQL and SQL) inside one solution, the new term Polyglot Persistence has been coined to designate the approach (Vorhies, 2015).

The main subject of our research is Enterprise search. NoSQL solutions are used for search in large data volumes of unstructured data. The example is a Google search. There are important differences between Google search and Enterprise search though (Oleson, 2015). The most important for us is that Google search (as well as search methods supported by other NoSQL platforms) is about anonymity while Enterprise search is about security. Enterprise search should expose information strongly according to user's access rights. We aim to create a model that supports high performance search in large volumes of unstructured data in accordance with user access rights.

2. Related work

Shermin (Shermin, 2013) develops an extended RBAC model for NoSQL databases. It is aimed to provide fine grained mechanism controlling access of users to data objects. The model is flexible and powerful and allows for dynamic allocation of access rights. Unfortunately the generic and dynamic nature of the model is an overhead making implementation of fast information search a tough issue. The model does not provide means to determine the user access rights to data objects at index time. This would be necessary to create (and possibly load into cache) fast indexes.

A simple RBAC user access control is implemented in NoSQL database Cassandra (Meng, 2016). Permissions on database resources are granted to roles, still a role here is a synonym of user or user group. No information is available how the user access model can be used for information search. No performance evaluation is given.

Adam Fowler (Fowler, 2015) shows how to implement a user access control in NoSQL database that supports ACID transactions and pre-commit triggers. Fowler arguments that the access rights must be assigned to the data object inside the transaction that creates (or modifies) the data object, hence the requirement for the ACID support. Unfortunately only a handful of NoSQL databases supports ACID which makes the proposed method less valuable.

3. Enterprise Content Management

Enterprise content management (ECM) “comprises the strategies, processes, methods, systems, and technologies that are necessary for capturing, creating, managing, using, publishing, storing, preserving, and disposing content within and between organizations” (Grahlmann et al., 2012). ECM covers a wide area of functionality (Nilsen, 2012; Korb and Strodl, 2010; Blair, 2004), including (Kampffmeyer, 2006):

- Document Management;
- Collaboration of supporting systems;
- Web Content Management;
- Records Management;
- Workflow and Business Process Management.

Nowadays there are a number of challenges ECM systems must face (Agrawal et al., 2012), the most important of them is the scaling problem - data volume is scaling faster than computer resources. Organisations handle larger amounts of documents, moving to cloud-enabled solutions creates large multi-tenant databases, organisations absorb social media data and streams of device generated data. The Volume, Velocity and Variety of enterprise data expands exponentially. Scaling up (i.e. replacing a smaller machine with a bigger one) is an expensive solution as 10-terabyte system could cost 100 times more than 1-terabyte system (Mcknight, 2014).

Shirky pointed out back in 2008 - it's not information overload, it's filter failure (Shirky, 2008). We are swiftly growing out of the old data persistence tools - the relational database systems have inherent problems of scalability (Wiggins, 2009). This have caused a creation of wide range of NoSQL (Not only SQL) technologies aimed to solve the named challenges.

4. NoSQL

The term NoSQL (Not only SQL) was initially used by Carlo Strozzi (Fowler, 2015) in 1998. The development of the Google's Bigtable structured distributed database (one of the first successful NoSQL technologies) started in 2004. More than 225 NoSQL platforms of various kinds are developed so far (Edlich, n.d.). A number of slightly different NoSQL taxonomies exist (Edlich, n.d.; Fowler, 2015; Solid IT, n.d.; Mcknight,2014)). We will use the following:

- Key-value stores (e.g., Redis and Memcached);
- Column family stores (e.g., Cassandra and HBase);
- Document stores (e.g., MongoDB, CouchDB and Elasticsearch);
- Graph stores (e.g., Neo4j and Titan);
- Hybrid (multi-model) stores (e.g., ArangoDB and OrientDB).

Elasticsearch is usually marketed as a search engine, still it is full blown document store and hence we put it in this category.

NoSQL Document databases have important advantages to offer for the persistence layer of ECM systems because they are schema-less, easily replicable and scalable (Potts, 2010). Schema-less means that NoSQL Document databases are convenient for the semi-structured data of organizations. NoSQL Document databases are easily replicable because of their share-nothing architecture (new server nodes can be instantiated easily and data replicated between them). They are extremely scalable as data can be easily sharded (split) and distributed on multiple nodes.

When considering use of NoSQL platform for Enterprise Content Management one should consider though adding a security layer because NoSQL platforms usually do not provide Enterprise level security (Winder, 2012; Hannan, 2015, Shermin, 2013).

Document store is the most convenient NoSQL technology for ECM (Potts, 2010; Rats and Ernestsons, 2013) still the Polyglot Persistence approach would be the best fit here to use ACID support of relational database for data maintenance and NoSQL document store for fast information search and retrieval.

5. Elasticsearch

We selected Elasticsearch data store for our search architecture. The main reasons are its extreme scalability as well as powerful search features. Elasticsearch database can be installed on a laptop and you are ready to start with no configuration. At the other extreme Elasticsearch can be installed on a multi-node cluster and configured to handle huge databases. Synthesio runs Elasticsearch on a 75 node cluster in two data centres to enable fast retrieval of up to 50 million of documents out of tens of billion (WEB (d)).

There are NoSQL document solutions that provide more functionality and flexibility than Elasticsearch (e.g., MongoDB) still features of Elasticsearch are a best fit for our domain and architecture in mind. Elasticsearch beats MongoDB and other platforms with similar functionality in respect to search performance. The evaluation results (Abramova et al., 2014) show that on a request flows consisting of 50% writes and 50% reads Elasticsearch performs about 4-5 times faster than MongoDB. Search against the nested objects is performed 20 times faster by Elasticsearch and aggregation – 40 times faster (Marechal, 2015). Moreover the performance of MongoDB degrades faster with the increase of data volume (Gupta, 2015).

6. Architecture for Enterprise search

One cannot tell by looking at a description of a document database model whether or not it will perform efficiently. One must consider how users will query the database, how much inserting will be done, and how often and in what ways documents will be updated (Sullivan, 2015). Hence we consider data usage patterns when making decisions about the data model and the architecture.

We propose to base the persistence architecture on two data stores. The primary store is an SQL database and is used for the input and maintenance of the current data. The secondary store is an Elasticsearch database and is used for the search and retrieval of data. The new data objects go first into primary store and then are replicated to the secondary store. The advantages of this approach are as follows:

- the load on primary store is reduced as search and retrieval requests are handled by secondary store;
- data search and retrieval requests are not forced to wait for write operations to release index locks;
- Elasticsearch inverted indexes are used for fast search (including full-text);
- Elasticsearch sharding and replication is used to improve performance with data and request volume growth;
- Elasticsearch replication is used to support high data availability.

More details on the mentioned advantages follow in the sections below.

6.1. No locking

Write transaction of the Relational database involves a number of data objects (e.g., tables and indexes). To ensure data consistency the objects involved have to be locked (made unavailable) for other processes while transaction is in progress. This means that other requests (write and read) have to wait while transaction releases the locks. This leads to fast performance degradation when request load grows.

Our model profits from no-locking write provided by Elasticsearch. Elasticsearch uses Lucene indexes that are immutable thus there is no need to lock index when writing data (Brasetvik, 2013). New index segments are created to index new data instead while index segments are merged in background later on. Thus Elasticsearch secondary database is available for search and data retrieval no matter how intense is the flow of new data replicated from the primary store.

6.2. Scalability

Elasticsearch database consists of number of shards. When the data store grows new nodes can be added to cluster. Elasticsearch automatically relocates shards when new nodes added. Thus database on 5 shards (with no replicas) can run on 1 to 5 node cluster.

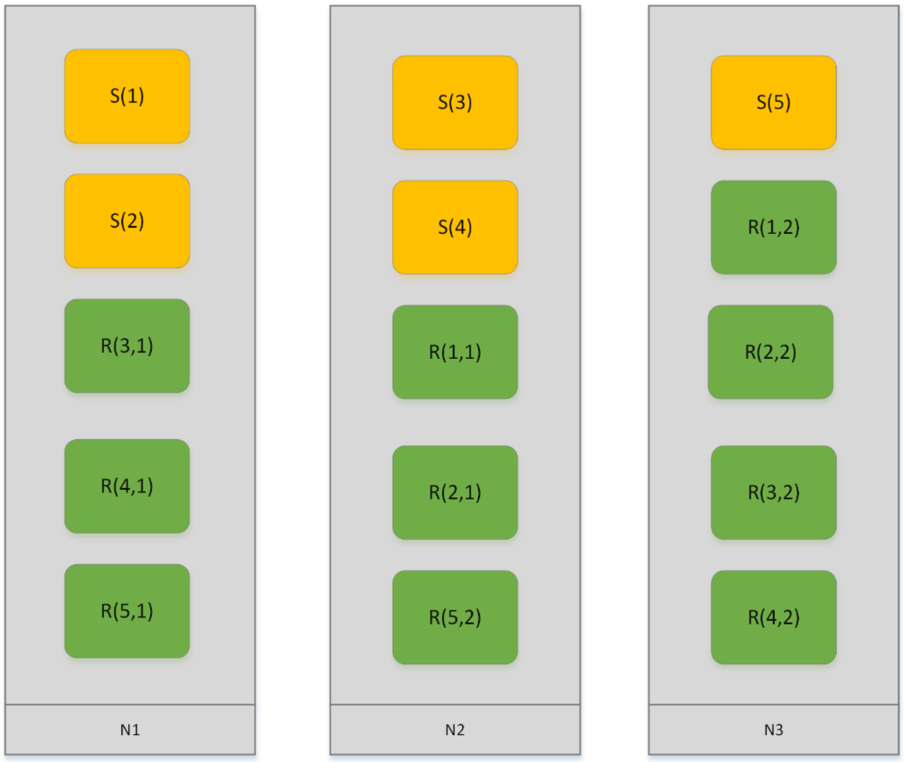

Figure 1. Sample Elasticsearch cluster.

6.3. Availability

Elasticsearch database can be configured to support replicas (copies) of the shards. The number of replicas Elasticsearch must maintain is called replication factor. If replication factor is set to two, for example, Elasticsearch creates and maintains 2 replicas of every shard. Elasticsearch distributes replicas of a shard to different nodes of the cluster (thus one needs a cluster of at least 3 nodes to use replication factor 2).

Replicas allow to scale Elasticsearch database to arbitrary number of nodes. The search requests are distributed between replicas while new data is written to the main (master) copy of the shard and then copied to the replicas. A sample Elasticsearch cluster of 5 shards and clustering factor 2 is shown on Figure 1. Here S(i) is a master copy of shard i and R(i,j) is replica j of the shard i, while N1, N2 and N3 – cluster nodes.

Shards and replicas allow to speed up data requests. Moreover – because of the replicas the cluster is functional even if a part of the cluster nodes are not available. The sample cluster above is fully functional if any of the three cluster nodes become unavailable (although every of the three nodes contain full data copy it is not safe to allow users in when connection between all three nodes have been lost – this may lead to so called split brain when users interact with three independent systems).

7. User access restriction model

Unlike a relational database permissions in NoSQL platform cannot be set on a schema level. Moreover, we cannot profit from use of transactions as long as Elasticsearch do not support them. Our solution is to write data object (document) in a data store along with the all user access related metadata in a single write operation. This means the additional data must be included in the write request by the application layer. The user access restriction model consists of the following:

- User is assigned a set of *user access attributes* (calculated out of his roles etc.);
- Data objects include *data access attributes*;
- Criteria of the search request are augmented with application layer created criteria that match user access attributes with data access attributes.

Figure **2** below shows the proposed user access restriction model in detail.

The model is ECM related but might be tuned for a particular case e.g. by adding more attributes. Sample user access restriction description (in json notation) is below.

```
{
"bool":{
"should":[
{"terms":{"canRead":["U1364"]}},
{"terms":{"case":["L11933","L13407","L14997",…]}
]
       }
}
```

The description tells that the data object is accessible to:

- User ID U1346 or
- Users having access to one of the cases L11933, L13407, L14997 etc.

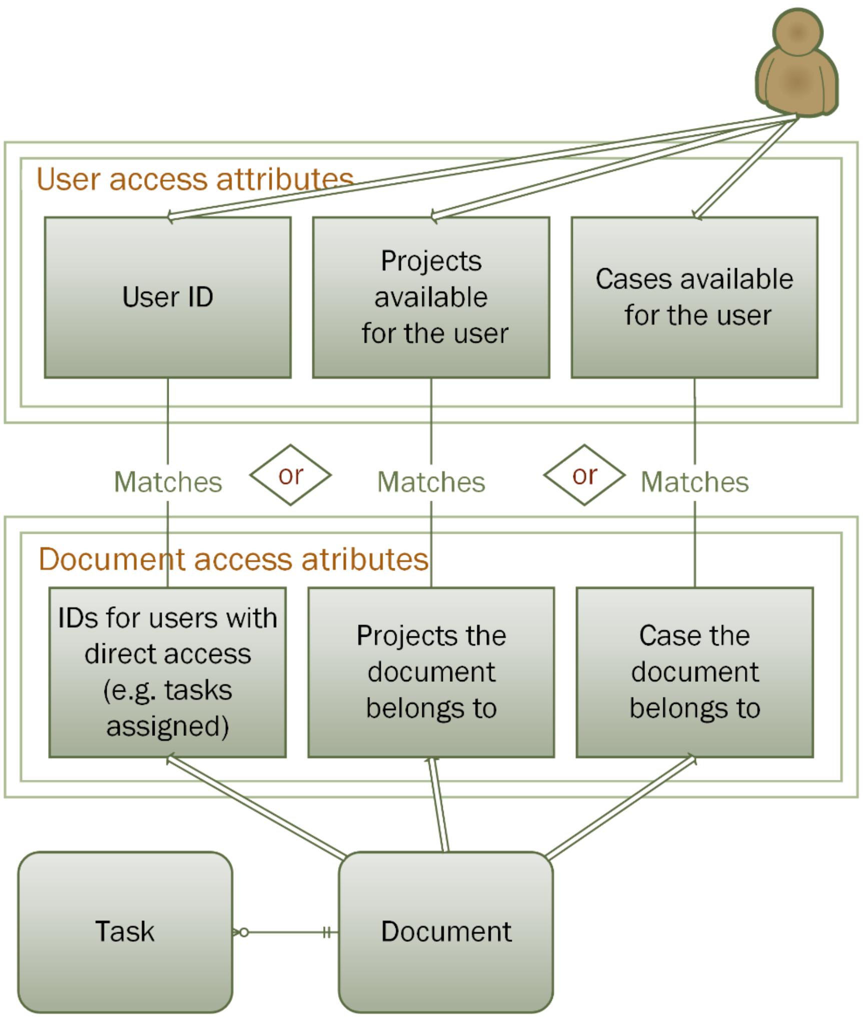
User access attributes
User ID
Projects available for the user
Cases available for the user
Matches
or
Matches
or
Matches
Document access atributes
IDs for users with direct access (e.g. tasks assigned)
Projects the document belongs to
Case the document belongs to
Task
Document

Figure 2. User access restriction model.

8. ECM data model

ECM systems add metadata to documents to facilitate enterprise business processes. The metadata fields vary but the data model should mainly include:

- Full text of the document
- Document title
- Document number

- Person in charge
- Creation time
- Deadline
- Groups the document belongs to (e.g. case, folder, project)
- Status
- Array of task descriptions consisting of author of the task, person in charge, task type, task status and comments.

All the information for the data access attributes described above can be found or can be derived from the named metadata fields. Still for performance reasons we add attribute containing the list of user IDs having access to the document. This is a unique list of user IDs that includes user ID of the person in charge of the document plus user IDs of the authors of the tasks plus user IDs of the persons in charge of the tasks of the document (a user ID is included in this list only once no matter how many times the ID appears in role of task author and a person in charge of the task or document).

The described data object contains parent child relation between the document and its tasks. Relational databases would use two tables and a parent-child relationship. NoSQL databases prefer aggregated models – this should mean the representation of tasks as a nested object and storing with the document as one aggregated item. Still this is not necessarily the best solution. Another option is to create two separate data types – one for documents and one for tasks and to store in a task a reference to a parent document. Elasticsearch allows to configure one data object type as a child of other one to cover this case.

The two alternatives (nested and parent-child) each has advantages and disadvantages. The advantages of the parent-child model are (WEB (a)):

- The parent object can be updated without re-indexing the children;
- Child objects can be added, changed, or deleted without affecting either the parent or other children;
- Child documents can be returned as the results of a search request.

The main advantage of the nested model is that it is generally about 5-10 times faster (WEB (b)).

It is rather important for ECM systems to search on tasks (e.g. to produce a list of tasks in progress assigned to a given user). Therefore the parent-child model looks a better candidate for our case. Still we have another solution here. The task data normally is requested for the current data, therefore we could consider a modification of our model:

- Remove older data from the primary store and keep full data only on secondary store;
- Execute advanced task searches against the primary store.

We could use the nested model for the secondary store in this case. We will leave this option though for the future research and stick with the parent-child model. Figure 3 shows the model in detail.

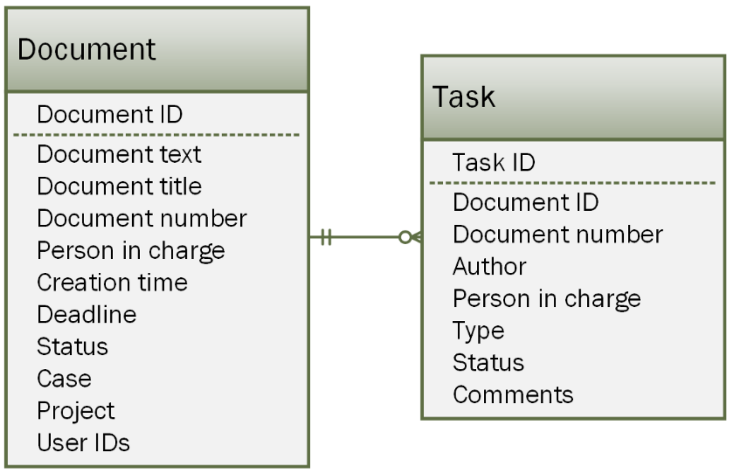

Figure 3. Data model for the secondary store.

It is rather important for ECM systems to search on tasks (e.g. to produce a list of tasks in progress assigned to a given user). Therefore the parent-child model looks a better candidate for our case. Still we have another solution here. The task data normally is requested for the current data, therefore we could consider a modification of our model:

- Remove older data from the primary store and keep full data only on secondary store;
- Execute advanced task searches against the primary store.

We could use the nested model for the secondary store in this case. We will leave this option though for the future research and stick with the parent-child model. Figure 3 shows the model in detail.

The heading attributes (*Document ID* and *Task ID*) are unique keys of the respective objects here. The attribute *Document ID* of the Task object is a reference to the parent Document object. We use denormalisation here and include the document number in the task object as well. This promises to be a performance improvement as document number is frequently used in task related searches. The overhead of maintaining the duplicated data is of little importance as document numbers are rarely changed. The User IDs attribute contains the IDs of the users with direct access to the document (see above in this chapter).

9. Execution of search

Three types of actions at three different points in time have to be carried out to enable search with the proposed user access restriction and data models:

- Data objects have to be indexed with data access attributes at *index time*;
- User access attributes have to be calculated (e.g., out of the roles and permissions assigned to the user) at *user login time*;

- Search criteria have to be supplemented with the access rights related clauses at *request execution time*.

The search performance generally depends on all three types of actions as described below.

9.1. Indexing of data objects

The document object has to be indexed every time when:

- A document is created or modified;
- A task is created that has author or person in charge not yet listed in User IDs attribute;
- A task is deleted or it is modified so that the person in charge is changed (we would have to read all tasks of the document to know if the User IDs list has been really changed in this case, therefore it is better just to index document).

The task object has to be indexed every time when:

- The task is created or modified;
- The document number of the parent document has been changed.

The indexing does not lock any Elasticsearch indexes therefore this should not have important impact on the search performance. This still should take some processing resources therefore we include document and task maintenance requests in our performance tests.

9.2. Processing of user access attributes

User permissions normally are calculated out of their roles, user groups etc. Still our access restriction model is transparent in relation to access control model used. It is just necessary to assemble user access attribute values before the user executes data requests. As a rule user access rights are not changed during the user session (users may be forcefully logged out in emergency cases) hence the most convenient time to calculate user access attributes is at user login time. The analysis of ECM production database data we carried out shows that the data volume of the user access attributes usually fits into 1KB. This allows to keep the user access attributes with the user session data.

9.3. Searching

With the preparatory actions taken during the index and user login time we have all necessary data ready for fast and secure search:

- Data access attributes are stored with the data objects and indexed; data access attribute values are not changed frequently hence the search results can be stored in cache which makes the search even faster;
- User access attributes are stored with the user session data.

When a user triggers the search request the search mechanism wraps the access control related clauses (we call them *user access filter*) around the search criteria and runs the request on the data store. This makes data store to return only the data objects that:

- Are accessible to the user (i.e., match the user access filter);
- Are relevant to the search criteria.

Below is the sample search request of phrase "Italian food" for the user U1815 that has access to cases L18327, L20271, L22271 etc. The results are ordered by date and first 20 hits are returned.

```
{
        "query":{
                "filtered":{
                        "filter":{
                                "bool":{
                                        "should":[

        {"terms":{"canRead":["U1815"]}},
                                                                {"terms":{"case":[
                                                                        "L18327",
                                                                        "L20271",
                                                                        "L22271",
                                                                        …]}}
                                        ]
                                }
                        },
                        "query":{

        "match_phrase":{"files.content":"Italian food"}
                        }
                }
        },
        "sort":[{"date":{"order":"asc"}}],
        "size":20
}
```

The filter part of the request selects data objects that are either available to user because she is either person in charge of the document or some task of the document, or she is an author of some of documents tasks. The match_phrase then search for the documents matching the phrase "Italian food" out of the documents selected by the filter.

This mechanism works very efficiently on large document volumes and heavy request flows because:

- The user access filter is very fast and allows to reduce importantly the data amount for the second part of the search (see explanation and
- Figure 4 above);
- User access rights are not changed frequently thus the results of the filtering can be stored in a cache for later reuse.

The analysis of the data in ECM production databases of total amount of more than 1 million documents carried out in scope of this research shows that 60% of the ECM users have access to less than 5% of the documents (Figure 4).

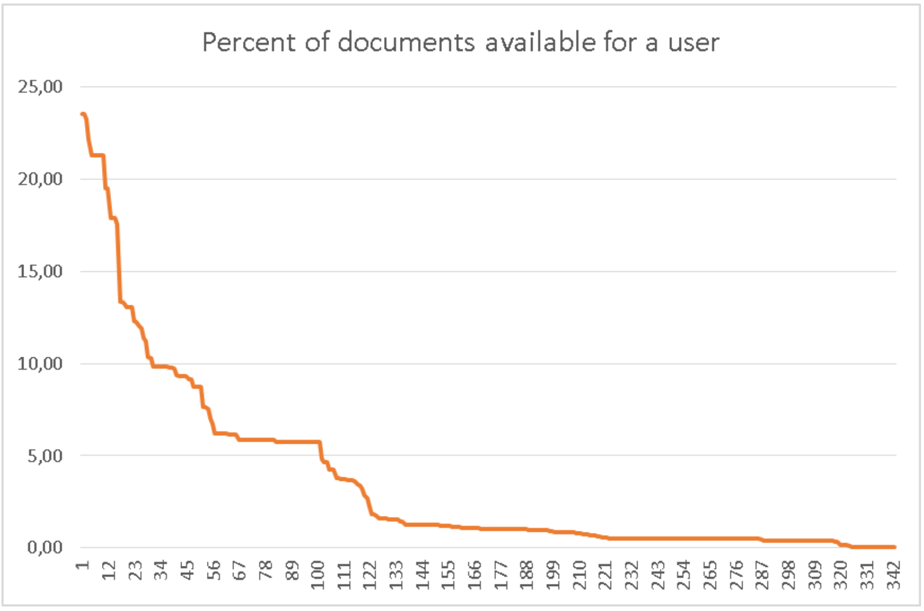

Figure 4. Percent of documents available for a user in a 1 million document database.

10. Performance

Some NoSQL database performance evaluation results are available (Abramova et al., 2014)(WEB (c)). The evaluation is done on a number of standard workloads, e.g. as provided by the YCSB (Yahoo Cloud Serving Benchmark) framework (WEB (e)), with a main goal to provide the ground for the comparison of different NoSQL technologies. We use the methodology developed in our earlier EU funded research "Definition and Analysis of models for Advanced data Visualisation" (Rats, 2015). In contrary to YCSB approach we start workload definition from user business activities (like – show my urgent tasks) and their frequency. User business tasks are further decomposed into sequences of user interactions (i.e., user request that can be executed by one or more data requests without user intervention). User interactions are further decomposed as series of data requests. This allows us to create workload and to estimate performance of our search model for the given number of business users.

We use for the performance evaluation a list of data request sequences that includes search (e.g., full-text search inside document content), filtering and processing of aggregates, as well as document and task creation and modification. We make assumptions on frequencies of execution of data request sequences by an ECM user. The results of the research mentioned above is used to assume frequencies of execution of the data request sequences by an ECM user. Table 1 shows the sample of activities and user interactions used for performance tests.

Table 1. Sample user business activities and interactions.

Activity	**User interactions**	**Frequency (month)**
Add document	Add new document to the data store (addDoc)	50
Update document (1)	Select documents with a given status (filterDoc) and then update document status (changeDoc) of one of the documents	40
Update document (2)	Search documents against the keyword (search-content-or) then update document status (changeDoc) of one of the documents	10
Create task	Select documents with a given status (filterDoc) and then create task (addTask) for one of the documents	100
Update task (1)	Select tasks with a given status (filterTask) and then update the task status (changeTask) of one of the tasks	95
Update task (2)	Search documents against the keyword (search-content-or) then select a task of the document (filterTask) and then update task status (changeTask)	5
Select tasks	Select tasks with a given status (filterTask)	100
Select documents (1)	Select the documents with a given status (filterDoc) then select tasks of the one of documents (filterTask)	20
Select documents (2)	Select the documents with a given status (filterDoc) then open the document file (getDoc)	80
Select documents (3)	Select the documents of a given case (filterCase) then search in a document content for a given keywords (search-content-and)	100
Search a document (1)	Search a document content for a given keywords (search-content-or)	40
Search a document (2)	Search a document content for a given phrase (search-content-phrase)	25
Create an aggregate (1)	Create for a given year an aggregate of document counts by months and statuses (aggMonth)	50

The total workload of the defined flow for one ECM user is 985 activities or 1700 user interactions or 2100 data requests for parent-child (2000 for nested) model per month. This translates to 5.9 activities (about 12 data requests) per user per hour.

The performance have been measured for 2000 concurrent users on several test databases for two data models (nested and parent-child) and for different document volumes. The workloads have been generated for 5 minute and 30 minute test periods with 30 second warming phase for each case. The data request workloads for 2000 concurrent users totals to 24 thousand data requests per hour or 6.6 data requests per second in average. The average request execution time is 0.14 seconds for the parent-child model and 0.12 seconds for the nested model. Figure 5 shows the average execution times for the user interaction types. The performance results for filterTask request on nested model is not included because the model does not allow for effective implementation of task related searches.

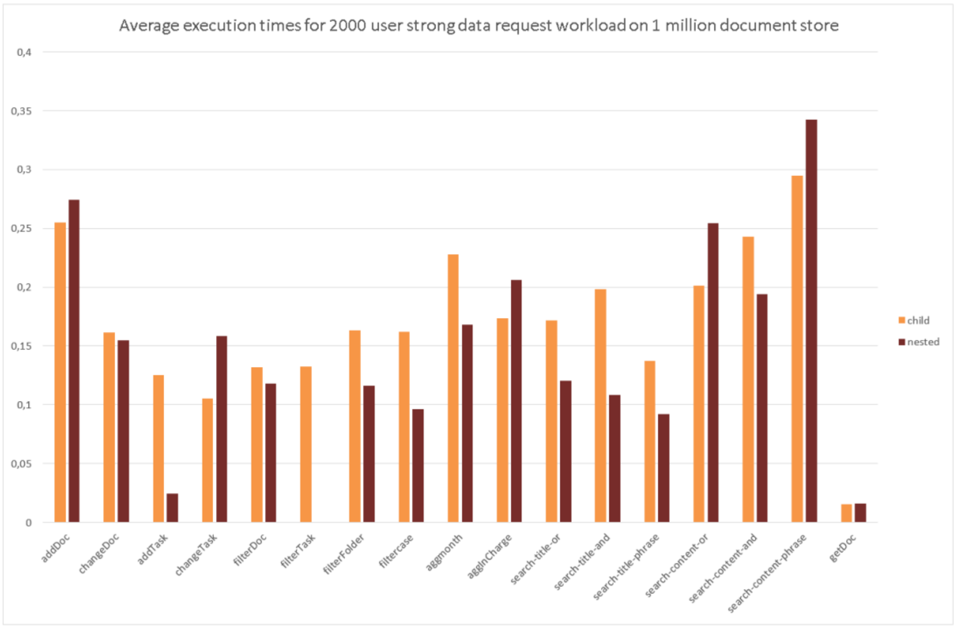

Figure 5. Average execution times for request types.

The test workloads have been executed on a one node 5 shard Elasticsearch database on a commodity server (CPU: Intel Core 2 Duo E6750 @ 2.66 GHz, RAM: 4 GB, HDD: 3 TB SATA).

The results are fully acceptable. It should be stressed here that one node solution does not provide for high availability as the crash of the server means the data store becomes unavailable. The smallest configuration from availability viewpoint is 3 node cluster with replication factor 1.

The test workloads have been executed on a one node 5 shard Elasticsearch database on a commodity server (CPU: Intel Core 2 Duo E6750 @ 2.66 GHz, RAM: 4 GB, HDD: 3 TB SATA).

The results are fully acceptable. It should be stressed here that one node solution does not provide for high availability as the crash of the server means the data store becomes unavailable. The smallest configuration from availability viewpoint is 3 node cluster with replication factor 1.

11. Conclusions and future work

We defined a polyglot persistence architecture consisting of two data stores – the primary store on SQL database and the secondary store on Elasticsearch. Data objects are inserted/updated into primary store and searched/accessed in secondary store. The primary store supports ACID transactions to ensure safe concurrent processing of data by multiple users while the secondary store features fast execution of large volumes of search requests on large volumes of data.

The model is validated on a 2000 user strong data request workload on a 1 million document database on a one node Elasticsearch database (see the bottom of the chapter 10 for the server machine specification). The workload generates about 6.6 data requests per second on average. The average request execution time is 0.14 seconds for parent-child model.

We will upgrade our model further to make more profit from the scalability of the Elasticsearch database. If we have all the searchable data in the secondary store it is not necessary anymore to keep all data in a primary store. This would allow to reduce the growth of the primary store which is an important advantage because of the inherent scaling problems of SQL technologies.

Users of different business roles may have different activity patterns and hence different execution frequencies for the mentioned data request sequences. The future enhancement of the workload generation process should consider this.

List of abbreviations

ACID	Atomicity, Consistency, Isolation, Durability. A set of properties of database transactions. A sequence of database operations that satisfies the ACID properties can be perceived as single logical operation on the data (called a transaction).
CAP	CAP (Consistency, Availability and network Partitioning) alias Brewer's theorem states that it is impossible for a distributed computer system to simultaneously provide more than two out of the three important guarantees - Consistency, Availability and network Partitioning.
ECM	Enterprise Content Management comprises the strategies, processes, methods, systems, and technologies that are necessary for capturing, creating, managing, using, publishing, storing, preserving, and disposing content within and between organizations.
EU	The European Union.
NoSQL	Not only SQL databases provides a mechanism for storage and retrieval of data which is modelled in means not restricted to the tabular relations of relational databases.
RBAC	Role-based access control. A method of regulating access to resources based on the roles of individual users. Access is the ability of a user to perform a task (e.g. view, create, or modify a data object).
SQL	Structured Query Language. The standard language for relational database management systems.

Acknowledgements

The research has received funding from the project "Competence Centre of Information and Communication Technologies" of EU Structural funds, contract No. 1.2.1.1/16/A/007.

References

Abramova, V., Bernardino, J., Furtado, P. (2014). Experimental Evaluation of NoSQL Databases. International Journal of Database Management Systems (IJDMS), 6(3). http://doi.org/10.5121/ijdms.2014.6301

Agrawal, D., Bernstein, P., etc. (2012). Challenges and Opportunities with Big Data: A white paper prepared for the Computing Community Consortium committee of the Computing Research Association.

Blair, B. T. (2004). An enterprise content management primer. Information Management Journal, 38, 64–66.

Brasetvik, A. (2013). Elasticsearch from the Bottom Up, Part 1.

Edlich, S. (n.d.). NOSQL Databases. Retrieved March 1, 2017, from http://nosql-database.org/

Fowler, A. (2015). NoSQL for Dummies. John Wiley & Sons, Inc.

Frank, C. (2012). Improving Decision Making in the World of Big Data. Retrieved February 28, 2017, from https://www.forbes.com/sites/christopherfrank/2012/03/25/improving-decision-making-in-the-world-of-big-data/#6292c9a51e85

Grahlmann, K. R., Helms, R. W., Hilhorst, C., Brinkkemper, S., van Amerongen, S. (2012). Reviewing Enterprise Content Management: a functional framework. European Journal of Information Systems. http://doi.org/10.1057/ejis.2011.41

Gupta, A. (2015). Why Couchbase over MongoDB? #SayNoToMongoDB. Retrieved January 20, 2017, from http://blog.arungupta.me/couchbase-over-mongodb-saynotomongo/

Hannan, T. (2015). Fundamentals of NoSQL Security.

Retrieved from http://basho.com/posts/technical/fundamentals-of-nosql-security/

Kampffmeyer, U. (2006). Enterprise Content Management ECM. White paper. Hamburg.

Korb, J., Strodl, S. (2010). Digital preservation for enterprise content: a gap-analysis between ECM and OAIS. In 7th International Conference on Preservation of Digital Objects (iPRES2010) (pp. 221–228). Vienna, Austria.

Makris, A., Tserpes, K., Andronikou, V. (2016). A classification of NoSQL data stores based on key design characteristics. Procedia - Procedia Computer Science, 97(October), 18–20. http://doi.org/10.1016/j.procs.2016.08.284

Marechal, L. (2015). MongoDB vs. Elasticsearch: The Quest of the Holy Performances. Retrieved January 20, 2017, from http://blog.quarkslab.com/mongodb-vs-elasticsearch-the-quest-of-the-holy-performances.html

Mcknight, W. (2014). Information Management Strategies for Gaining a Competitive Advantage with data. Elsevier.

Meng, Y. (2016). Cassandra 3.9 Security Feature Walk-through. Retrieved January 11, 2017, from https://www.pythian.com/blog/cassandra-3-9-security-feature-walk/

Nilsen, O. R. (2012). Enterprise Content Management in Practice. University of Agder, Kristiansand, Norway.

Oleson, J. (2015). Why Enterprise Search is Not the Same as Google Search - BA Insight Blog. Retrieved February 28, 2017, from http://bainsight.com/blog/why-enterprise-search-is-not-the-same-as-google-search

Potts, J. (2010). Alfresco, NOSQL, and the Future of ECM.
Retrieved from http://ecmarchitect.com/archives/2010/07/07/1176

Rats, J. (2015). Simulating user activities for measuring data request performance of the ECM visualization tasks. International Journal of Applied Mathematics and Informatics, 9, 96–102.

Rats, J., Ernestsons, G. (2013). Clustering and Ranked Search for Enterprise Content Management. International Journal of E-Enterpreneurship and Innovation, 4(4), 20–31. http://doi.org/10.4018/ijeei.2013100102

Shermin, M. An Access Control Model for NoSQL Databases (2013). Retrieved from http://ir.lib.uwo.ca/etd

Shirky, C. (2008). It's Not Information Overload. It's Filter Failure. Retrieved March 1, 2017, from http://www.mascontext.com/issues/7-information-fall-10/its-not-information-overload-its-filter-failure/

Simon, S. (2012). Brewer's CAP Theorem. CS341 Distributed Information Systems. Basel: University of Basel.

Solid IT. (n.d.). DB-Engines Ranking - popularity ranking of database management systems. Retrieved March 1, 2017, from http://db-engines.com/en/ranking

Strazdins, G. (2016). Data Version Control for Relational Databases: Small and Start-up Business Perspective. Baltic J. Modern Computing, 4(4), 978–993. http://doi.org/10.22364/bjmc.2016.4.4.23

Sullivan, D. (2015). NoSQL for Mere Mortals (1st Edition). Addison-Wesley Professional.

Vorhies, B. (2015). Polyglot Persistence?
Retrieved February 28, 2017, from http://data-magnum.com/polyglot-persistence/

Wiggins, A. (2009). SQL Databases Don't Scale. A Tornado of Razorblades. Retrieved March 1, 2017, from http://adam.herokuapp.com/past/2009/7/6/sql_databases_dont_scale/

Winder, D. (2012). Securing NoSQL applications: Best practises for big data security. Retrieved from http://www.computerweekly.com/tip/Securing-NoSQL-applications-Best-practises-for-big-data-security

WEB (a). Parent-Child Relationship. Elasticsearch: The Definitive Guide [2.x].
Retrieved March 3, 2017, from https://www.elastic.co/guide/en/elasticsearch/guide/current/parent-child.html

WEB (b). Practical Considerations | Elasticsearch: The Definitive Guide [2.x] | Elastic.
Retrieved March 3, 2017, from https://www.elastic.co/guide/en/elasticsearch/guide/current/parent-child-performance.html

WEB (c). Benchmarking Top NoSQL Databases.
Retrieved March 6, 2017, from http://www.endpoint.com/

WEB (d). How we reindexed 36 billion documents in 5 days within the same Elasticsearch cluster. Retrieved March 2, 2017, from https://thoughts.t37.net/how-we-reindexed-36-billions-documents-in-5-days-within-the-same-elasticsearch-cluster-cd9c054d1db8#.2h2jsthfo

WEB (e). Yahoo Cloud Serving Benchmark | research.yahoo.com. Retrieved March 6, 2017, from https://research.yahoo.com/news/yahoo-cloud-serving-benchmark

Received March 13, 2017, revised May 2, 2017, accepted May 3, 2017