Computing and Informatics, Vol. 38, 2019, 454–472, doi: 10.31577/cai_2019_2_454

DEVELOPING AND EVALUATING ECM DATA PERSISTENCE ARCHITECTURE

Juris Rats

RIX Technologies
Blaumana 5a-3
Riga, LV-1011, Latvia
e-mail: juris.rats@rixtech.lv

Abstract. A conceptual data persistence architecture and methodology to evaluate its performance is created. Results of the empirical research indicate that the architecture created is convenient for high intensity processing of large ECM data volumes. The synthesis of SQL and NoSQL technology allows to handle high volume transactions on current data and full-text search on large sets of history data. The history data is moved from SQL to NoSQL data store thus allowing to use a cluster of commodity hardware to store major volume of ECM data. Scaling out (increasing a number of cluster nodes) is less expensive than scaling up (buying a more powerful server hardware) that is normally needed to upgrade SQL database.

Keywords: Polyglot persistence, NoSQL, ECM, Elasticsearch, hot-warm architecture

Mathematics Subject Classification 2010: 68M14, 68U35

1 INTRODUCTION

Forrester research [10] shows 40 % of firms were implementing and expanding big data technology adoption. Another 30 % were planning to adopt big data in the next 12 months. 25 % annual growth for NoSQL technologies is predicted.

Large number of technologies (Google BigTable, Apache Hadoop, NoSQL databases of all types, etc.) evolved to address big data. They propel a number of important paradigm shifts; the most important being:

- understanding the importance of clustered solutions;
- understanding there is no "one best choice" for all cases [19];
- understanding it might be not enough to employ one data persistence technology for the use case.

The third shift caused the introduction of the polyglot persistence – a process for storing data in the best database available, no matter the data model and data storage technology [9].

ECM (Enterprise Content Management) solutions deal with a large and fast expanding amounts of data and user requests. Traditionally SQL databases are used to persist ECM data. The SQL solutions do not scale out well, but it is a general opinion that NoSQL databases are not good on transaction support hence they do not qualify.

The aim of the research is to show that ECM data can be split into two parts (current and history data) that each are manipulated differently. The current data is one that is created or updated and need transaction support. The history data is not updated but searched, aggregated or retrieved instead. The history data accounts for the major part of the data volume hence clustered NoSQL database might be the best fit for this part of data.

The MS SQL database is used to store and manipulate current data while Elasticsearch (ES) [3, 13] – to store and query current and history data. Current data is replicated from MS SQL to ES as it is changed while history data is removed from MS SQL and stored in ES exclusively.

The research aims as well to create a methodology and a tool for performance measurement of a data persistence solution on a representative ECM data and request flows consisting of randomly dispersed sequences of user requests. The methodology is based on results of the earlier research [17].

Section 2 outlines the related work. The typical workflow scenario, objects involved and data statistics are described in Section 3, Section 4 gives a brief description of the technology used, while Section 5 illustrates the architecture proposed by the research. They are followed by Section 6 describing test data, Section 7 describing the methodology used to evaluate the performance of the proposed architecture, and Section 8 analyzing the results of the evaluation. Section 9 outlines the conclusions and is followed by acknowledgements.

2 RELATED WORK

The term NoSQL (Not only SQL) was initially used by Carlo Strozzi [11] in 1998. More than 225 NoSQL platforms of various kinds have been developed since [8]. The new platforms generally are aimed at specific problem areas and there are no general suggestions on what solution would be the best for a specific use case.

Polyglot persistence paradigm has been researched by various authors. A contemporary outline is given by Sadalage and Fowler [18]. They write that using

a single database engine for all of the requirements usually leads to non-performant solutions; storing transactional data, caching session information, traversing graph of customers and the products their friends bought are essentially different problems [18]. Use of several technologies should be considered instead of sticking to one. Document store is the most convenient NoSQL technology for ECM [16], still the polyglot persistence approach would be the best fit here to use strong transaction support of relational database for data maintenance and NoSQL document store for fast information search and retrieval. Some solutions already use a similar approach, e.g. using MySQL database together with Elasticsearch [21].

Clustered solutions allow to cope with large and rapidly expanding data and request volumes, but they are inherently more complicated than their one server counterparts. The complexity relates, in particular, to the increased number of parameters to watch and configure to tune the solution as well to the methods of performance measurement.

To develop a performance measurement tool for the performance evaluation of the data persistence architecture proposed, the number of benchmarking tools have been reviewed, e.g. YCSB [1], BigBench [12], GridMix [14], PigMix [6], HBench:java [20] and GraySort [7]. GridMix, PigMix, HBench:java and GraySort are dedicated for benchmarking of specific systems. BigBench is a benchmark proposal dedicated for a specific business model (product retailer) and specific two layer data model with ETL backed data transfer between layers.

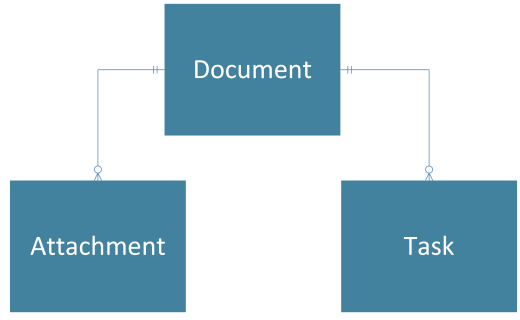

Figure 1. Data model

YCSB (Yahoo! Cloud Service Benchmark) is a generic framework and tool used for benchmarking of a number of distributed systems (Cassandra, HBase, MongoDB, Elasticsearch, Redis etc.). The shortcomings are as follows:

- It is assumed (see e.g. [5, 12]) that a request flow (workload) consists of mutually independent data requests; in reality user requests are grouped in a sequences of dependent requests, the model should be expanded to cover this.
- All test data ([5, 12]), including the texts used for full-text search, is generated – the generated text does not represent real data well when measuring search request performance.

3 ECM DATA ANALYSIS

The research is focused on a document management, but the patterns are the same for web content management, records management and workflow management as well. Figure 1 shows the data model used in our research. Table 1 gives the detailed description of the data attributes.

	D	A	T	Comments
docNum	x	x	x	Document number. Replicated from the document object to all child attachment and task objects.
inCharge	x	x	x	Person in charge of the document (Document and child Attachments) or for the task (Task object).
author			x	Creator of the task.
canRead	x	x		List of user ids having access to the document and its attachments because they are authors or persons in charge of some child task of the document. Replicated to the child attachment objects.
case	x	x		Case the document is attached to. Replicated.
docType	x			Document type.
date	x	x	x	Indexing date and time of the document. Replicated to all child attachments and tasks. Determines the ES index the document and its children are allocated to.
deadline			x	Task deadline.
folder	x	x		Folder the document is attached to. Replicated.
project	x			List of projects a document is attached to.
status	x		x	Document or task status.
summary	x	x		Document title, attachment title.
content		x		Attachment content.
name			x	Task type.
comment			x	Task comments.

Table 1. Document, attachment and task attributes

The document here is a placeholder for zero or more attachment files, that an organization receives (or sends) in one go. The task is an action that an employee has to perform on the document to move it along the workflow. Tasks are of several types and multiple tasks may be related to a document.

One of regular scenarios is – an organization receives e-mail with one or several attachments from a partner or a customer (the document), a new document is created, the document and attachments are indexed, a workflow of tasks (assigned to one or more employees) is manually or automatically attached. Employees complete tasks (and/or assign new ones), create a response document (with one or several attachments) that is sent back to the correspondent. Described data items form the major part of data volume for document management process.

The analysis shows that users mostly interact with the current data and the rest is accessed scarcely. Figure 2 shows this dependency of user activity volume from the document age for 5 customer databases (K1–K5). The figure shows percentage of data requests (mixed read, write, search, aggregate) addressed to data of each document age group (up to 1 month, 1 to 2 months, etc.).

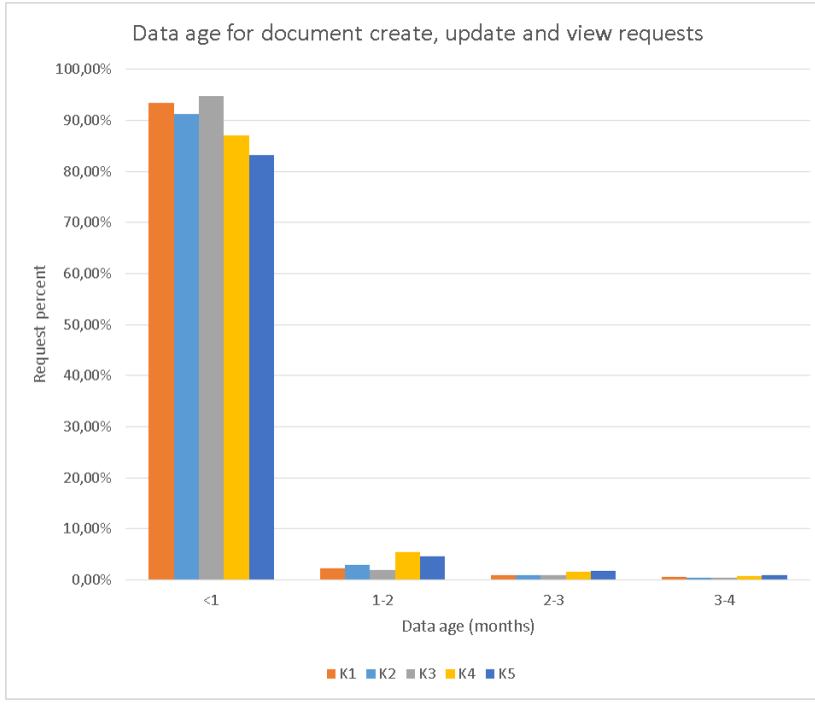

Figure 2. Data age for document create, update and view requests

To make use of the fact that most of the data is accessed scarcely we should build our persistence model in a way that allows separate storage and processing of frequently and scarcely used data. We propose to create two data stores:

- Current data store for create, update and delete requests;
- General data store for search, aggregation and retrieval.

This model would allow to use separate persistence technologies for transaction support (Current data store) and for search and aggregation in a large, expanding data volumes (General data store).

4 ELASTICSEARCH

Elasticsearch (ES [3, 13]) is a search engine built on top of Apache Lucene [15] – perhaps the most advanced, high-performance, and fully featured search engine library in existence today. Elasticsearch uses Lucene internally for indexing and searching, but it aims to make full-text search easy by hiding the complexities of Lucene. Elasticsearch is used for our research because of its advanced text searching features and because it is:

- a distributed real-time document store where every field is indexed and searchable;
- a real-time analytics engine;
- capable of easy scaling to hundreds of servers and petabytes of structured and unstructured data.

Unlike the relational databases ES stores all the data into indexes. Indexes store both the data and all the information necessary for search. ES index does not have a predefined structure, but it can be configured (through field mappings) according to the types of search anticipated for the particular fields. One might define if the particular field is to be searched or not, should it be ready for full-text search, ranged search, exact match, geographic search, etc.

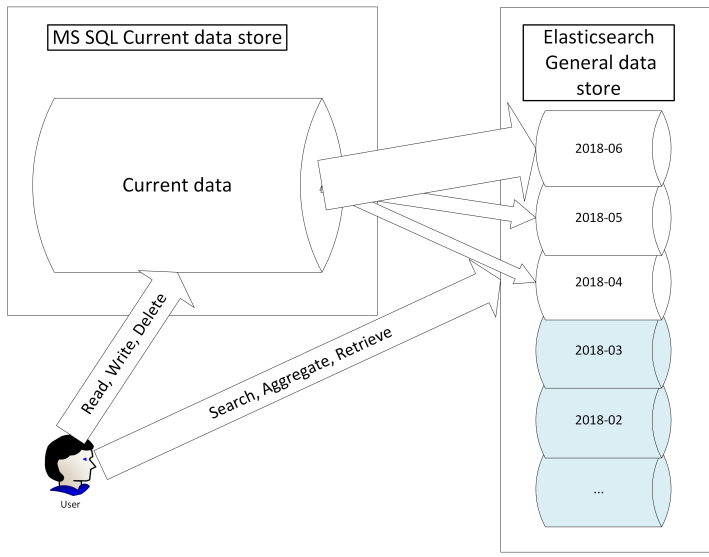

Figure 3. Proposed architecture

5 THE PERSISTENCE ARCHITECTURE

Figure 3 shows the proposed architecture. General data store consists here of a time dependent ES indexes. Time-dependent means here that an index contains data related to a particular time period (e.g. a month or a year). This allows to store data for different time periods on different nodes in a distributed infrastructure.

The following features are shown there:

- user creates, updates and deletes data in the Current data store;
- changes in the Current data store are replicated to the General data store;
- user searches, aggregates and retrieves data from the General data store;
- data in the General data store are split into time dependent indexes;
- as time passes indexes are switched to read-only mode (indexes highlighted in blue);
- as long as indexes of the General data store are switched to read-only mode, the data of the matching time periods may be removed from the Current data store.

Indexes in blue thus differ from the white ones because they are read-only and they may have no backing data in the Current data store. When implementing the architecture customer may decide when to make time periods read-only and when to remove data from the Current data store. Customer may decide as well to keep all the history time periods online, or put them offline at some point. The latter option allows to reduce infrastructure costs at an expense of increased latency for some rare user requests.

5.1 Hot-Warm Architecture Model

As concerns the General data store, this research follows the hot-warm architecture paradigm [2] for separate handling of frequently and scarcely used data. The hot-warm paradigm suggests to use different groups of cluster nodes to store frequently used (hot) data and scarcely used (warm) data. In respect to our data model new time periods are allocated to hot nodes and older time periods are switched to warm nodes when appropriate. ES allows to do this easily and transparently from the application. One has to issue a simple request that changes appropriate attribute of the index and ES automatically moves the index from hot nodes to some warm nodes. It is important to stress that moving data from hot to warm nodes concerns only the General data store, and it is independent from the process of moving data from the Current to General data store.

Figure 4 shows hot-warm ES cluster with 2 hot nodes and 3 warm nodes. Cluster has as well 3 master eligible nodes. These nodes elect a single master node of the cluster that is responsible for the cluster management functionality. Other two are here to replace master in case of emergency. 3 master eligible nodes and two of them available is the minimal configuration for the healthy cluster as this prevents

so called split brain scenario [2] when more than one node is elected (because of the lost communication) as a master. This situation is dangerous as two masters act on a cluster data concurrently that may lead to data corruption or loss.

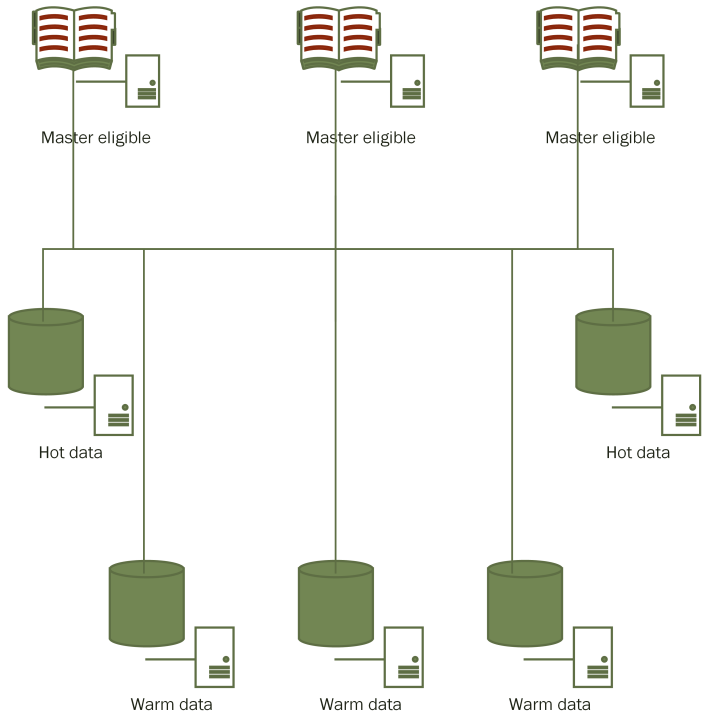

Figure 4. Hot-warm Elasticsearch cluster

With the architecture in place we will discuss below its advantages.

5.2 No Locking

ES uses Lucene [15] immutable indexes thus there is no need to lock index when writing data [3]. New index segments are created to index new data instead while index segments are merged in background later on. Thus General data store is available for search and data retrieval no matter how intense is the flow of new data replicated from the Current data store.

Downside of the immutable index technology is that newly indexed data does not become available for search instantly but with a delay instead. The delay varies from below a second normally to tens of seconds or more when system is heavy loaded. In contrary to the relational database case this does not result though in all search requests waiting while locks are released. Latest updates might not be included in the search results instead. ES provides several options to deal with this

problem – application may return the control to user not waiting for index refresh (to proceed with his work) or waiting for index to be refreshed (if user wants to see his changes before to proceed).

5.3 Scale Out

ES index consists of number of shards. When the data volume grows new nodes can be added to the cluster. ES automatically relocates shards when new nodes added. Thus ES index with e.g. 5 shards can run on 1 to 5 node cluster.

Replicas allow to scale out ES database even more. Primary shard and its replicas are allocated each to a different cluster node to achieve this. ES index with 5 shards and one replica can run on 2 to 10 node cluster.

Parameter	Comments
Time phased indexes	Major part of user requests is directed to the current data, that way major part of the all requests may be addressed to a small part of all indexes; this makes the request lighter and decreases the response time.
Read-only indexes	ES provides for optimization of read-only indexes; normally ES index shard consists of multiple (several tens to hundreds) segments, every search request is executed against all the segments; if index is not supposed to be changed anymore, segments can be merged into one; this speeds up search requests.
Hot-warm architecture	Having separate groups of cluster nodes for current (hot) and history (warm) data allows to deploy more powerful hardware for hot nodes to support low search latency (and save money on hardware for warm nodes).

Table 2. Means to improve search speed

5.4 Availability and Fast Search

Replicas are redundant data copies. Thus in addition to decreased search request latency they provide increased data availability. Index shard is available if the primary shard or one of replicas is available.

Elasticsearch has proven to be one of the fastest and richest search engines capable of handling very large data and user request volumes. The proposed architecture provides a couple of means to profit from these ES advantages (Table 2).

6 TEST DATA

The test data stores are created following the methodology presented in [17], elaborated and tuned for the current use case. Values for all but the fields employed

for the full text search are generated according to the frequencies calculated out of the real data in five customer databases (see Section 3). Texts for the two fields used for performance measurement of full-text search (attachment content and task comments) are extracted from Common Crawl – the open repository of web crawl data [4]. The data set of September 2017 is used that contains extracted texts from more than 2 billion pages in English or similar (Latin based charset) languages. This enables for creating of large data sets usable for full-text search.

ES hot-warm cluster of 6 nodes is created in MS Azure cloud and three large volume test data stores are generated (using statistics of the customer databases mentioned in Section 3) for performance tests (Table 3). Store names indicate data model and time span used. Shared model uses shared index for all three data objects (documents, attachments, tasks) while multi model stores each object in a separate index (see Section 4 for ES storage explanation). Yearly model uses index for a year of data while monthly – for a month of data.

Store	Time Span	Index Shards	Object Count (millions)	Volume (GB)
Shared-year	6 years 2 months	35	1140	1 200
Shared-month	14 months	14	145	160
Multi-month	14 months	42	145	160

Table 3. Test databases

7 METHODOLOGY FOR PERFORMANCE EVALUATION

The aim of the evaluation is to measure how the solution performs:

- on large volumes of data specific for the use case in question;
- on a user request flow specific for the use case in question;
- on a specific cluster configuration one has or would like to change.

The methodology developed in our earlier research [17] is expanded to:

- support time partitioned data model to allow separate management of current and history data;
- support specific user request flows for current and history data;
- comprise a number of cluster configuration parameters.

Below these three dimensions of the proposed methodology are described in more detail.

7.1 Time Partitioned Data Model

The data storage is split into time specific ES indexes (see Section 4). The yearly and monthly indexes are analyzed for comparison. The date field of the document

determines an index where the document and all its child attachments and tasks are routed to.

To ensure realistic search results test database is generated according to the following principles:

- Document, attachment and task metadata are generated in accordance with the frequencies typical for the real databases (based on statistics of 5 customer databases, described in Section 3).
- Attachment content and task comment data is extracted from the commoncrawl.org repository that contains texts of the real web pages; these fields are used for full-text search.

7.2 User Request Flow

The performance measurements are based on a user interaction flow (interaction schedule) model developed in our earlier research [17]. The user interaction flow is decomposed into user business activities (like – show my urgent tasks). The business activities are represented as sequences of user interactions (i.e., user request that can be executed by one or more data requests without user intervention) as shown in Figure 5 in a portion of a sample user interaction flow specification. User interactions are further decomposed as series of data requests not shown in a specification sample.

```
{
    "sequences" : [
        { "seq" : ["aD"], "count" : 100},
        { "seq" : ["aD","aA"], "count" : 400},
        { "seq" : ["aD","aA","aT"], "count" : 100},
        ...
        { "seq" : ["sFPa","gA"], "count" : 5}
    ],
    "tasks" : {
        "aD" : { "archi" : false, "pars" : ["addDoc"],"tf" : 0},
        "cD" : { "archi" : false,  "pars" : ["changeDoc"],"tf" : 0},
        ...
        "gT" : { "archi" : false,  "pars" : ["getTask"],"tf" : 0}
    },
    "tf" : [
        {"freq" : 90, "threshold1" : 1000, "threshold2" : 5000},
        {"freq" : 90, "threshold1" : 2000, "threshold2" : 8000},
        {"freq" : 80, "threshold1" : 5000, "threshold2" : 15000}
    ]
}
```

Figure 5. Sample user request flow description

The specification describes sequences (user business activities), tasks (user interactions and data requests) and time functions (tf). A sequence description:

- lists user interactions (tasks) of the sequence, e.g., a sequence consisting of two user interactions – aD (add a document) and aA (add an attachment),

- specifies an average times (per year) an average user would run the sequence (user business activity).

A task (user interaction) description specifies if the interaction is addressed to current data ("archi": false), specifies the user interaction name (addDoc) and optional parameters, and specifies the time function used for performance measurements.

A time function description (tf) specifies what latency (in milliseconds) has to be assured for what percentage of the user interaction (e.g. 1 second for 90 % of interactions, 5 seconds for all the interactions). Figure 6 shows the first 30 seconds of the sample evenly distributed user interaction schedule for 8000 users.

To evaluate performance a set of user interaction sequences is used that includes search (e.g., full-text search inside document content), filtering and processing of aggregates, as well as document, attachment and task creation and modification. The results of the research [17] are used to assume frequencies of execution of the user interaction sequences by an ECM user. The list of user interaction sequences includes the requests to the current as well to the history data. User interaction simulation model is tuned to the time dependent index structure of our architecture model allowing to explicitly direct a part of requests to a particular index (or several indexes).

7.3 Cluster Parameters

Measuring a performance of a clustered solution is a challenge because we have to assess both performance of a healthy cluster and emergency performance (recovery from node unavailability). On a production cluster we can measure mainly a performance of a healthy cluster (as it mainly is healthy). Therefore we need a sibling cluster to play with crashes and recoveries. This is a costly option hence a cloud infrastructure should be considered as the sibling cluster might be created there when needed and disposed afterwards.

The performance tests are executed on a number of configurations. A number of parameters analyzed are described below.

Data model. Elasticsearch nested objects and parent-child constructs are not used as they are resource hungry. Relationship between documents and child attachments and tasks are maintained at an application level. Two different data models are evaluated – all three types of objects stored in a single ES index (shared model), and each object type stored in separate index (multi model). The shared model needs 3 times less indexes while the multi model takes less storage space.

Time span for a single index. Yearly and monthly index is tested. The first one needs less indexes, the second one allows more control over data (e.g. indexes may be moved from hot to warm nodes on a monthly basis).

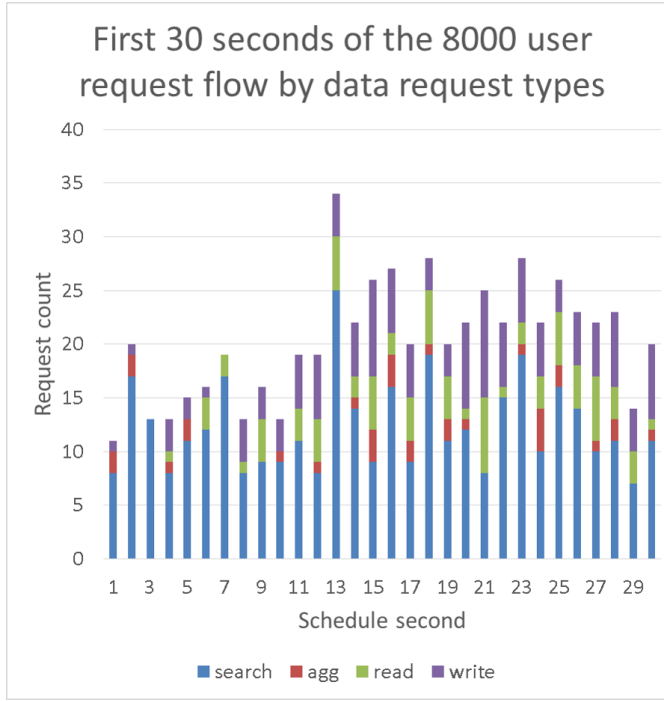

Figure 6. Sample request flow by request types

Cluster configuration and node count. The impact of overall node count, as well as of several role configurations (e.g. dedicated master eligible nodes versus warm nodes configured as master eligible nodes) are assessed.

Number of replicas. Cluster performance is evaluated for 1-3 replicas for an index shard.

Node hardware characteristics. RAM and heap volume as well as disk volume and speed impact the overall cluster performance. Several configurations of these parameters are evaluated.

Overall data volume. The performance of the cluster for data object volumes worth of 3 to 6 years of data is evaluated. The yearly amount of data is 190 million of data objects (documents, attachments and tasks).

User interaction flow characteristics. Measurements are performed for user interaction flows of different volumes (up to average 27 user interactions per second). User interaction flows are generated to contain mixed sequence of search, aggregation, retrieval, as well as data insert and update requests. User interaction flows are generated for 13 minute test runs (3 minutes warming and 10 minute rating phase) and contain 230 to 18 870 data requests each.

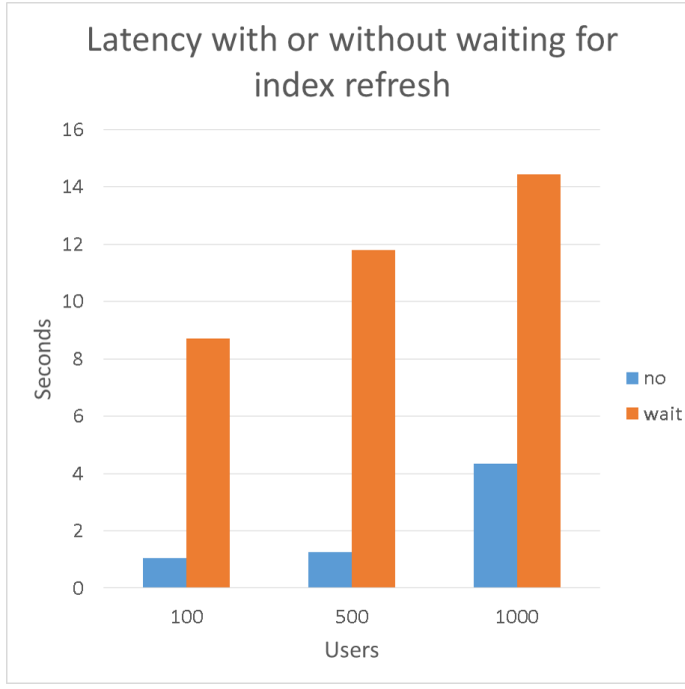

Figure 7. Latency with or without waiting for index refresh

8 RESULTS OF THE MEASUREMENTS

240 tests are executed to measure different configurations of above described parameters. One test here is an execution of a 13 minute (3 minute warming and 10 minute rating phase) long request flow.

Generally the analysis of the results supports the opinion dominant in ES support forums and elsewhere that cluster and data model parameters must be tuned for a particular use case. A number of interesting patterns have been observed and they are explained in the Sections 8.1, 8.2 and 8.3.

8.1 RAM and Index Volume Ratio

ES node loads index data from disk to RAM when started. The volume of data to be loaded in RAM depends on the total index data volume and on field mappings (see Section 4). A number of configurable ES parameters may influence this (e.g. types of indexing mapped for particular fields). If RAM volume of the node is too small to load all the necessary data, it is not possible to start the node. The RAM and index volume ratio (to be loaded on node start-up) thus must be carefully watched by the administrator. Our tests show that at least 7 GB RAM is necessary for a node to

handle 1.2 TB of disk index volume for our index mappings. To be on the safe side it is better to have excess amount of RAM though.

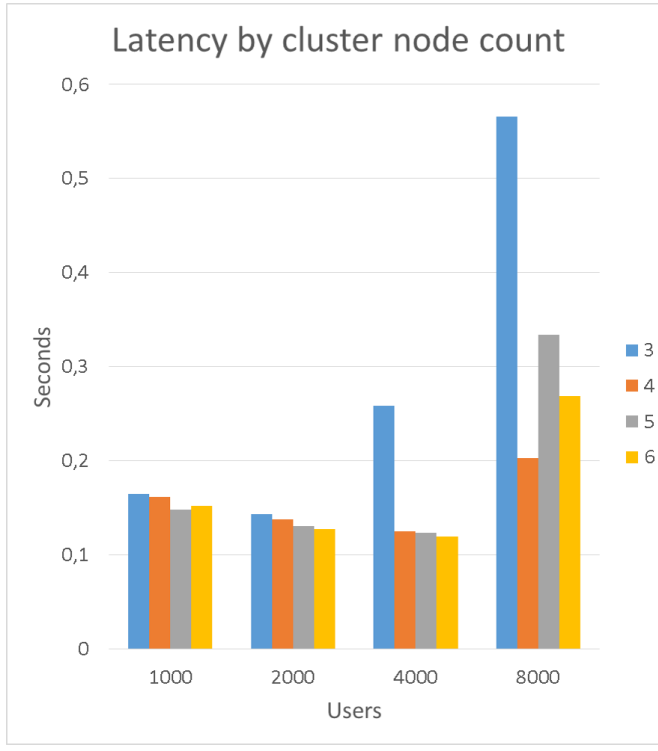

Figure 8. Latency of 3-6 node cluster for 7 GB node RAM

8.2 Index Refresh

ES by default refreshes indexes every second. This means that every second ES takes the index segments with freshly indexed data and makes them available for search (further indexing requests go to newly created segments). New data gets available to search though only when this index refresh process is complete. When cluster load grows index refresh process may slow down considerably to take several tenths of seconds or more. Figure 7 shows the pattern for shared-year data store (1.14 billion objects), cluster with 5 data nodes, 1 replica, 7 GB RAM, HDD disks.

Index requests must be implemented to wait for index refresh only when it is absolutely necessary, e.g. if user needs to immediately see the new/changed data. Index requests that do not wait for index refresh perform better when cluster load grows.

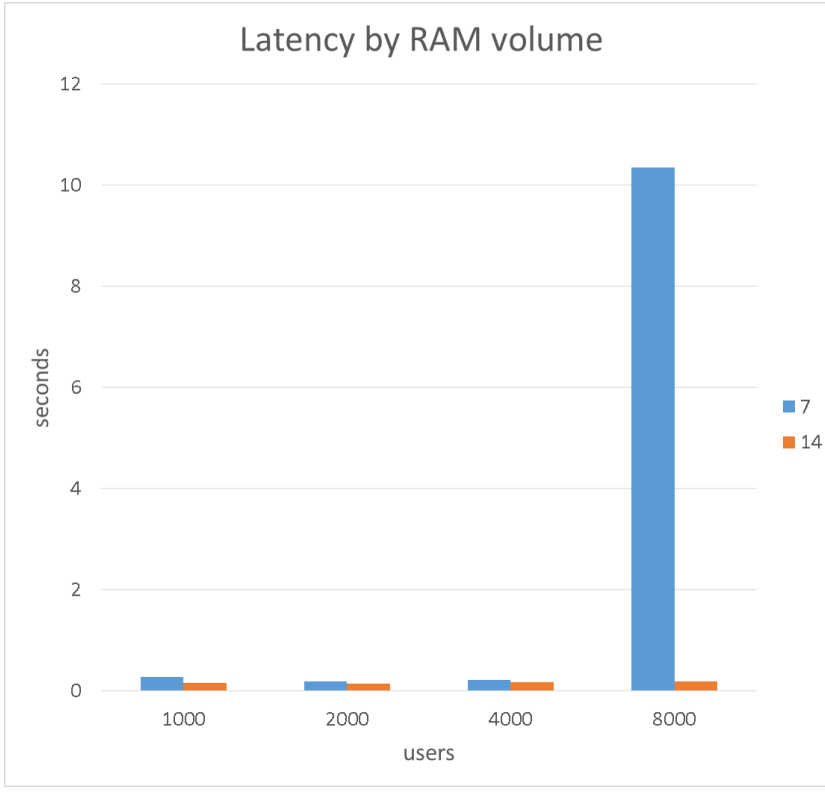

Figure 9. Latency by node RAM volume for cluster with 3 nodes

8.3 Scale Up or Scale Out

Scaling up or out are two options that frequently are compared. The performance tests executed show that ES cluster node performs better when it has plenty of RAM volume. Nodes with 7 GB and 14 GB RAM are tested and it appears (see Figures 8 and 9) that for large data request amounts it is better to expand RAM from 7 GB to 14 GB than to double the cluster node count.

For larger RAM volumes it might be more convenient to increase node count as smaller nodes mean less damage and shorter recovery times when a node gets down.

8.4 Cloud Infrastructure Costs

The performance tests show what could be the potential infrastructure costs for a data store and user request volume we are interested in (see Table 4).

Data volume (years)	6 years 2 months
Data volume (million objects)	Documents – 120, attachments – 120, tasks – 900
Data request flow	Up to 27 per second
Hot node specifications	14 GB RAM, SSD 1 TB
Warm node specifications	14 GB RAM, HDD 2 TB
MS Azure cloud cost per month (EUR)	2 090

Table 4. MS Azure cloud cluster configuration and costs

The data and request volumes correspond to the ones of comparatively large organization (8 000 users, 1.3 million documents per month).

9 CONCLUSIONS

The polyglot persistence architecture is defined consisting of two data stores – the Current data store and the General data store. Data is inserted/updated into the Current data store and searched/accessed in the General data store. MS SQL is employed for the Current data store to ensure strong transaction support while Elasticsearch is used for the General data store to provide fast execution of large volumes of search requests on large volumes of data. The General data store is split into time dependent indexes and handled by a clustered ES solution of a hot-warm architecture to allow for separate management of current and history data.

The methodology and tools for performance evaluation of the proposed architecture are created. The methodology allows for creation of test data and test workloads used to measure the performance and to evaluate impact of a number of parameters (node count and configuration, replica count, RAM volume, disk speed, total data amount, user request volume, etc.). Analysis of the evaluation results uncovers a number of interesting patterns. The results on several sets of parameters may be used to evaluate and compare alternative ways of cluster scaling (e.g. more nodes against more node RAM) for a production system.

The architecture is considered to be convenient for ECM solutions of large data and user request amounts. The measurements performed show that MS Azure hosted cluster configuration costing about EUR 2 000 per month would handle database of 1.14 billion objects (documents, attachments and tasks) for average 27 per second flow of mixed data create, update, read, search and aggregate requests.

Acknowledgement

The research is co-funded by the European Regional Development Fund (ERDF) (project No. 1.2.1.1/16/A/007).

REFERENCES

[1] Abubakar, Y.—Adeyi, T. S.—Auta, I. G.: Performance Evaluation of NoSQL Systems Using YCSB in a Resource Austere Environment. International Journal of Applied Information Systems, Vol. 7, 2014, No. 8, pp. 23–27.

[2] Bennacer, S.: "Hot-Warm" Architecture in Elasticsearch 5.x. 2017. https://www.elastic.co/blog/hot-warm-architecture-in-elasticsearch-5-x.

[3] Brasetvik, A.: Elasticsearch from the Bottom Up, Part 1. 2013, available at: https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up.

[4] Common Crawl, open repository of web crawl data. Available at: http://commoncrawl.org/.

[5] Cooper, B. F.—Silberstein, A.—Tam, E.—Ramakrishnan, R.—Sears, R.: Benchmarking Cloud Serving Systems with YCSB. Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10), 2010, pp. 143–154, doi: 10.1145/1807128.1807152. Available at: https://www2.cs.duke.edu/courses/fall13/cps296.4/838-CloudPapers/ycsb.pdf.

[6] Dai, J.: PigMix. 2013, available at: https://cwiki.apache.org/confluence/display/PIG/PigMix.

[7] Dvorský, M: History of Massive-Scale Sorting Experiments at Google. 2016, available at: https://cloud.google.com/blog/big-data/2016/02/history-of-massive-scale-sorting-experiments-at-google.

[8] Edlich, S.: NOSQL Databases. Available at: http://nosql-database.org/.

[9] Foote, K. D.: Utilizing Multiple Data Stores and Data Models: Is Polyglot Persistence Worth It? 2017, available at: http://www.dataversity.net/utilizing-multiple-data-stores-data-models-polyglot-persistence-worth.

[10] Forrester Forecasts Big Data Tech Market Will Grow ∼3x Faster Than Overall Tech Market. 2016, available at: https://go.forrester.com/press-newsroom/forrester-forecasts-big-data-tech-market-will-grow-3x-faster-than--overall-tech-market/.

[11] Fowler, A.: NoSQL for Dummies. John Wiley and Sons, Inc., Hoboken, New Jersey, 2015.

[12] Ghazal, A.—Rabl, T.—Hu, M.—Raab, F.—Poess, M.—Crolotte, A.—Jacobsen, H.-A.: BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13), New York, June 2013, pp. 1197–1208, doi: 10.1145/2463676.2463712.

[13] Gormley, C.—Tong, Z.: Elasticsearch: The Definitive Guide. O'Reilly Media, Inc., Sebastopol, CA, 2015.

[14] GridMix. Available at: https://hadoop.apache.org/docs/r1.2.1/gridmix.html.

[15] McCandless, M.—Hatcher, E.—Gospodnetić, O.: Lucene in Action. 2nd Edition. Manning Publications Co., 2019. Available at: https://livebook.manning.com/#!/book/lucene-in-action-second-edition.

[16] Potts, J.: Alfresco, NOSQL, and the Future of ECM. 2010, available at: http://ecmarchitect.com/archives/2010/07/07/1176.

[17] Rats, J.: Simulating User Activities for Measuring Data Request Performance of the ECM Visualization Tasks. International Journal of Applied Mathematics and Informatics, Vol. 9, 2015, pp. 96–102.

[18] Sadalage, P. J.—Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, Inc., Upper Saddle River, New Jersey, 2012.

[19] Vorhies, B.: Polyglot Persistence? 2015, available at: http://data-magnum.com/polyglot-persistence.

[20] Zhang, X.—Seltzer, M.: HBench:Java: An Application-Specific Benchmarking Framework for Java Virtual Machines. ACM 2000 Java Grande Conference, 2000. Concurrency and Computation: Practice and Experience, Vol. 13, 2001, No. 8–9, pp. 775–792, doi: 10.1002/cpe.578. Available at: https://www.eecs.harvard.edu/margo/papers/javagrande00/paper.pdf.

[21] Zwierzynski, P.: How We Use Elasticsearch. 2015, available at: https://www.bigeng.io/how-we-use-elasticsearch.

Juris Rats is Project Manager at RIX Technologies, Riga, Latvia. He is the Latvian University graduate and there he received his Ph.D. in computer science. He was working as a programmer, software and business analyst, IT project and team manager, university lecturer and researcher in several IT companies and universities in Latvia as well as in Germany and UK. His research interests include big data technologies, NoSQL databases, distributed systems and ECM systems. Lately, he was involved in four successfully completed ERDF co-financed research projects. Currently, he has been working on ERDF co-financed research project on use of hybrid data models for effective processing of enterprise data.