Baltic J. Modern Computing, Vol. 6 (2018), No. 3, 304-319
https://doi.org/10.22364/bjmc.2018.6.3.06

Polyglot Persistence Architecture for Enterprise Content Management

Juris RĀTS

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

juris.rats@rixtech.lv

Abstract. The aim of the research is to create and evaluate polyglot persistence architecture for an Enterprise Content Management solution. MS SQL database is used for Current data store that handles the current data while Elasticsearch – for General data store where both current and history data is persisted and queried. The general data store is represented by time (e.g. monthly) spanned indexes on an Elasticsearch cluster of a hot-warm architecture.

The proposed architecture is evaluated on a MS Azure cloud hosted Elasticsearch cluster on a several test databases of volume up to 1.14 billion of objects. Various parameter configurations are tested to explore for performance patterns. Results of the performance tests are outlined and suggestions are brought forward on resilience management, performance measurement and management of the cluster in production environment.

Keywords: Big Data, Polyglot persistence, NoSQL, ECM, Elasticsearch, clustered processing, hot-warm architecture.

1. Introduction

Big data technologies are being adopted by enterprises/institutions at an increasing speed. Forrester research (Vassallo, 2016) 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. The same research forecasted 25% annual growth for NoSQL technologies.

Big data has caused several important paradigm shifts. The first is the understanding that scaling up does not pace up with the increase of data and user request levels. Clustered data processing has evolved to provide easy scaling out.

The second is understanding there is no "one best choice" for all cases (Vorhies, 2015). A myriad of NoSQL solutions have been created 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).

The third is the notion of polyglot persistence. Polyglot persistence is a process for storing data in the best database available, no matter the data model and data storage

technology (Foote, 2017). The term comes from understanding that the data persistence technologies have their limits. The booming volumes of data make it hard to manipulate all data by a single technology. Polyglot persistence paradigm suggests one should analyse data manipulation patterns first and then search for appropriate persistence technology for each pattern.

Polyglot persistence paradigm allows to look differently on a number of important issues. For example – it is believed that SQL databases are good for applications demanding strong transaction support while NoSQL technologies are not. Banking, order management, storage management etc. solutions hence should use SQL database for a persistence layer as they have a strong transaction support. Having polyglot persistence in mind we should be asking – what data manipulation patterns do we have and what technologies are best for each pattern. For the cases above there are data types that need transaction support and there are data types that don't. Looking at the money transfer use case, for example, we could notice that debtor and creditor accounts are involved in money transfer transaction. Transaction record is created as a result of the transaction that is not modified anymore and used for search, aggregation and retrieval only. Transaction records do not need transaction support and they form a large and fast growing part of a total data amount. Therefore NoSQL databases are a good choice for this part of banking system's data.

The aim of our research is to create and evaluate polyglot persistence architecture for an Enterprise Content Management (ECM) solution. We need a transaction support here to support creation and modification of current data. Major part of data volume is history data though and is used only for search, aggregation and retrieval. No transaction support needed for this part of data. We use MS SQL database in our architecture to store and manipulate current data. Elasticsearch (ES) is used 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.

2. Related work

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 are focusing on Document stores (e.g., MongoDB, CouchDB) and Elasticsearch, that is marketed as a search engine, but in fact is a document store as well. 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).

Polyglot persistence paradigm has been researched by various authors. A contemporary outline is given by Sadalage and Fowler. 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 (Sadalage & Fowler, 2012). Use of several technologies should be considered instead of sticking to one. 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.

ES is used mostly as a secondary database. Data is replicated to ES from a primary database and then used to provide fast and advanced search. Although ES is used as a primary database in a number of cases there are still resilience issues unsolved ("Elasticsearch Resiliency Status | Elastic," n.d.) that have to be addressed for use cases where data loss of any kind is unacceptable. One can find some recommendations out there how to implement ES as a primary database, like using ES Snapshot and Restore or using the messaging processor Kafka ("Elasticsearch as a primary database - Elasticsearch - Discuss the Elastic Stack," 2018). This area is poorly explored though.

3. ECM data analysis

ECM covers a wide area of functionality that includes (Kampffmeyer, 2006):

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

We will focus on a document management in our research 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** outlines the data attributes.

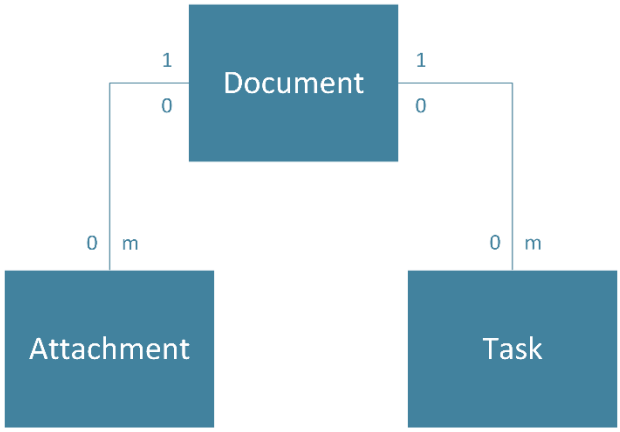

Figure 1. Data model

Table 1. Document, attachment and task attributes.

Object	Attributes
Document	Document number, document title, document type, document date, document status, folder, case, person in charge, list of authorised users
Attachment	Parent document number, parent document date, folder, case, person in charge, list of authorised users, attachment title, attachment content
Task	Parent document number, parent document date, task creator, person in charge, task deadline, task type, task status, task comment

List of authorised users here contains 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.

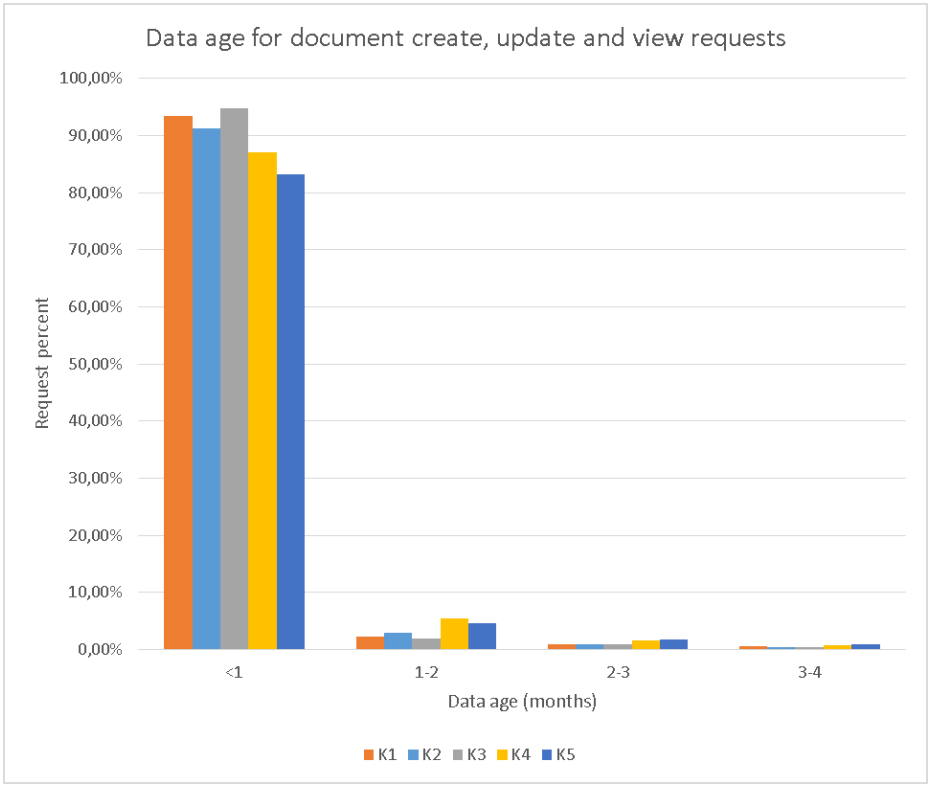

Figure 2. Data age for document create, update and view requests

Our analysis of customer data request statistics shows that mostly users are interacting with current data and the rest is accessed scarcely. **Figure 2** shows the pattern for 5 customer databases.

The patterns show major part of the data is scarcely accessed and scarcely updated. To make use of scarcely accessed pattern we should build our persistence model in a way that allows separate storage and processing of frequently and scarcely used data. The scarcely updated means it would be handy to create two stores for our data:

- 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. The persistence architecture

The General data store contains current as well as history data of the organisation. The volume of the latter grows as time passes. The transaction management can be delegated to the Current data store, thus it is an obvious choice to use clustered, horizontally scalable solution to support the General data store. As long as transactions are ruled out we have technologies available that provide fast search and aggregation on a very large data and request volumes. We selected ES as it is one of the most advanced search engines available and provides functionality of NoSQL document database (Rats, 2015). MS SQL is used as a persistence technology for the Current data store.

Figure 3 shows the proposed architecture.

General data store consists here of a time related ES indices. Following features are shown in the figure:

- 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 indices;
- As time passes indices are switched to read-only mode (blue highlighted periods); the period data is removed from the Current data store simultaneously

Blue time periods thus differ from the white 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 if 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.

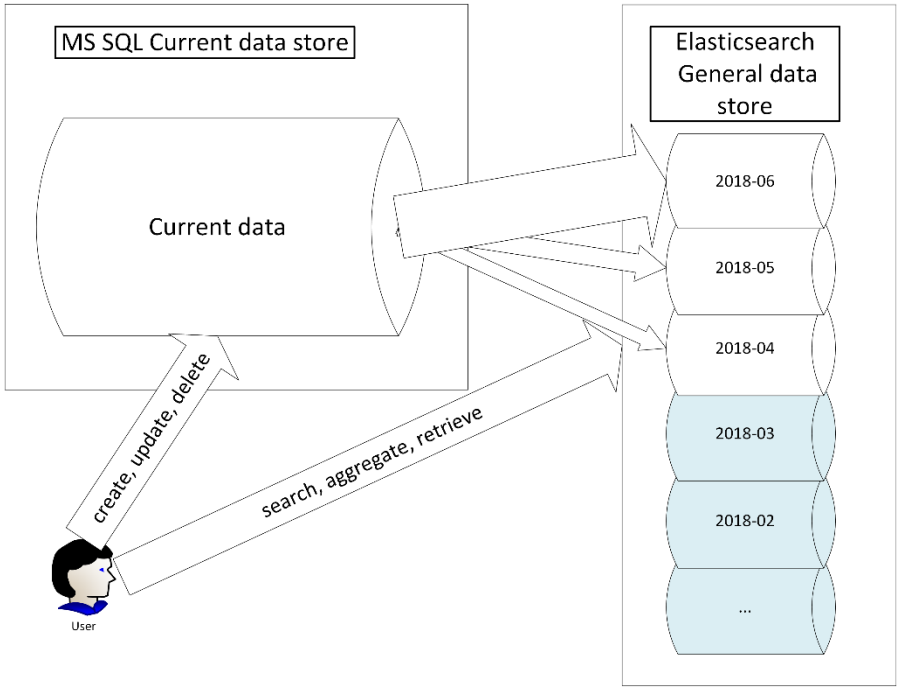

Figure 3. The persistence architecture.

As long as history data may be not present in the Current data store we have to decide how to handle creates, updates and deletes for time segments not in the Current data store. The persistence solution has to:

- restore in the Current data store from the General data store the data objects involved in the transaction;
- make writable the involved time period in the General data store;
- execute the transaction;
- replicate the data changes back to the General data store;
- make the time period in the General data store read-only again.

This is a heavy process thus data should be removed from the Current data store when it is unlikely to be updated. In our case the decision to keep in the Current data store the first 24 months should mean the heavy updates will happen in 0,5% of the update requests (as indicates the data analysis referenced above in section 3).

4.1. Hot-warm architecture model

We will follow the hot-warm architecture paradigm (Bennacer, n.d.) 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. **Figure 4** shows hot-warm ES cluster with 2 hot nodes (hot partition) and 3 warm nodes (warm partition). Cluster has as well 3 master eligible nodes (these are not data nodes, may be included either in the hot or the warm part of the cluster in dependence of the expected load of the master node). These nodes elect a single master node of the cluster. 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 (Bennacer, n.d.). Split brain may happen e.g. in a cluster with 2 master eligible nodes. If the connection between them is lost both nodes may think the other one is down. The node that currently is not a master will become one and that will result in a cluster with two masters that acts on a cluster independently. This may cause both inconsistent and lost data.

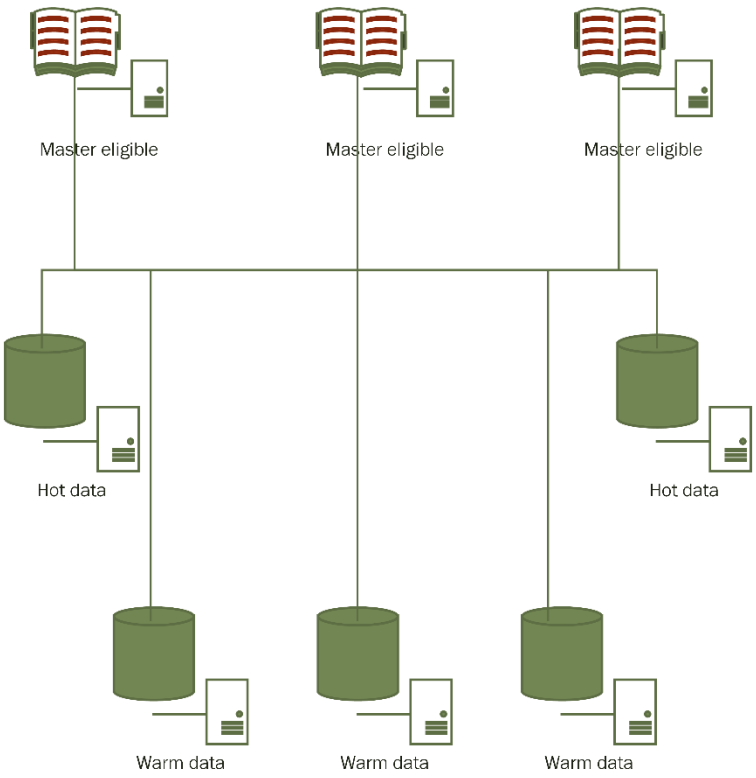

Figure 4. Hot-warm Elasticsearch cluster.

With the architecture in place we will discuss below its Pros (sections 4.2 to 4.6) and Cons (section 4.7).

4.2. No locking

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

ES 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 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 the possible lag before the updates in the index become available. This 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 mean all search requests waiting when 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).

4.3. Flexible

A number of parameters can be configured to tune the architecture:

- Number of hot and warm nodes;
- Node infrastructure (RAM volume, disk type and volume, etc.);
- ES index configuration (number of shards, number of replicas);
- Number of time periods kept in the Current data store;
- What time periods to keep online

4.4. Scale out

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

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

The search requests are distributed between replicas while new data is written to the primary shard and then copied to the replicas. A sample ES index with 3 shards and 2 replicas is shown on **Figure 5**. Here N1, N2 and N3 are nodes, S1, S2 and S3 are primary shards and Ri(j) is j-th replica of the shard Si.

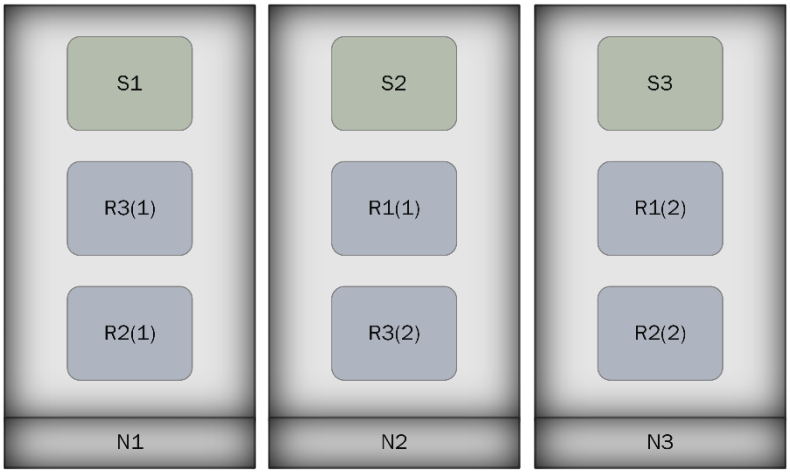

Figure 5. Sample Elasticsearch index.

4.5. Availability

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. The sample index above (Figure 5) is available even if two of the three cluster nodes go down.

4.6. Fast and powerful search

ES has proven to be one of the fastest and richest search engines out there capable of handling very large data and user request volumes. Our architecture provides a couple of means to profit from these ES values (**Table 2**).

Table 2. Means to improve search speed.

Parameter	Comments
Time phased indices	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 indices	ES provides for optimisation of read-only indices; 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 speed 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).

4.7. Downsides

Disadvantages of the proposed architecture are ones inherited from the polyglot persistence paradigm. Usage of two persistence technologies means increased development and maintenance costs (as experts for both technologies need to be involved), increased infrastructure costs (as hardware for both databases and backups has to be in place) and tougher testing (Dhandala, 2015). We suggest the Pros of our proposed architecture still outweigh the Cons.

5. Performance evaluation

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 (Rats, 2015).

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, attachment and task creation and modification. The results of the research mentioned above is used to assume frequencies of execution of the data request sequences by an ECM user. The list of data request sequences is expanded with the requests to the history data. Data request simulation model is tuned as well to the time based index structure of our architecture model allowing to explicitly direct a part of requests to an index (or several indexes).

ES hot-warm cluster of 6 nodes is created in MS Azure cloud and three different test data storages are generated for performance tests. Two of the data stores contain 145 million objects each, the third – 1.14 billion objects.

The performance tests are executed on a number of different configurations. Two different data models, yearly and monthly data indexes, several cluster node configurations, 1-3 replicas for primary shards, node hardware specifics, overall data volume, data request flow characteristics etc. are analysed for impact on cluster performance.

5.1. Results outlined

240 test runs on a data request flows generated were executed to measure different configurations of above described parameters. Generally the analysis of the results supports the opinion dominant in ES support forums and elsewhere that cluster and data model parameters depend on a particular use case. A number of interesting patterns have been observed though and are explained in the chapters below.

5.2. RAM and disk volume ratio

ES node loads index data from disk to RAM when started. The amount of data loaded depends on index mappings but anyway there may arise a situation when RAM is too

small to load all necessary data. That leads to ES node crash. Our tests show that for our index mappings 7GB RAM may handle up to 1.2TB of disk index volume. RAM has to be expanded to cope with larger disk volumes.

5.3. 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 6** shows the pattern for 1.14 billion object large data store of shared model, yearly indexes, cluster with 5 data nodes, 1 replica, 7GB RAM, HDD disks.

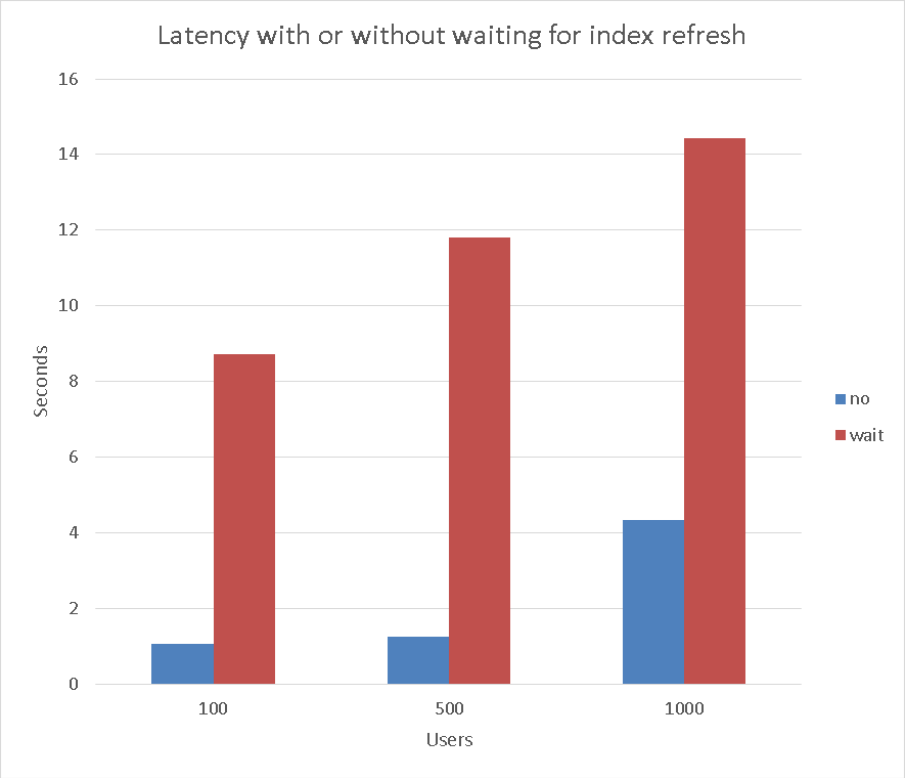

Figure 6. Latency with or without waiting for index refresh.

"Wait" and "no" in the figure designates two different methods of latency calculation. "Wait" means the latency includes waiting for the index refresh, "no" – the waiting for index refresh is not included in the latency calculation.

We have two options for cluster configuration when the index refresh times go up:

- to scale the cluster (introducing new nodes or expanding node RAM);

- assess our use cases if they demand immediate availability of new data for search; ES data requests may be configured so they return control without waiting for index refresh.

5.4. Impact of the history data requests

Our measurements show (**Figure 7**) that latency of the requests on current data grows for large history data and user request volumes. More users in diagram mean larger request volume. Data series 0 represent case with no history data requests, 2 – with data requests on 2 years of history data etc.

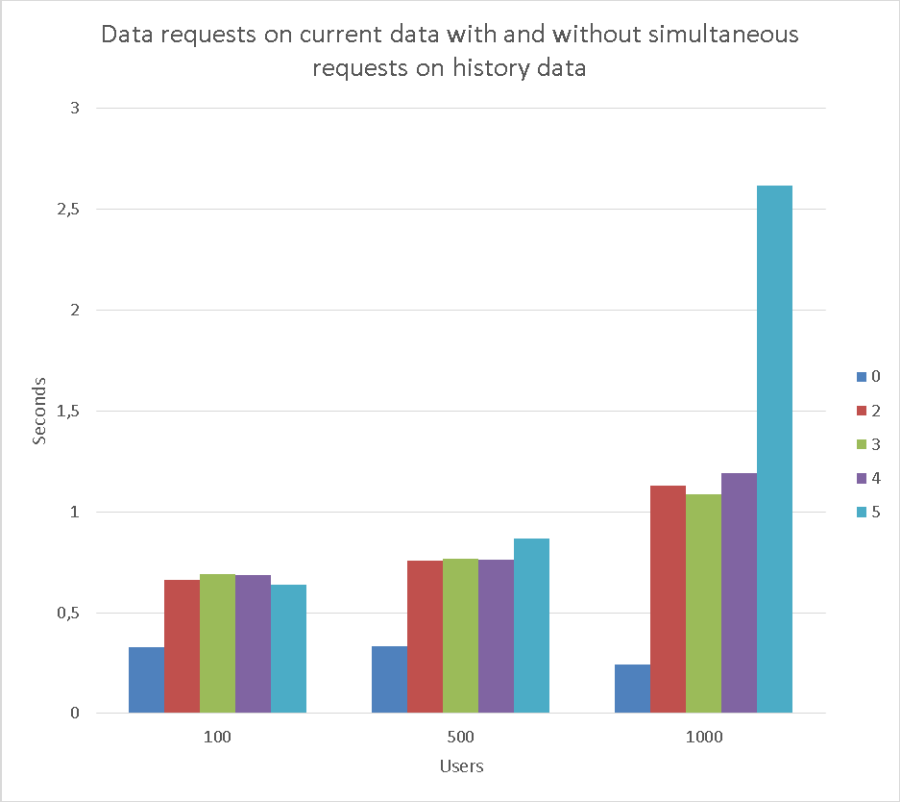

Figure 7. Impact of history data requests.

This might be caused by the ES design feature that every cluster node can role as a coordinating node. Coordinating node gets the user request, dispatches it to the nodes that may process it, then collects the results and sends it back to client. As random cluster node was used as a target for the data flow requests, hot nodes might be used as a coordinating nodes for some requests to history data and that could impact the latency on hot nodes.

6. Challenges

New technologies have their advantages and have their risks and challenges. The challenge all new technologies share is that they are new – people have to be trained to develop with, to maintain and to use the technology. The main challenge for proposed architecture arises from the same source its strength comes – the distributed computing.

6.1. Resilience

Elasticsearch is a clustered, scalable solution and is meant to provide for high performance and data availability. The cluster must survive network partitions and node crashes and uninterruptedly serve the user. ES employs sophisticated mechanisms to support this:

- moves index shards around the nodes to balance the node load or to recover from nodes becoming unavailable
- handles network partitions in cases when new index data has been propagated to some replicas but to others not
- master node manages the synchronisation process of critically important cluster state that knows everything about where on the cluster lives every index shard, what mappings every index has etc.

This allows ES cluster to function properly even when some of cluster nodes are down or unavailable. In a clustered solution it is hardly possible though to foresee and manage all possible disaster scenarios and hence to ensure nothing ever goes wrong. ES resilience is improved constantly but still situations are not ruled out where the data may be lost or, e.g., indexing requests duplicated. It does not help at all to know these are unlikely cases if data loss of any kind is unacceptable for your uses case (as we mentioned in section 2).

ES is mostly used as a secondary database that means the data may be restored from the primary in the (unlikely) case it is lost. Our architecture proposes to remove older data from the primary (Current data) store thus a part of the General data store becomes the only source of truth. This means we need a tools in place to survive possible data loss.

The solution we propose here is based on the feature of our architecture described in chapter 4 – the indexes of aged data are switched to read-only mode before the source data is removed from the Current data store. This means no data changes occur in these indexes and only thing that can go wrong is we can lose some index shard (when shard has been moved to another node but cluster state has not been updated accordingly).

Table 3 outlines the steps to take to address the resilience issue in question.

This way we may be sure we have snapshot before we remove data from the Current data store and before we switch index to read-only mode. If something goes wrong while this process is in progress it can be repeated.

Table 3. Handling of read-only node recovery.

Event	Actions to take
Index data is to be removed from the Current data store	• create index snapshot • remove index data from the Current data store • switch index to read-only mode
Cluster node recovery is reported	• check all read-only indexes for presence of all shards • if there are indexes with lost shards, recover the index from the snapshot

6.2. Performance measurement

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 must be used with care. Cloud solution here comes in handy as we can create the sibling cluster when needed and dispose it afterwards.

6.3. Maintenance

There are alternative ways to scale an ES cluster – adding nodes (hot or warm), adding replicas, expanding node RAM, switching to faster (SSD) disks, expanding disk volume etc. The main issue here is how to determine:

- that it is time to change something;
- what scaling option to select.

As we mention above in section 5.1 what is best depends on a use case. This means that if you are not ready to play with the production cluster (sure you are not) you need a sibling test cluster here again.

We suggest the following would be reasonable to maintain an ES production cluster:

- monitor and record the cluster performance;
- log user requests and create user request flows for performance tests;
- analyse the performance data (e.g. using ES beats and Kibana);
- create test clusters with different parameters changed (nodes added, RAM increased etc.) when substantial changes in data or user request volumes surfaced; measure performance for each case and select the best option.

7. Conclusions

We define a polyglot persistence architecture consisting of two data stores – the Current data store on SQL database and the General data store on Elasticsearch. Data objects are inserted/updated into the Current data store and searched/accessed in the General data store. The Current data store supports ACID transactions to ensure safe concurrent processing of data by multiple users while the General data store features fast execution of large volumes of search requests on large volumes of data. The General data store is split into time dependent (e.g. monthly) indexes and handled by a clustered ES solution on a hot-warm architecture ES cluster to allow for separate management of current and history data.

The architecture is evaluated measuring its performance on a MS Azure cloud hosted ES cluster. Impact of a number of parameters (node count and configuration, replica count, RAM volume, disk speed, total data amount, user request volume etc.) is evaluated and analysed. The suggestions are outlined on how to deal with the maintenance of the production cluster based on the proposed architecture.

The architecture is considered to be convenient for ECM solutions for large data and user request amounts. The measurements performed show that MS Azure hosted cluster configuration costing about €2000 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.

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).
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.
ES	Elasticsearch is a search engine and NoSQL document database
MS	Microsoft corporation
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.
SQL	Structured Query Language. The standard language for relational database management systems.
YCSB	Yahoo Cloud Serving Benchmark is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs.

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

Bennacer, S. (n.d.). "Hot-Warm" Architecture in Elasticsearch 5.x. Retrieved April 17, 2018, from https://www.elastic.co/blog/hot-warm-architecture-in-elasticsearch-5-x

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

Dhandala, N. (2015). The Pros and Cons of Polyglot Persistence. Retrieved June 5, 2018, from https://opensourceforu.com/2015/08/the-pros-and-cons-of-polyglot-persistence/

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

Elasticsearch as a primary database - Elasticsearch - Discuss the Elastic Stack. (2018). Retrieved May 30, 2018, from https://discuss.elastic.co/t/elasticsearch-as-a-primary-database/85733/16

Elasticsearch Resiliency Status | Elastic. (n.d.). Retrieved January 23, 2017, from https://www.elastic.co/guide/en/elasticsearch/resiliency/current/index.html

Foote, K. D. (2017). Utilizing Multiple Data Stores and Data Models: Is Polyglot Persistence Worth It? - DATAVERSITY. Retrieved May 22, 2018, from http://www.dataversity.net/utilizing-multiple-data-stores-data-models-polyglot-persistence-worth/

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

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

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

Sadalage, P., Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Vasa (1st Editio). Addison-Wesley Professional. http://doi.org/0321826620

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

Vassallo, J. (2016). Forrester : Marketing : Forrester Forecasts Big Data Tech Market Will Grow ~3x Faster Than Overall Tech Market. Retrieved April 26, 2018, from https://www.forrester.com/Forrester+Forecasts+Big+Data+Tech+Market+Will+Grow+3x+Faster+Than+Overall+Tech+Market/-/E-PRE9484

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

Received June 8, 2018, revised September 27, 2018, accepted September 27, 2018