Information Retrieval and Analysis for a Modern Organization

Artyom Topchyan <a.topchyan@reply.de>
Yerevan State University,
Alek Manukyan 1, Yerevan, 0025, Armenia

Abstract. With the growing volume and demand for data a major concern for an Organization is to discover what data there actually is, what it contains and how it is being used and by who. The amount of data and the disparate systems used to handle this data increase in their number and complexity every year and unifying these systems becomes more and more complex. In this work we describe an Intelligent search engine system, specifically designed to tackle the problem of information retrieval and sharing in a large multifaceted organization, that already has many systems in place for each Department, which is an integral part of a joint Operational Data Platform(ODP) for data exploration and processing.

Keywords: data-driven projects; information retrieval; streaming processing; mesos; kafka

DOI: 10.15514/ISPRAS-2016-28(4)-1

For citation: Topchyan A.R. Information Retrieval and Analysis for a Modern Organization. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 4, 2016, pp. 7-28. DOI: 10.15514/ISPRAS-2016-28(4)-1

1. Introduction

With the growing volume and demand for data a major concern for an Organization is to discover what data there actually is, what it contains and how it is being used and by whom. The amount of data and the disparate systems used to handle this data increase in their number and complexity every year. This trend is driven by business and technical demand, which especially stems from the need for more Data-Driven projects [1][2][3]. Data-Driven projects aim at increasing the quality, speed and/or quantity of information gained from Data collected by the Organization. But it is very challenging to move ahead with such projects without access to a defined model to handle data exploration and processing. This leads to most of the project time being spent on actually finding out information about the existing data, finding people involved, data owners and where the data lies, as opposed to actual analysis. As described in our previous work this can be quite costly time and resource and each stage of a Data-Driven project is impacted by this. There is currently no single

accepted approach for tackling such problems, so we described and implemented a joint Operational Data Platform(ODP) for data exploration and processing, which aims to be an end-to-end platform for solving the issue of managing large amounts of data and information about this data by means of scalable automation and information extraction [1]. This platform has been successfully implemented and is actively used by large organizations to implement Data-Driven projects. A high level overview of the entire solution as presented in our previous work is outlined in Fig.1. One of the main components of the platform was the Information Marketplace, which is an intelligent search engine system, specifically designed to tackle the problem of information retrieval and sharing in a large multifaceted organization, that already has many systems in place for each Department. As outlined above, we believe this to be the key challenge when exploring possible use cases for Data Scientists. In this work we outline, the problem definition and technical implementation, in more detail. We will first outline why we believe such a solution is required and how it fits into an existing, ever changing landscape of an organization. Then we will outline in detail what data is used and what and how information is extracted from it in order to build a search index over all project information and data, that an organization has and continuously develops in the future. We will specifically concentrate on the architectural and algorithmic scalability challenges of extracting information from large and varied datasets using existing methods.

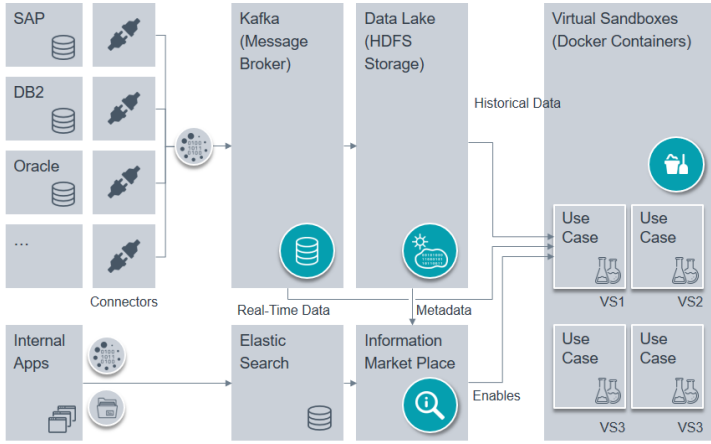

Fig. 1. Operational Data Platform

2. Information Retrieval and Sharing

One of the biggest challenges faced by an Organization when exploring possible use cases for Data Scientists is in experience knowledge sharing and transfer. Large Organizations have a large number of Departments, which vary widely based on their size, project they take on and the way these projects are completed and documented.

This leads a large variety of data sources, column names and documentation being created on the same subject by a large number of stakeholders from different Departments some of whom might not be part of the Organization anymore. This leads to challenges for Data Scientists and the IT Department, which have to identify the relevant information and people or documents describing the data, especially when the project involves more than one data source. The most important consideration here is time spent on actually finding out if data is available and similar issues as opposed to more productive data analysis.

To bridge this issue, there are large undertakings for an organization wide change management process, which pushes for standardization. On a technical level this changes translate to the centralization and standardization of project related documentation as well as rigid data views in a central database. Due to the complexity of the data and the Organization itself an Enterprise Data Warehouses or a Wiki-System cannot directly solve this issue, while satisfying all the requirements on structure and intelligence. This leads to the creation of Organization wide specialty tools, data/knowledge repositories and integration layers providing each Department with access and management capabilities, in order to adapt to the specific requirements of the Organization.

Essentially what this entails are a full organizational restructure to create solution for data and information management. In reality, this is a vast and complex process and can cost a large amount of money and resources from the side of the Organization and in some cases might decrease productivity. Each individual Department has an approach of managing projects and in most cases such a monolithic system allows for less flexibility for individual Departments. This may lead to decreased productivity. It is often unrealistic to expect full and informed cooperation from each Department and its staff in order for each project to be documented, every data column described, every project contributor to be listed and all the interconnections between documents and data of multiple Departments being identified and documented. This is further complicated by the fact, that there is always more data and information each year, so all of the created documentation should be retroactively update periodically. The learning period for a complete change and standardization of such processes and the time required to update all of this information, can bring an entire Department to a halt for an extended period of time.

There are also a large number of technical considerations, such as file format support, performance of the search and indexing. Data Scientists and the Business Department would like to see changes to any information as fast as possible.

To summarize, lets pose a list of requirements we accumulated based on our experience working with teams of Data Scientists at large organizations:

1. **Index decentralized documentation repositories.** Each Department should be able to retain their knowledge repositories with little to no functional changes. Data should be acquired from these repositories with as little effort as possible from the Departments side.

2. **Support a large number of formats.** Each Department should be able to use any binary file format for their project documentation.
3. **Document Metadata has to be preserved.** Such properties as authors, auditors, names, comments should all be extracted and preserved and be uniquely identifiable.
4. **Low Latency Indexing.** Changes to or new documents should be reflected as fast as possible.
5. **Content Indexing.** As a minimum all textual content has to be searchable, there has to be a possibility to later extend this to non-textual content, such as audio and image files.
6. **Author extraction.** Each document must be tagged automatically with the authors of that document. This can be extracted depending on the metadata or by analysis of the document or related documents.
7. **Document summarization.** A short summary of the Document would be very beneficial when exploring a large collection. Giving context knowledge about document can often be use full to understand the content at a glance. This could be implemented by extracting important sentences or keywords.
8. **Context search.** Each document describes a specific context or project in relation to the organization or the Department. This representation of the document if very valuable specifically for finding similar projects, but which are not specifically referring to specific data sources, projects or Departments.
9. **Cluster by Department context.** It is often useful to look at groups of documents referring to specific contexts as opposed to exploring them one by one. This is specifically interesting to see if documents of one Department refer to a context usually associated with another Department.
10. **Data source search.** It should be possible to search for specific data sources referred to in the documents. Meaning every document should point to specific data sources it mentions. Used with the document clustering approach, documents that have no mention of any document can also be tagged.
11. **Data source summarization.** It is very important to present an outline of the data, that is actually available in connection to a Data source a document refers to.
12. **Flexible faceting.** All of this extracted information should be made available as facet views, available when using the tool to explore the organizational information.

13. **Decoupling of functionality.** As we described above, the landscape of the organizational data and systems is always changing and there should be no hard dependencies between specific Departments or types of analysis. If a Department stops producing documentation in a specific format or at all or author information should no longer be stored, the systems should not require a substantial rewrite to remove these functions.

14. **Flexibility for extensions.** Similar to the previous requirement the system should be easily extensible if further analysis is required, such as for example support for extracting information from image data. This should also not invalidate the previous requirement and require a substantial modification to the system. New attributes or indices should be handled transparently.

15. **Performance.** This tool will be used by Data Scientist and the Business Department in their day to day work, this means it should allow for a responsive experience and support many hundreds of concurrent users.

16. **Scalability.** The entire solution entails fairly complex computation as there are potentially tens of gigabytes of documentation (from many Departments) and data meta information (data schemas) as well as terabytes of data that has to be analysed. This means the solution should be scalable in order to provide the similar levels of performance independent of new documentation or data source.

17. **Fault tolerance.** This is more of an exploratory tool and as such full end-to-end correctness is not necessarily required. By correct we mean, that no documents are processed more than once and all results are always consistent throughout the solution. Nevertheless, we need to provide a certain level of fault tolerance thought the system so little to no supervision is required to ring the system to a consistent state. In this case we are looking more at the system being eventually consistent [12].

This is not an exhaustive list, but it outlines the base requirements a system for information retrieval and sharing should fulfill. It is list of complex functional requirements which we need to map to a technical problem definition and implementation.

In the next section we will explore this definition and the architecture as well as the algorithmic implementation of this tool, which we called the Organizational Information Marketplace.

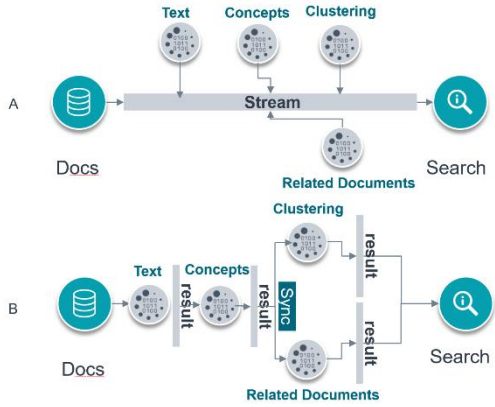

Fig. 2. (a) Data-flow requirements in the context of stream processing. (b) Data-flow requirements in the context of batch processing.

3. Architecture

There are significant architectural and algorithmic considerations when mapping the requirement set outlined in the previous chapter to a technical implementation. To achieve all the requirements for performance this has to be distributed application running on multiple machines, which introduce a large set of benefits and problems.

First let's consider the architectural side of the problem, before outlining how the required information is extracted and modelled. To outline the architecture, we first need to define, what input we have and what required output has to be produced.

Input. Our input is a collections of documents for each Department/data source the organization has. These collections are unbounded and may increase and decrease as well as change over time and so can the data sources. Currently a single collection of documents can be larger than ten gigabytes with documents containing hundreds of words. Additionally, such a system should be ready to process raw organizational data as well. This would be important to analyze the actual data content as opposed to definitions. This can become extremely large with terabytes of data coming every month.

Output. As the output we require the document with additional extracted or calculated metadata information, such as the summary of the document, the semantic/contextual representation of the document, similar documents and others. This information should be exposed through an interface, that supports a rich set of text based queries. As the entire content of the document has to be indexed with all the additional extracted information, we are looking at indexes tens of gigabytes in size per Departments.

Based on the requirements we need to find a way to get from the input to the output as fast as possible, whilst applying a non-fixed number of potentially process

intensive transformation functions in sequence on an unbounded collection of documents. Additionally, the latency of processing, should not be affected by the number of documents in a collection at any point in time, meaning there should be a defined way of scaling each step independent of the others. It is an accepted approach in the Industry to frame this problem in the context of stream processing [5][4] as opposed to a periodic batch process. In stream processing each collection of documents can be represented as a separate stream of documents, which changes constantly over time. Each new element of the stream is processed one by one, out-of-order in parallel. The output of each transformation is represented as an another stream of datum and can be directly consumed by another transformation. This is quite different from a periodic batch approach, where each collection is considered static at the point in time and all transformations have to be applied to the entirety of collection. This is disadvantageous for our case as certain transformations can depend on others, which means a transformation has to be applied on all documents before proceeding to the next step. Because This means we need to either join the transformations together into more monolithic blocks or create large intermediary collections of transformed documents and orchestrate the order of the batch jobs. In our experience, this is highly impractical, as this entails either very large, not flexible transformation steps or a complex scheduling problem. Such problems are very tricky to tune to scale correctly. Not to mention the fact, that this process be definition is high latency, but this of course depends on the number of documents and the complexity of transformations at any point in time. The difference between the two approaches is highlighted in Fig.2. The main issue comes from the necessity to sync between transformations. A transform has to be notified that a previous step has been completed. This introduces complexity and latency. In our approach we settled on decreasing the granularity of data slices and define a stream of data. This in our opinion allows for flexibility of adding and removing steps at essentially any point. As opposed to only passing simple events around [7] we model the data as an unbounded stream, so we can process these elements on-by-one or in larger batches as well without intermediate storage. This greatly simplifies the amount of state that has to be tracked in the entire solution and as a side effect, in our experience, leads to more concise, performant and testable services. But we still need to define an approach for scaling this type of solution and more importantly ensuring a certain amount of automatic fault-tolerance to make the solution as flexibly and low-maintenance as possible.

3.1 Components for Fault Tolerance and Scalability

ThereisasignificantconnectionbetweenFault-ToleranceandScalability. Scalability is usually most directly solved by trying to parallelize the costly process, most often, by launching more instances of the same service and balancing the work between each instance. But according to the CAP theorem, which states that it is impossible for a distributed computer system to simultaneously provide Consistency (all nodes see the same data at the same time), Availability (every request receives a response about

whether it succeeded or failed) and Partition tolerance (the system continues to operate despite arbitrary partitioning due to network failures. To this end the most important architectural block of a streaming processing systems is the representation of the "stream". As we described we need a proven way of storing unbounded amounts of information, which has to be tolerant to failures and scale to a large number of events and services.

3.1.1 Stream component

A widely accepted approach to this is using a distributed message broker, such as Apache Kafka [25], which is a high throughput, distributed and durable messaging system. Each stream can be represented as a topic in Kafka. A Topic is a partitioned log of events. Each partition is an ordered, immutable sequence of messages that is continually appended to—a commit log. The messages in the partitions are each assigned a sequential id number called the offset that uniquely identifies each message within the partition [25]. This is a flexible representation as it allows us to create many topics per stage, which can be consumed by processors in a stage(consumers). Each Kafka consumers is part of a consumer group and Kafka itself is balancing the partitions in the topic over the consumer processes in each group. This means that it is simple to achieve processing scalability just by launching more consumer processes. As data is automatically distributed and consumption is balanced, Availability is straightforwardly addressed by dynamically launching more instances of the service and relying on a balancing mechanism. On the other hand, as described in [9][8] streaming systems are most sensitive to Partitioning and Consistency problems. The generally accepted approach to try and solve this [10] [29] is essentially based on offset tracking. Offset tracking is tracking how far along in the stream the processing service is. If each instance of the service has to make sure it processed a single or batch of event before requesting new events. It would be quite expensive to commit such offsets for every event, so offsets are usually committed is small batches. This is supported as a core functionality within Kafka itself, which offers approaches to store offsets in a separate topic as well as allows for automatic batching of events. Using Kafka, we introduce a scalable and fault tolerant representation of a stream, both on the storage and processing level.

3.1.2. Service Scheduling component

With the described approach each service is only responsible for deciding how many events to process and from which offsets to start and work balancing, storage and transfer are handled by the stream representation. Kafka has no control over the consumer processes as well has no idea how costly each operation or what state(offsets) have to be tracked. This means it is simple to scale processes just by launching more processes, but making sure all events were processed correctly during failures, as well as quickly is up to the consumer itself. To this end we require a defined way for dynamically scaling process as well as restarting and retrying in case

of consumer outage. To solve this issue, we can exploit the fact, that with the proposed way of handling data storage and state, each service is essentially immutable. Which essentially means a crashed service does not need to be restarted or recovered, but we can just start a new copy, which will catch up to the state the previous service was automatically. Coupled with the fact, that we have a large number of machines at our disposable the problem of service fault tolerance and scaling becomes a process scheduling problem. Technically we have a set amount of resources for each stage, each stage takes a certain amount of time to compute and we are trying to reach a certain amount of latency. Using this information, we can approximate how many processes have to run and where in order to achieve the required latency. We require a component, that can decide whether to start, stop or restart processes. This is an accepted approach to handle scaling largescale distributed application and has been successfully implemented in the industry [11] [27] by means of a global resource manager. Such a resource manager is essentially a higher level version of an Operating Systems Kernel, but running on many machines. Applications decide based on their processing time and requirement, what resources they require. The resource manager tries to accommodate this. If the service crashes or the computing node fails, it can be transparently restarted somewhere else assuming the application can transparently handle something like this. Which our proposed approach can. Here we use Apache Mesos, a commonly accepted High-Availability implementation of such a system. Such a system has also proven to be very flexible to new services or any adaption [13]. The system just needs to know what resource and what processes to run.

3.2 View component and Architecture Implementation

The last thing missing from our architecture is the "view “layer. This is where the output of the processing stages is stored in its final form, as well as as any extra data. Based on the outlined requirements we need a flexible and scalable storage systems, that supports a large scale of analytical queries. Considering the fact, that most of our data are documents. It is more efficient to store the output in a Search Engine. In this work we use Elasticsearch [26] due to its proven scalability and performance, specifically in the context of stream processing. Based on this we can define our general purpose architecture for building a scalable and Fault-Tolerant stage based Event Driven application for the purpose of extracting information, context from an organizations documentation and allowing this to be efficiently queried. A high level overview of the architecture is presented in Fig. 3. It should be noted, that in this work we concentrate on the architectural aspect of scaling and fault-tolerance and do not define a novel approach, but more an architectural framework using the contexts from these projects in order to efficiently and accurately solve the problem for this specific use case.

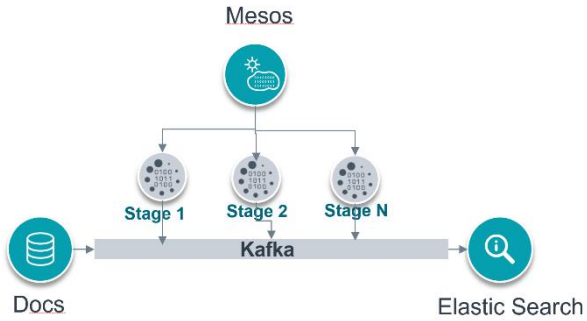

Fig. 3. Information Marketplace Architecture

4. Processing Stages and Implementation

Based on the described architecture we need to map the requirements described in the previous chapters to a set of stages of transformation a single document has to go through and queryable views, which are the end result of these transformations. First lets define the views, which are needed to satisfy the requirements:

- Document Full-Text Search View
- Search View based on Author, Time, File-Format, Department, Data source
- Search View based on extracted keywords, contexts, summaries
- Related document View, based on contexts and keywords

These view are essentially queries against the Elasticsearch database, where the data will be stored and so we will not outline them in much detail outside of the stage definitions. What is required is a full representation of the document, data sources and Departments. Based on this we can define the stages of transformation:

1. Document load from source
2. Document original format to plain text conversion with metadata extraction
3. Map document to specific organizational data source
4. Extract document summary
5. Extract semantic representation of the document(contexts)
6. Find closely related/similar documents
7. Find Departments a data source might refer to
8. Index document

We will now outline in more detail what each stage does and how it is implemented as well as scalability and fault tolerance considerations for each stage.

4.1 Load document

- Input — Directories to watch for files
- Output — Stream of document creation, update, delete

Documents may reside on a filesystem in every Department. We need to detect all the files already there and detect the creation of new files to fulfill the low-latency requirement. We implemented a distributed directory watcher, which watches for filesystem event and emits these events into a Kafka stream. Each watcher keeps a reverse index of the filesystem it is watching. This is required in order to track what files we have already processed, as no duplicates should be processes if possible. The filepath and the last modified date are used as the offset key in the reverse index. The offsets are only committed if the file was detected, and has not been modified in the last few seconds. This is done in order to avoid reading files, which are still being written to. With this approach we ensure fault tolerance and exactly once processing of all files. Because this index is stored as a collection of offsets in a Kafka topic no duplicates or missing documents will be processed during service or full node failures. We implemented these watcher using the Kafka Connect Framework [29], which is a framework tightly coupled with Kafka and offers some utility methods to more simply write data to Kafka. Each watcher writes an event to Kafka containing the source of the event, the nature of the event and the filepath. This stage is then defined as a collection of such watchers accessing multiple organizational sources distributed throughout the organization.

4.2 Document conversion

- Input — Stream of document
- Output — Stream of document metadata and text representations

We need to extract information from a large variety of document formats. As most of the document ion is written we need to convert it to a simple text representation as opposed to a native binary format such as Microsoft Word or PDF. For parsing a large variety of format we use Apache Tika [28]. Tika also allows us to extract a full set of file metadata from the file, such as authors, modified dates, owners and etc. This stage produces three different text representations:

1. HTML bases representation, preserving the formatting, useful for display
2. Plain Text representation
3. Filtered text, based on language stop words are filtered, characters are normalized

As document arrive in a highly asynchronous manner from many data sources in a single stream we can implement a distributed stream processing application running on top of Mesos to convert this representation to Text. Essentially each streaming consumer receives the file, extracts the text and metadata and materializes the result to another stream. Processes can be pre-allocated based on the number of documents

committed and average processing time, this is implemented keeping the main ideas of [14].

4.3. Data source mapping

- Input - Stream of documents
- Output - Stream of documents with Organization data sources labelled

As we outlined in previous sections understanding which data a document is referring can be very beneficial to Data Scientists, when trying to build a project. This info can both be used to find out what the data is about or the reverse, which data matches a certain topic. What we have as input is a lot of documents with no extra information apart from their content and origin. We need to map each document received to one or more data sources. If we had some documentation for each data-source of the organization we could rephrase this issue as a context matching problem, this could be solved by one of our other stages of analysis. But it is very rarely the case, that an Organization has any significant amount of information on every data source they have, quite often the information is there, but has been lost in the thousands of documents scattered across Departmental repositories. This is the aspect of the problem we are trying to solve in this stage. As it stands assuming we have the technical definition of each data source, schemas, column definitions and etc., we need to find instances of documents that contain references on these attributes. A straightforward approach would be to scan through each document and count how many instances of all permutations of all columns are contained inside the document. But if we consider the complexity of such calculations for a typical organization, it becomes a very suboptimal approach. For example, this stage has to function with low latency processing at an organization with dozens of different database systems, which have thousands of tables in total and each table containing potentially hundreds of columns. Such an organization has tens of thousands of pieces of documentation, with a hundreds of documents being modified or created every day. Such documents usually contain hundreds of words on average. This is an obviously suboptimal calculation, which is not practical to compute with low latency. For example, let's say we have 3000 tables with 100 column search and 10000 documents with 500 words each. This would mean for one document we would need to do $500 \times 300000 = 150000000$ not simple comparisons. Another downside of such an approach is, that we would need to keep all data related information in memory as a collection, which would lead to large resource requirements. To this we adopt a method similar to the approach described in [21], which rephrases this issue as a comparison of sets of data. Indeed, we can efficiently represent a document and each data source specification as sets, and then our problem would be to find a set most similar to each incoming document. Of course this would mean, that we are taking the overlap of a document and a full set of data columns, which would never truly fully overlap, but as described in [21] this would give us a good approximate match and we could then analyse which of the matched data source is truly mentioned in the document. For this we build an efficient representation of each data source specification using the Minhash (min-wise

independent permutations locality sensitive hashing scheme) algorithm [20]. 4.3.1. Minhash

The Jaccard similarity coefficient is a commonly used indicator of the similarity between two sets. For sets A and B, the Jaccard similarity is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cup B|}$$

We can phrase this as the ratio of the size of the intersection of A and B to the size of their union. J lies in 0 <= J <= 1. More specifically, if J = 1 the sets are identical, if J = 0 there are no shared members otherwise the value can be interpreted as “the probability that a random element from the union of two sets is also in their intersection” or “the probability that a randomly chosen element chosen from one of the sets is also in the other set.”. This is widely accepted measure of similarity, but it is still quite expensive to compute [21] especially for our case. To this end [20] describes an algorithm to estimate the Jaccard similarity (resemblance) between sets of arbitrary sizes in linear time using a small and fixed memory space. It is also quite suitable to use in a stream processing context due to the low computation time and compact representation. First lets define a matrix representation of the comparison of multiple sets. Let’s define matrix M where m_{ij} is 1 if element i is in set S_j and 0 otherwise. Let’s denote N as the number of all unique elements in each set. Then for N let’s take K random hash functions $\{h_1, h_2, \dots h_k\}$, which map each element to a random unique ID from[N]. Then for k counters $c_1, c_2, \dots, c_k$ we can define the Minhash for the set S as: Then set mj(S) = cj and we can define the Jaccard distance estimate as:

$$JS_k(S_i, S_{\hat{i}}) = \frac{1}{k} \sum_{i=1}^{k=1} I(m_j(S_i) = m_j(S_{\hat{i}}))$$

, where I(σ) = 1 if σ = 1.

This also gives us a compact representation, which we can calculate and store for each data source specification, meaning we only need to calculate the Minhash for each incoming document. But this is still computationally expensive as we still need do the computation at least O(N_{tables}), meaning the query cost increases linearly with respect to the number of data sources. A popular alternative is to use Locality Sensitive Hashing (LSH) index.

4.3.1 Minhash-LSH

One approach to implement LSH is to "hash" each element many times, so that similar items are more likely to be hashed to the same bucket. We can then just look at the pairs, which ended up in the same bucket. The main idea as described in [21] is that most of the dissimilar pairs will never hash to the same bucket, and therefore will never be checked, and the number of false positives is low.

Algorithm 1 Min hash on Set S

```
for i = 1 to N do
    if S(i) = 1 then
        for j = 1 to k do
            if h_j(i) < c_j then
                c_j ← h_j(i)
            end if
        end for
    end if
end for
```

For a Minhash representation computed as described above, an effective way to choose the hash functions is to take the matrix of the representation and partition it in to b partitions with r rows each. Then we use a hash function for each partitions and calculate the mapping to many buckets. Each partitions can use the same has function and keep a separate bucket for each partition. This means columns with the same vector in different bands will not hash to the same bucket. Then we can check each band for matches in each bucket.

Based on the describes algorithms we create a Minhash representation for all data source specifications and build and LSH index over these. This allows us to very efficiently query for the most similar data sources for an incoming document in a stream. We take the 3 top matches and find, which one actually is most referenced inside the document. This is attached to the document as extra metadata and is written to another stream. Due to the implementation this can be easily scaled by launching more instances as the Minhash representation is fairly compact.

4.4 Document summary

- Input — Stream of documents
- Output — Stream of documents with added summary

This stages receives a stream of documents as input and outputs the same document with an additional summarization field attached to the metadata. A text summary in this context is a set of most important phrases/sentences of the document. Such kind of method are very sensitive to stop words, words that do not impact the semantic meaning of the text, such as "and" or " and etc. So this stage relies on the third form of the text representation, which is already filtered and cleaned. As we only have unstructured documents and most organization do not tag or write abstracts for each document, we need to rely on an unsupervised method for text summarization [19]. We summarize the given text, by extracting one or more important sentences from the text. This summarizer is based on the "TextRank" algorithm [18], which was chosen when considering its performance and a simpler implementation for key-sentence extraction vs keywords.

4.4.1 TextRank

In the TextRank algorithm [18], a text is represented by a graph. Each vertex corresponds to a word type or sentence. Vertices v_i and vj are connected with a weight w_{ij}, which corresponds to the number of times the corresponding word types co-occur within a defined window in the text. The main objective of TextRank is to compute the most important vertices of the text. If we denote the neighbors of vertex v_i as Adj(v_i), then the score S(v_i) is computed iteratively using the following formulae:

$$S(v_i) = (1-d) + d \times \sum_{v_j \in Adj(v_i)} \frac{w_{ji}}{\sum_{v_k \in Adj(v_j)} S(v_j)}$$

Vertexes with many neighbors that have high scores will be ranked higher. We are of course not interested in keyword extraction, but sentences based summarization. To this end as shown in [18] it is we can adapt the same algorithm to sentences. Each vertex will represent a sentence and, as "co-occurrence" is not a logical relation between sentences, defining another similarity measures. A similarity of two sentences is defined as a function of how much they overlap. The overlap of two sentences can be determined simply as the number of common tokens $w_1^i \ldots w_N^i$ between the lexical representations of the two sentences S_i and S_j. In [18] it is defined as:

$$Sim(S_i, S_j) = \frac{|\{w_k | w_k \in S_i, w_k \in S_j|\}}{log(|S_i|) + log(|S_j|)}$$

By applying the same ranking algorithm 0.3 and sorting the sentences in reversed order of their score we can find the most important sentences, that represent the document. This algorithm was implemented and applied in a stream of document to produce the output of the stage. The algorithm performance only depends on the length of the documents, and based on our average length of documents, the processing speed satisfies out latency requirements.

4.5 Context extraction

- Input — Stream of documents with extracted information
- Output — Stream of documents with context
- Output — Context Model for each Department

We require a representation of each document in the context of the Department it is generated by, as well as other Departments. To do this we need a model, that captures the concepts most used and referred to by projects in the Department. This would give us a valuable contextual representation of key concepts. This could be achieved by extracting keywords or popular words similar to the methods described in the Document Summarizing stage, but what this kind of approach lacks is the document level representation. We want to consider the context both on the level of separate

concepts as well as groups of these concepts(documents). This would give us a more complete view of the context of the Department as well as allow us to extract, such a contextual representation for any document. Meaning we would see how a document relates to work usually carried out in the Departments. We also want to analyse the extracted context to find similar concepts. Another requirement we have is that any model used should be flexible and require little maintenance to be updated in the future. For this Probabilistic Topic Models are a widely accepted choice [17]. Using these models, we can infer the semantic structure of a collection of document using Bayesian analysis. These models are fairly popular and have been used to solve a wide range of similar problems [23] [24]. In this work we specifically adopted the Latent Dirichlet allocation (LDA) model.

4.5.1 Latent Dirichlet allocation

One of the most popular models is Latent Dirichlet allocation (LDA) [17]. LDA is a model for extracting underlying topical information from unstructured data. LDA models each document as being represent by multiple "topics". The topics are represented as a collection of words, or more specifically a probability distribution over a fixed vocabulary of terms. In LDA documents in a corpus are assumed to be generated from a set of K topics. Each topic k is represented by a Dirichlet distribution over the vocabulary:

$$\beta_k \sim Dir$$

Each document d is also repented as a Dirichlet θd.Eachword w indocument d is then assumed to be generated by drawing a topic index z_{dw} from a multinomial distribution Mult(θ(d)), then choose a word wdn for that topic from Mult ($\varphi_{z_{dw}}$). By assuming this generative process, we can go backwards and estimate these matrices from a collection of documents. In this case a collection per Department, which will generate a model per Department. There are a number of way to train such a model [17] [16]. In this work as we require low latency inference as well as flexibility to do automatic model updates, we adopt the Online Variational Bayes algorithm [16]. This method also has the quality of being static in memory as the entire stream of documents does not have to be loaded fully when required. Mini-Batches of documents are processed to infer these matrices. As with most vibrational algorithms, the processing consists of two steps:

- The E-Step: Given the minibatch of documents, updates to the corresponding rows of the topic/word matrix are computed.
- The M-Step: The updates from the E-Step are blended with the current topic/word matrix resulting in the updated topic/word matrix.

Each document received is added to the model and after that the topics are extracted, this allows us to continuously update the model as well as extract topics from documents. It is more effective to update the model with minibatches [16]. A mini-batch is a small batch of documents. In our case, this is usually set to 100. This is also

efficient for throughput as well as simplified offset tracking. Offsets are committed only after a mini-batch of documents has been successfully processed. After each iteration the model is save to disk and only then the offsets are committed. This means we ensure the update model always contains all the documents and is up to date. The topics are attached to the document with their distribution, this is necessary later on for ranking text matches. A model is produced per Department in order to captured the unique context of each. This means the stage contains at least one instance of each model, but the output of the stage is still a single stream. Related documents as we are extracting the context of each document via the LDA model and attaching this information with the specific weights to the document that is indexed in the Search Engine, it is fairly trivial to find contextually related documents, just by including this field and its weights when executing text searches

4.6 Related documents

As we are extracting the context of each document via the LDA model and attaching this information with the specific weights to the document that is indexed in the Search Engine, it is fairly trivial to find contextually related documents, just by including this field and its weights when executing text searches.

4.7 Departmental context

- Input — Stream of Documents
- Output — Stream of documents with attached "Department" label

If we take the described LDA Topic Model for each Department as a contextual representation of the concepts used, we can use this as the representation of what the Department does in the context of the organization. We essentially have a collection of sets of topics per Department and we want to find which other Departments this document relates to. We can exploit the Minhash based set matching approach we used in the previous stages and apply it to the problem of matching a document to an organization. We can define an LSH index over each set of topics for each organization. Then for any incoming document we need to find the most similar set of topics from each organization a document relates to. Based on this we can extract a very rough estimation of "related Departments".

As we are using the same efficient representation as with the data source mapping stage we can efficiently calculate this with low resource requirements.

4.8 Document indexing

- Input — Stream of documents with extracted information
- Output — Elasticsearch indexes

This is the stage that actually indexes the data into a search engine, in this case Elasticsearch is used to create a reverse token index of the documents.

We implemented these watcher using the Kafka Connect Framework [29]. Each index writes an event from Kafka into Elasticsearch. This stage is then defined as a collection of such indexers accessing multiple sources throughout the organization. The data is keyed with the filenames, event timestamp, and a unique offset is generated via a has function to ensure, that no duplicate data is written into Elasticsearch even during node faults. We can rely on Elasticsearch's idempotent write semantics to ensure exactly once delivery. By setting ids in Elasticsearch documents, the connector can ensure exactly once delivery by simply overwriting the data and not creating new records. You can restart and kill the processes and they will pick up where they left off, copying only new data. As opposed to the data loading stage we can read from and write to Elasticsearch in parallel and so we can launch more services to scale this stage based on amount of input.

5. Conclusion

We proposed a flexible way for discovering data and interconnections of the data, based on metadata, functional descriptions and Documentation, in an automated and intelligent way, while not requiring a full Departmental restructure. We outlined the set of requirements desired by many Organizations in the industry. We described a scalable, fault-tolerant and flexible Architecture as well as technical and algorithmic implementation, that satisfies all these requirements. We believe the approach, proposed solution and its architecture provide a solid basis for implementing similar information retrieval solutions at large Organizations, that are starting to explore Data-Driven projects.

References

[1]. Topchyan A.R. Enabling Data Driven Projects for a Modern Enterprise. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, 2016, pp. 209-230. DOI: 10.15514/ISPRAS-2016-28(3)-13

[2]. Rahman, Nayem, and Fahad Aldhaban. "Assessing the effectiveness of big data initiatives."2015 Portland International Conference on Management of Engineering and Technology (PICMET). IEEE, 2015.

[3]. Davenport, Thomas H., and Jill Dych´e. "Big data in big companies."International Institute for Analytics (2013).

[4]. Dunning, Ted, and Ellen Friedman. Streaming Architecture: New Designs Using Apache Kafka and Mapr Streams. O'Reilly Media .2016.

[5]. Marz, Nathan, and James Warren. Big Data: Principles and best practices of scalable real-time data systems. Manning Publications Co, 2015

[6]. Michael Hausenblas and Nathan Bijnens. Lambda Architecture. http://lambda-architecture.net, 2015.

[7]. K. Mani Chandy. vent-Driven Applications: Costs, Benefits and Design Approaches, California Institute of Technology, 2006.

[8]. Akidau, Tyler, et al. "MillWheel: fault-tolerant stream processing at internet scale."Proceedings of the VLDB Endowment 6.11:1033-1044, 2013.

[9]. Zaharia, Matei, et al. "Discretized streams: Fault-tolerant streaming computation at scale."Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.
[10]. Akidau, Tyler, et al. "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, outof-order data processing."Proceedings of the VLDB Endowment 8.12: 1792-1803, 2015.
[11]. Verma, Abhishek, et al. "Large-scale cluster management at Google with Borg."Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015.
[12]. Boritz, J. "IS Practitioners' Views on Core Concepts of Information Integrity". International Journal of Accounting Information Systems. Elsevier, 2011.
[13]. Netflix. Distributed Resource Scheduling with Apache Mesos. http://techblog.netflix.com/2016/07/distributedresource-scheduling-with.html
[14]. Newell, Andrew, et al. "Optimizing distributed actor systems for dynamic interactive services.". Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016
[15]. Cohen, William, Pradeep Ravikumar, and Stephen Fienberg. "A comparison of string metrics for matching names and records.". Kdd workshop on data cleaning and object consolidation. Vol. 3, 2003
[16]. Hoffman, Matthew, Francis R. Bach, and David M. Blei. "Online learning for latent dirichlet allocation.". Advances in neural information processing systems, 2010
[17]. Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation. "Journal of machine Learning research 3.Jan: 993-1022, 2003
[18]. Mihalcea, Rada, and Paul Tarau. "TextRank: Bringing order into texts. "Association for Computational Linguistics, 2004.
[19]. Hasan, Kazi Saidul, and Vincent Ng. "Conundrums in unsupervised key phrase extraction: making sense of the state-of-the-art. "Proceedings of the 23rd International Conference on Computational Linguistics: Posters. Association for Computational Linguistics, 2010.
[20]. Broder, Andrei Z. "Identifying and filtering near-duplicate documents. "Annual Symposium on Combinatorial Pattern Matching. Springer Berlin Heidelberg, 2000.
[21]. E. Cohen et al. "Finding interesting associations without support pruning. "IEEE Transactions on Knowledge and Data Engineering, vol. 13, no. 1, pp. 64-78, 2001.
[22]. Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2014.
[23]. Krestel, Ralf, Peter Fankhauser, and Wolfgang Nejdl. "Latent dirichlet allocation for tag recommendation. "Proceedings of the third ACM conference on Recommender systems. ACM, 2009.
[24]. Maskeri, Girish, Santonu Sarkar, and Kenneth Heafield. "Mining business topics in source code using latent dirichlet allocation. "Proceedings of the 1st India software engineering conference. ACM, 2008.
[25]. Apache Kafka. http://kafka.apache.org, 2015.
[26]. Gormley, Clinton, and Zachary Tong. Elasticsearch: The Definitive Guide. "O'Reilly Media, Inc.", 2015.
[27]. Apache Mesos. http://mesos.apache.org, 2015.
[28]. Apache Tika. https://tika.apache.org, 2015.
[29]. Confluent Inc. Kafka-Connect. http://docs.confluent.io, 2015.

Извлечение и анализ информации в современных предприятиях

А.Р. Топчян <a.topchyan@reply.de>
Ереванский государственный университет,
0025, Армения, г. Ереван, ул. А. Манукяна, дом 1

Аннотация. С ростом объема данных и потребности в них одной из основных проблем организаций становится обнаружение природы данных, выявление несомой ими информации и установление того, как и кем они используются. Объем данных и число разнородных систем, используемых для их обработки, растет, данные и системы все время усложняются, и совместное использование этих систем становится все более и более сложным. В этой работе мы описываем интеллектуальную поисковую систему, в основном предназначенную для решения проблемы поиска и обмена информацией в большом многопрофильной организации, в которой уже имеется много действующих систем для каждого отдела. Эта система является неотъемлемой частью совместной оперативной платформы данных (ODP) для исследования и обработки данных.

Ключевые слова: проекты, ориентированные на данные; извлечение информации, потоковая обработа; Mesos; Kafka

DOI: 10.15514/ISPRAS-2016-28(4)-1

Для цитирования: Топчян А.Р. Извлечение и анализ информации в современных предприятиях. Труды ИСП РАН, том 28, вып, 4, 2016, стр. 7-28 (на английском). DOI: 10.15514/ISPRAS-2016-28(4)-1

Список литературы

[1]. Topchyan A.R. Enabling Data Driven Projects for a Modern Enterprise. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, 2016, pp. 209-230. DOI: 10.15514/ISPRAS-2016-28(3)-13
[2]. Rahman, Nayem, and Fahad Aldhaban. "Assessing the effectiveness of big data initiatives."2015 Portland International Conference on Management of Engineering and Technology (PICMET). IEEE, 2015.
[3]. Davenport, Thomas H., and Jill Dych´e. "Big data in big companies."International Institute for Analytics (2013).
[4]. Dunning, Ted, and Ellen Friedman. Streaming Architecture: New Designs Using Apache Kafka and Mapr Streams. O'Reilly Media .2016.
[5]. Marz, Nathan, and James Warren. Big Data: Principles and best practices of scalable real-time data systems. Manning Publications Co, 2015
[6]. Michael Hausenblas and Nathan Bijnens. Lambda Architecture. http://lambda-architecture.net, 2015.
[7]. K. Mani Chandy. vent-Driven Applications: Costs, Benefits and Design Approaches, California Institute of Technology, 2006.
[8]. Akidau, Tyler, et al. "MillWheel: fault-tolerant stream processing at internet scale."Proceedings of the VLDB Endowment 6.11:1033-1044, 2013.

[9]. Zaharia, Matei, et al. "Discretized streams: Fault-tolerant streaming computation at scale."Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.
[10]. Akidau, Tyler, et al. "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, outof-order data processing."Proceedings of the VLDB Endowment 8.12: 1792-1803, 2015.
[11]. Verma, Abhishek, et al. "Large-scale cluster management at Google with Borg."Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015.
[12]. Boritz, J. "IS Practitioners' Views on Core Concepts of Information Integrity". International Journal of Accounting Information Systems. Elsevier, 2011.
[13]. Netflix. Distributed Resource Scheduling with Apache Mesos. http://techblog.netflix.com/2016/07/distributedresource-scheduling-with.html
[14]. Newell, Andrew, et al. "Optimizing distributed actor systems for dynamic interactive services.". Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016
[15]. Cohen, William, Pradeep Ravikumar, and Stephen Fienberg. "A comparison of string metrics for matching names and records.". Kdd workshop on data cleaning and object consolidation. Vol. 3, 2003
[16]. Hoffman, Matthew, Francis R. Bach, and David M. Blei. "Online learning for latent dirichlet allocation.". Advances in neural information processing systems, 2010
[17]. Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation. "Journal of machine Learning research 3.Jan: 993-1022, 2003
[18]. Mihalcea, Rada, and Paul Tarau. "TextRank: Bringing order into texts. "Association for Computational Linguistics, 2004.
[19]. Hasan, Kazi Saidul, and Vincent Ng. "Conundrums in unsupervised key phrase extraction: making sense of the state-of-the-art. "Proceedings of the 23rd International Conference on Computational Linguistics: Posters. Association for Computational Linguistics, 2010.
[20]. Broder, Andrei Z. "Identifying and filtering near-duplicate documents. "Annual Symposium on Combinatorial Pattern Matching. Springer Berlin Heidelberg, 2000.
[21]. E. Cohen et al. "Finding interesting associations without support pruning. "IEEE Transactions on Knowledge and Data Engineering, vol. 13, no. 1, pp. 64-78, 2001.
[22]. Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2014.
[23]. Krestel, Ralf, Peter Fankhauser, and Wolfgang Nejdl. "Latent dirichlet allocation for tag recommendation. "Proceedings of the third ACM conference on Recommender systems. ACM, 2009.
[24]. Maskeri, Girish, Santonu Sarkar, and Kenneth Heafield. "Mining business topics in source code using latent dirichlet allocation. "Proceedings of the 1st India software engineering conference. ACM, 2008.
[25]. Apache Kafka. http://kafka.apache.org, 2015.
[26]. Gormley, Clinton, and Zachary Tong. Elasticsearch: The Definitive Guide. "O'Reilly Media, Inc.", 2015.
[27]. Apache Mesos. http://mesos.apache.org, 2015.
[28]. Apache Tika. https://tika.apache.org, 2015.
[29]. Confluent Inc. Kafka-Connect. http://docs.confluent.io, 2015.