9-2017

Building A Big Data Analytical Pipeline With Hadoop For Processing Enterprise XML Data

Viktor Dmitriyev
University of Oldenburg, Germany, iktor.dmitriyev@uni-oldenburg.de

Felix Kruse
University of Oldenburg, Germany, felix.kruse@uni-oldenburg.de

Hauke Precht
University of Oldenburg, Germany, hauke.precht@uni-oldenburg.de

Simon Becker
University of Oldenburg, Germany,, simon.becker1@uni-oldenburg.de

Andreas Solsbach
University of Oldenburg, Germany, andreas.solsbach@uni-oldenburg.de

See next page for additional authors

Authors
Viktor Dmitriyev, Felix Kruse, Hauke Precht, Simon Becker, Andreas Solsbach, and Jorge Marx Gómez

BUILDING A BIG DATA ANALYTICAL PIPELINE WITH HADOOP FOR PROCESSING ENTERPRISE XML DATA

Research full-length paper

Track 01 - Big Data and Business Analytics Ecosystems

Dmitriyev, Viktor, University of Oldenburg, Germany, viktor.dmitriyev@uni-oldenburg.de

Kruse, Felix, University of Oldenburg, Germany, felix.kruse@uni-oldenburg.de

Precht, Hauke, University of Oldenburg, Germany, hauke.precht@uni-oldenburg.de

Becker, Simon, University of Oldenburg, Germany, simon.becker1@uni-oldenburg.de

Solsbach, Andreas, University of Oldenburg, Germany, andreas.solsbach@uni-oldenburg.de

Marx Gómez, Jorge, University of Oldenburg, Germany, jorge.marx.gomez@uni-oldenburg.de

Abstract

The current paper shows an end-to-end approach how to process XML files in the Hadoop ecosystem. The work demonstrates a way how to handle problems faced during the analysis of a large amounts of XML files. The paper presents a completed Extract, Load and Transform (ELT) cycle, which is based on the open source software stack Apache Hadoop, which became a standard for processing of a huge amounts of data. This work shows that applying open source solutions to a particular set of problems could not be enough. In fact, most of big data processing open source tools were implemented only to address a limited number of the use cases. This work explains and shows, why exactly specific use cases may require significant extension with a self-developed multiple software components. The use case described in the paper deals with huge amounts of semi-structured XML files, which supposed to be persisted and processed daily.

Keywords: Big Data, Hadoop, ETL, ELT, XML, Data Analytical Pipeline.

1 Introduction

Ideas behind big data phenomena are quite widespread in society on various levels. This particular phenomena is hyped across media not only due to the fact that humankind nowadays is able to generate enormous amounts of data in comparison to previous centuries/decades, rather than due to already available and well known use cases demonstrating real value behind application of big data ideas. Another factor behind such enormous popularity of the big data adoption is availability of software tools, which are able to cope with processing of huge data amounts with relative small investments into infrastructure (or in another words by using commodity hardware).

In 2006 Michael Palmer said: “Data is just like crude oil. It’s valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc. to create a valuable entity that drives profitable activity; so must data be broken down, analysed for it to have value” (Palmer, 2006). One decade after the phrase "data is a new oil", we can witness that the term big data becomes more popular in various domains such as science or enterprise. Moreover, it intersects with daily life of every human via wider adaption of big data by corporations (e.g. automotive industry, electronics, etc.). For instance, part of data that is related to customers is almost literally "crude oil" for every single company in a way that this data can be used in order to boost client satisfaction and acceptance rate. Thus, big data brings many use cases that demonstrates real value to be gained and enhance profit after companies create

knowledge out of their raw data (Bange, Grosser, & Janoschek, 2015). However, as time passes by, it is going to be more difficult to analyse bigger amounts of data. Processing will be harder not only due to available amount of data but due to the nature. For instance, there is a common vision, which claims that within the next year's in total 10 percent of all produced data is going to be structured whether remained 90 percentage is going to remain semi- or unstructured (Intel Corporation, 2012; Thusoo et al., 2009).

Currently, software tools processing huge amounts of data are available not only in form of proprietary software systems, but also as open source solutions. Usually, such open source solutions are backed by software giants (e.g. Google, Microsoft, and Facebook). This means, that experiences gathered by data-driven software companies can be reused to some extent in terms of open source projects, which in this case are able to cope with processing huge amounts of data. For instance, in case anyone would like to review open source projects maintained by the Apache Software Foundation, it can be easy figured out that most of the software projects available out there (and also under active development) are actually related to data processing. Beside the software projects that already run by the Apache Software Foundation, there are many project in the incubating /pre-publishing stage. And for sure, besides software giants, other companies want to take part in "big data race" in order not to lose competitiveness on a market and even open new niches for themselves. Having a good list of open source tools, related to data processing, together with desire to integrate data-driven products into the company process leads to an idea to adapt open source software. And as long as many companies have historically adopted and integrated various software tools within its daily business process, it can be time consuming to find the best possible way, to integrate open source based big data analytical pipeline. However, there still exists a common background among multiple companies. For example, in using the Extensible Markup Language (XML) for writing log files.

This paper presents an end-to-end approach on how to process an XML based data in the Hadoop ecosystem context. It demonstrates an approach to address problems based on the usage of an XML files by demonstrating completed end-to-end Extract, Load and Transform (ELT) cycle, which is based on open source stack primary from Hadoop ecosystem extended with own self-developed software components. The use case described in this paper deals with semi-structured XML data with a daily volume of nearly 10/15 GB distributed among millions of XML files.

This paper organized as follows: section 2 provides related work. In sections 3, 4, 5 important outlines on the applied approach including experiment results are present. Finally, conclusion and future work are discussed in the section 6.

2 Big Data, Open Source and Enterprise

Informally, big data can be defined as limitation of analytics and storage capabilities of standard database engines. The term big data is often described by the three Vs: *volume*, *velocity* and *variety*.

Volume states the fact that based on the huge size of the data, there are processing limitations. *Velocity* argues that data input speed is also crucial, because data is generated and inserted with high speed. *Variety* states, that data comes from different heterogeneous sources (e.g. social networks, sensors, transactional data etc.) (Laney, 2001). According to the works (De Mauro, Greco, & Grimaldi, 2016; Ylijoki & Porras, 2016), the definition given by Laney is still relevant. However, De Mauro et. al rephrased the original definition by considering newly appeared during last decade improvements: "big data is the information asset characterized by such a high volume, velocity and variety to require specific technology and analytical methods for its transformation into value" (De Mauro et al., 2016).

Despite big data is kind of a buzzword there is no way for business to escape from it without losing competitiveness on the market. Datameer Inc. in 2013 reported that the major goals for the companies to implement big data are: (1) increase revenue, (2) decrease costs, and (c) increase productivity (Groschupf et al., 2013). However, raw data are useless, without knowledge extraction. That means a proper analytical methods must be applied in order to evolve from raw data, which is usually gathered, to knowledge, which is actually enables to perform a better decision. Mainly, there are three paradigms

that are used to implement analytics over data on a large scale: SQL, MapReduce (Dean & Ghemawat, 2008). As well as in-house implementation. SQL proved it's applicability by the long and robust history of usage during the last five decades. While MapReduce appeared less than a decade ago, it's already one of most adopted computational approach to support complex analysis on huge volumes of data. Currently, there are debates in the academia on a topic of selecting "silver bullet" for complex data analysis.

Multiple researches are arguing that SQL wasn't designed for the current needs, and new paradigms, like MapReduce, should overtake an analytical challenges addressed by the era of big data. Such works as (Chen & Hsu, 2013; Pavlo et al., 2009) showed that database management systems with SQL on board were significantly faster and required less code to implement information extraction and analytical tasks. But a process of database tuning and data loading takes more time in comparison to MapReduce.

Extract, Transform and Load (ETL) or Extract, Load and Transform (ELT) process is an important part of any analytical pipeline. Delivering data into analytical part of a pipeline is crucial. Because in the case process of a data import won't succeed, there will be no data to process. And having no data to process leads to the complications in further steps of any analytical pipeline. There are already existing guidelines that are combining ideas from big data and business intelligence/data warehousing from enterprise architecture perspective such as ELTA (Dmitriyev, Mahmoud, & Marin-Ortega, 2015). However, concepts like ELTA are not providing fine-granular and detailed information about processing semi-structured data like XML, which is widespread and widely adapter as a data format within enterprise.

The paper from Fadika et. al works with the parallel and distributed approaches for processing large-scale XML datasets (Fadika, Head, & Govindaraju, 2009). It primary focuses on the performance metrics for processing large XML datasets. For the parallel approach authors of the work are using their own software called PIXIMAL, whether the Hadoop ecosystem was used for the distributed approach. The work presents a way, how the Hadoop ecosystem can be used for processing large scale XML datasets with the Apache Axis Toolkit to parse the XML datasets. However, main focus of the work is only in a single step from complete ELT process, which is parsing XML files (Fadika et al., 2009).

Another work that demonstrates processing/parsing of XML files with MapReduce programming pattern is done by Zinn et. al in 2010 (Zinn, Bowers, Köhler, & Ludäscher, 2010). The approach of XML parsing provided within the work applies MapReduce pattern on the semantical level of XML. Thus, letting parsing happen directly inside Map and Reduce phases. Despite such approach is efficient, having a lot of values/nodes/scopes inside XML file causes too many Reduce tasks running simultaneously. Another benefit of the current work in comparison with work done by Zinn et. al is flexibility. In the current work, presented SaPPy XML parser can be reused within any distributed engine that is able to execute Python code (e.g. Apache Spark). Similar, but with more features (e.g. labelling, indexing, etc.) approach was demonstrated in the work (Kunfang & Lu, 2016).

The Xbase project presented by Yan et. al handles XML files in Hadoop with in healthcare scenarios (Li, Yan, Yan, & Zhang, 2010; Yan, Zhang, Yan, & Li, 2010). However, in the particular case of the Xbase main requirement was to smoothly integrate together RDBMS, Hadoop and XML. Thus, XML files inside Hadoop have to be fully indexed, and be available for further bi-directional processing. The work of Khatchadourian et. al presents an extension of XPath to process XML data in the Hadoop called ChuQL (Khatchadourian, Consens, & Siméon, 2011b). ChuQL is "... a language that allows XML to be processed in a distributed manner using MapReduce, it extends the syntax, grammar, and semantics of XQuery, and also leverages the compositionality, side-effects, and higher-order functions of XQuery 3.0 ..." (Khatchadourian, Consens, & Siméon, 2011a).

Authors of the ChuQL language are based their solution on the StreamXMLRecordReader. The StreamXMLRecordReader is as well utilized by presented in current paper approach.

The work of Dede et. al deals with large scientific XML data, which should be analysed in an adequate time (Dede, Fadika, Gupta, & Govindaraju, 2011). The authors implemented an indexing solution for large XML data using MapReduce. With the indexes the amount of relevant XML data can be reduced. With less relevant XML data the query will be more efficiency. The approach is to get the relevant

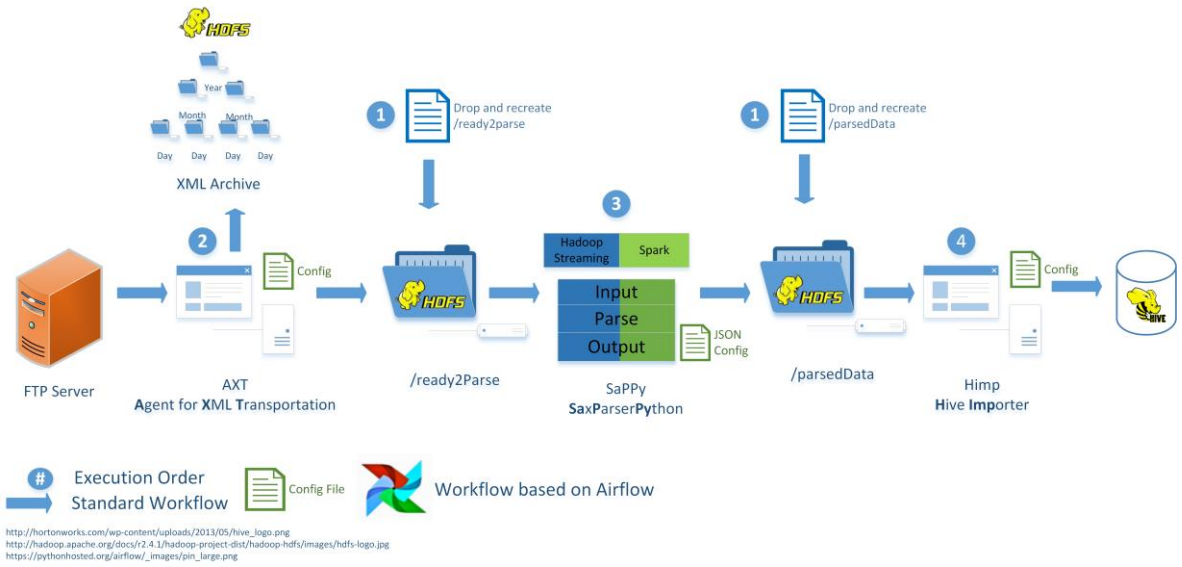

Figure 1. Analytical Pipeline's ELT Data Flow

XML data with an index. Every time the user sends a query against the system, the relevant XML data will be parsed. The system works for simple queries like search a specific number or a string in the XML data.

The approach in this work is to parse the relevant data and make them usable for more complex queries and advanced analytics. This approach can be used to explore the XML data.

The paper from Vasilenko and Kurapati presents another solution to parse XML data with the Hadoop (Vasilenko & Kurapati, 2014). Demonstrated in the paper approach implements a new serializer/deserializer. In order to define XML extraction rules directly in the Hive table CREATE statement is used. The solution can handle complex XML data constellations like attributes, name to attributes value inside particular XML tag.

The works done by Antonellis et. al and Zhang et. al both are primary dedicated to fetch only particular parts of the XML data (Antonellis, Makris, & Pispirigos, 2016; Zhang, Quanlin, & Liu, 2015). Both approaches can be also a good alternative for the use cases, when only particular sub-set of the whole data is required.

3 Analytical Pipeline - ELT Overview

The Extract, Load and Transform (ELT) process is shown in figure 1. The primary data source of the ELT is a server with File Transfer Protocol (FTP), which is shown on the left site of the figure. To be more precise, before getting onto the FTP server, firstly, a single XML file is generated on one of a thousands of machines, and then transported to the FTP server every 5 seconds. According to the figure 1, these raw data distribute over many small XML files should be brought to structured form and persisted inside Hive-database. Despite there is a way how to organize mapping between XML data and Hive schema (Luo, Liu, & Watfa, 2014), the use case that was considered within current work required more complex Hive schema structure. The figure shows a completed process of data flow. Below, each step that is flagged with a number is described in more detailed way.

1. **File System Preparation**. This step is optional and only verifies where pre-configured file system structure is already existing to be used in further processing steps. Folders should be created accordingly, it case such structure is not yet existing as part of file system (e.g. first run of the pipeline).

2. **Load Data into HDFS with AXT**. The *Agent for XML Transportation* (AXT) imports data from FTP server into the Hadoop Distributed File System (HDFS) (Borthakur, 2008). AXT is a self-development implemented in Java and provides a number of configuration options. For instance, a workflow is designed in such a way that data of a whole day are loaded into HDFS at once. Thereby AXT stores an original files in a folder, which is organized in a hierarchic structure as follows - year, month, and day. Further, AXT prepares loaded XML data for transformation and saves them into "/ready2Parse" folder. In section 4 functionality of AXT described in details.
3. **Parse Data with SaPPy**. The *Sax Parser on Python* (SaPPy) parses given XML file and saves obtained data inside structured CSV files. SaPPy uses a configuration file for extraction information from XML files. Which means it can be adopted according to particular XML schema. In order to use benefits of the Hadoop, SaPPy is executed either with Spark (Zaharia et al., 2012) or Hadoop Streaming (Apache Software Foundation, 2017c). Both solutions allow a parallel and distributed execution on the cluster. Details of SaPPy and it's execution with Spark/Hadoop Streaming are presented in section 5.
4. **Import Data into Hive-database**. The last task in the ELT process is to import obtained after XML parsing CSV files into a Hive-database for further processing. This step is done by the Hive-Importer (HImp), which is a small and lightweight Java based utility. HImp scans predefined via configurations folder (e.g. "/parsedData") and imports CSV files into Hive-tables. The configuration file specifies the matching between CSV files and tables in located in Hive-database schema.

Further sections provide more details information about each particular steps described above.

4 Analytical Pipeline - From FTP to Hadoop

In some cases, application of big data ideas can be harder then it seems to be. For instance, many software tools that are dealing with data import tasks (e.g. Apache Kafka, Gobblin, Logstash) are "chuck-based", which means losing some raw data (e.g. one line or a chunk) during a loading process is acceptable as long as most of the data were delivered correctly into target system. But in some cases, it's unacceptable and even not possible. For instance, in case of usage of XML format. Let's thinks about what will happen in case one part of XML file will be accidentally lost. In such case, a complete XML file will be corrupted and cannot be further processed within a pipeline. Such circumstances are leading to the idea that not all software tools available out there in form of the open source are going to work for the use case, which includes processing of huge amount of files in XML format. On the other hand, due to historical reasons (e.g. best practices from enterprise, XML-tools availability, etc.) XML based data are widespread and heavily adopted as intermediate or even primary data formats within a historically grown information systems and applications. And as long as most popular de-facto data processing tools are not dedicated to process "consistent" data formats (the ones that cannot be broken into chunks) such as XML processing large amounts of XMLs can be a challenging task.

Since proper end-to-end handling of XML files is not well supported within the Hadoop ecosystem by default, a special software component was developed to handle the transportation of XML files into a targeted HDFS. As it was mentioned above, this developed software tool is called Agent for XML Transportation (AXT).

Besides its direct straightforward purpose to transport of XML files from FTP into HDFS, AXT is also able to merge and enrich these files. Since XML files might be compressed with any particular compression, in case developers decided to save some space and applying compression before saving XML file on FTP. Decompression of previously compressed files should be firstly performed by a AXT in order to make data inside HDFS consumable by parser. In this particular use case, ZML files represent a compressed version of original XML files, which means AXT is able to decompress ZML files out of the box.

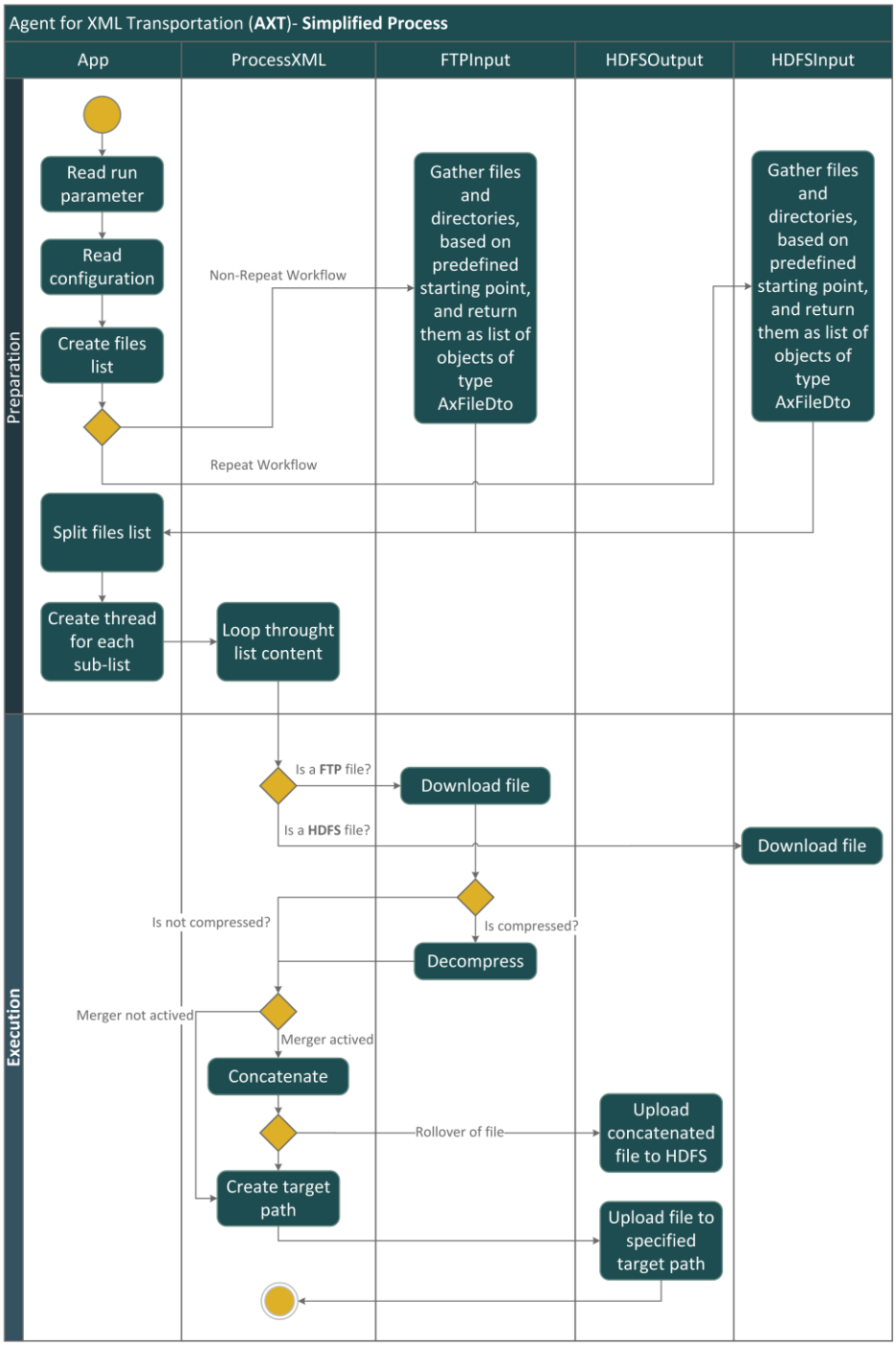

Figure 2. AXT Activity Diagram

In order to provide a user with full control over AXT, a configuration file is used. It allows user to handle base settings of AXT without interfering with source code. Furthermore, runtime parameters can be set for each run of AXT. A standard procedure of AXT is shown in form of activity diagram in figure 2.

When AXT starts, the App class is run to read passed to it runtime parameters. Then, a configuration context, which is based on the settings from a configuration file, will be loaded to be further used with instantiated version of AXT. As the next step, a list of files will be created. Normally, file list is gathered from FTP server. However, AXT also supports "repeated workflow". It means that already imported before into HDFS XML files are going to be processed again.

Before further processing, a retrieved list of files will be split into several sub-lists based on the preconfigured amount of threads. By running data extraction in multi-threading mode AXT achieves better

performance, despite introducing an overhead on software development and testing phases. For each sub-list, the following steps are taking place:

1. As the first step, AXT checks if current file comes from FTP server or from HDFS. Then, based on the file type, a different download logic is applied. If it is a file from FTP, AXT checks if the file is compressed (e.g. ZML). Such check won't be performed if a current file comes from HDFS, since in the mentioned scenario no compressed files will be stored in a HDFS. In case file comes from FTP in a compressed form, AXT will decompress it.
2. As the second step, AXT checks if a "merger" is activated In case merger mode was activated, XML files concatenation will take place. This is done in order to achieve greater performance while working with HDFS blocks storage approaches. Also AXT checks after each merge, if a roll-over should take place. The roll-over behaviour is configured via a settings file and it's based on a size of a single merged file.
3. The last step generates a destination path within targeted HDFS, followed by the upload of a produced concatenated (e.g. merged) and decompressed XML files.

As shown in this section, data import task can be crucial. The evaluation of tools such as Apache Kafka, Gobblin, Logstash, which are existing in the Hadoop ecosystem, showed that none of these software tools were able to fully address specific requirements of appearing while working with the XML files. Even though, XML is a common file format, there is no proper open source tool available out there to be used in combination with FTP Server and HDFS. Inspired by this knowledge, a new data import tool was developed with focus on the proper handling of XML files. The Agent for XML Transportation (AXT) is able to download XML files from an FTP server, enrich them with metadata and upload these files into HDFS. Next to this, it can concatenate files in order to meet the configured block size of the targeted HDFS. Since it uses configuration files and various run/start parameters, it is highly configurable.

5 Analytical Pipeline - From Semi-structured to Structured Data with Hadoop

In order to get a structure out of semi-structured data from the merged XML files, a XML parser should be used in order to extract values from given XML file according to the given schema.

The figure 3 shows how exactly XML parser can be started on cluster in order to get valuable data out of it. Mainly, there are two possibilities existing: (1) Hadoop Streaming and (2) Apache Spark. The figure shows the XML data as the data source and the CSV data as the output. Both input and output are stored in the HDFS. The XML parser (see point 2 in the figure 3 "XML Parser + JSON Configuration") transforms a given XML file into structured CSV file according to the preconfigured via JSON file configuration. As long as parser implemented in Python programming language it can be used without any modifications with Apache Spark engine and with Hadoop Streaming.

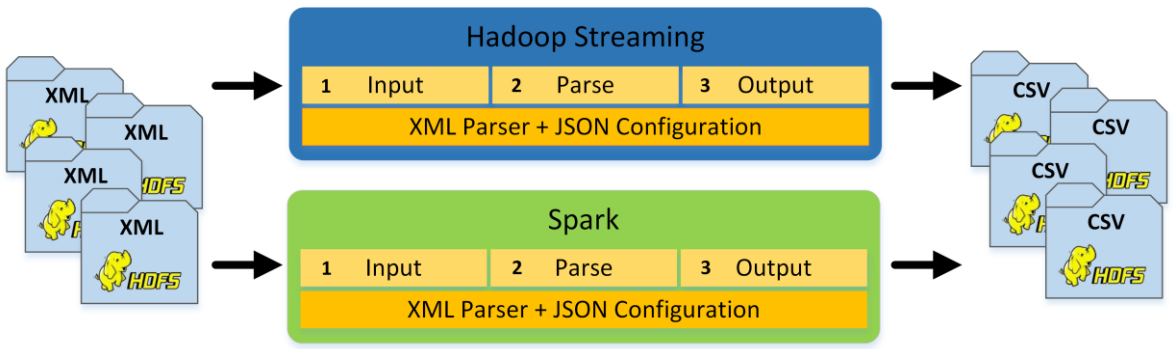

Figure 3. Processing XML files with SaPPy XML Parser via Hadoop Streaming/Apache Spark (adapter from (Marx Gómez, Dmitriyev, Precht, & Kruse, 2016))

XML Parser: Since XML files in the current use case can be more than hundred megabytes in size per file, it is not recommended to use a Document Object Model (DOM) based XML parsers (Refsnes Data, 2017). Because such type of parsers usual have a huge memory footprint, while parsing particular XML file. Another drawback of DOM usage can be purely software implementation driven, which means particular implementations of DOM parsers may have memory leaks (which is actually a case, when standard XML DOM Parser from Python 2/3 is used). Despite, special benchmarks were perform in order to identify best possible XML parsers in Python 2/3 (from memory and performance perspective) , investigation and description of memory leaks of a Python based XML parsers are out of scope of current work. This is because a DOM based parser tries to read in the whole XML file into the RAM which might cause exceptions like "out of memory exception" due to the fact, that the size of a file is multiplied in the RAM since a full object structure is built for further processing. Instead of DOM based parser, a Simple API for XML (SAX) based parser was developed (Megginson, 2004). It was called *Sax Parser on Python* (SaPPy). A SAX based parsers, works with events, which means a parser fires so called "open tag event". And as long as such event was fired is should be handled by an event listener. In this scenario, such type of event listener uses a configured list of relevant tags, attributes and values in order to identify parts of XML that are containing relevant information to be extracted. With this approach, SaPPy is highly configurable and applicable to other use cases, which tend to parse XML files using schema different from the one used by current use case. Once a tag or attribute of interest was found, the information is extracted and stored in a temporary data structure. This temporary data structure allows later on to transform fetched from XML values into desired output. After file was processed data structure obtained at the end is written to an output that consists out of multiple CSV files. Structure of CSV files can be defined via another configuration. The output structures of CSV files matches the table definition of the relational Hive-based database schema. These generated outputs are further processed by HImp, which takes out files and import them into Hive-database (see section 3).

Hadoop Streaming and Apache Spark: The figure 3 shows the Hadoop Streaming (blue) and the Spark (green) solutions to parse XML data in a Hadoop. Hadoop Streaming is out of the box feature provided by Apache Hadoop framework. Hadoop Streaming is based on the MapReduce paradigm. You can execute any executable or script as mapper and reducer (Apache Software Foundation, 2017c). In this use case the mapper and reducer are developed in Python. The mapper program parse the XML Data in parallel in the cluster and the reducer program creates the CSV file output. Apache Spark is an open source framework backed by multiple companies and maintained by the Apache Software Foundation. It is an in-memory cluster computing framework. Apache Spark can be used as a standalone solution and also in combination with the Apache Hadoop framework. In the case of combination with the Apache Hadoop the YARN should be used as the resource manager (Vavilapalli et al., 2013). The framework has a Python, Java and Scala API. Furthermore it offers higher-level tools such as Spark SQL, MLlib for machine learning, GraphX for graph processing, and Spark Streaming (Apache Software Foundation, 2017a). In this use case a Python related Spark APIs were used. This script loads XML files from the HDFS into the Apache Spark datatype named RDD (Resilient Distributed Dataset). Then the script parses loaded XML files in parallel fashion by utilizing cluster computational power. And after parsing of XML files succeeds, results are dumped back into HDFS in form of CSV file. From the performance perspective, Apache Spark was two times faster in comparison with Hadoop Streaming on the cluster with the same hardware configurations. However, it should be explicitly noted that Apache Spark was not initially dedicated to process data once, like in case of the XML parsing. Besides that, it should be also taken into consideration that Apache Spark can be hard to troubleshoot in case something will go wrong. For example, in case it will be not enough memory dedicated to the container, Apache Spark executors can be simply dropped without much information left inside log files. Such issues happen extremely often while trying to construct the above described parsing pipeline using DOM instead of SAX.

In order to be able to process parsed data that were stored inside CSV files a specific Hive schema was designed. This schema was a result of analysis of a structure of the targeted XML, which lead to construction of dedicated relational database model. Hive does not offer any possibilities to define con-

straints in form of foreign keys, in comparison to other databases conformed with Atomicity, Consistency, Isolation and Durability principles (Hellerstein, Stonebraker, & Hamilton, 2007). Nevertheless the tables are linked by virtual relations, thus a relational database systems functionality was simulated in order to make further data processing smoother. This relations have to be later utilized by an applications. For instance, a specific dashboard with Key Performance Indicators (KPI) was implemented in Apache Zeppelin (Apache Software Foundation, 2017b). As long as the workflow supposed to be executed once a day, which means a new data have to be loaded once a day, there supposed to be no problems with data consistency. However, the workflow could also be executed again. For instance, in case some new data are coming from the day, which was already processed and persisted inside Hive database. The files that are already have been persisted inside Hive for that specific day have to be deleted/dropped first. Delete operations in Hive are only available on tables that are supporting ACID. Usually, this functionality is used for relational database models that are supposed to be consistent. However, it should be explicitly mentioned, that in the current scenario (as well as in many other cases related to processing of huge amount of data), the ACID cannot be fully fulfilled due to many reasons (e.g. some data consistency can be ignored due to availability of many samples). Instead of using delete operation on single records, each table was partitioned according to the following three criteria: day, month and year. Using this workaround, HImp is able to drop whole partition of each table (if such exists) before inserting a new data. Furthermore, there is a side-effect regarding the analysis on the schema that lead to marking with a date each record in each table. However, that may be a helpful for faster analysis on incoming from XML files values.

6 Conclusion

Despite ideas of big data are not new and widely adopted in form of the open source software, there are still existing challenges. The current paper shows an end-to-end pipeline for processing huge amounts of XML files inside Hadoop ecosystem. There are a number of lessons were learned while developing such type of a pipeline based, which is primary on the open source stack. For instance, there is a clear understanding that processing of XML files is not fitting into idea of processing data in form of a "chunks". It is not possible due to the nature of XML. According to its format, any XML file should be consistent in order to be valid for processing workloads. It means, that in order to meet consistency requirement of the XML format on a large scale, self-developed software components have to be introduced on various stages of the ELT cycle. In case of current work, such tools are AXT, SaPPy, and HImp. Another lesson learned is about efforts have to be invested, before any open source tool will work properly according to custom requirements of a use case. Thus, a decision about adopting any open source stack should consider the amount of time that is needed for constructing required tool set. Besides that, there is always a possibility that there are going to be a set of required components, which have to be self-developed or significantly adjusted.

References

Antonellis, P., Makris, C., & Pispirigos, G. (2016). Distributed XML Filtering Using HADOOP Framework. In *Algorithmic Aspects of Cloud Computing* (pp. 75–83). Springer.

Apache Software Foundation. (2017a). Apache Spark 2.2.0 Documentation. URL: http://spark.apache.org/docs/latest/ (visited on 24.07.2017)

Apache Software Foundation. (2017b). Apache Zeppelin. URL: https://zeppelin.apache.org (visited on 24.07.2017)

Apache Software Foundation. (2017c, July). Apache Hadoop MapReduce Streaming. URL: https://hadoop.apache.org/docs/r2.7.3/hadoop-streaming/HadoopStreaming.html (visited on 24.07.2017)

Bange, C., Grosser, T., & Janoschek, N. (2015). Big Data Use Cases 2015 – Getting real on data monetization. In *SAS, Business Application Research Center – BARC GmbH* (Vol. 15–17, pp. 69–91).

Borthakur, D. (2008). HDFS architecture guide. In *Hadoop Apache Project http://hadoop.apache.org/common/docs/current/hdfsdesign.pdf* (p. 39).

Chen, F., & Hsu, M. (2013). A performance comparison of parallel DBMSs and MapReduce on large-scale text analytics. In *Proceedings of the 16th International Conference on Extending Database Technology* (pp. 613–624). ACM.

De Mauro, A., Greco, M., & Grimaldi, M. (2016). A formal definition of Big Data based on its essential features. In *Library Review* (Vol. 65, pp. 122–135). Emerald Group Publishing Limited.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. In *Commun. ACM* (Vol. 51, pp. 107–113). New York, NY, USA: ACM. https://doi.org/10.1145/1327452.1327492

Dede, E., Fadika, Z., Gupta, C., & Govindaraju, M. (2011). Scalable and Distributed Processing of Scientific XML Data. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing* (pp. 121–128). Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/Grid.2011.24

Dmitriyev, V., Mahmoud, T., & Marin-Ortega, P. M. (2015). SOA enabled ELTA: approach in designing business intelligence solutions in Era of Big Data. In *International Journal of Information Systems and Project Management* (Vol. 3, pp. 49–63). https://doi.org/10.12821/ijispm030303

Fadika, Z., Head, M. R., & Govindaraju, M. (2009). Parallel and distributed approach for processing large-scale XML datasets. In *2009 10th IEEE/ACM International Conference on Grid Computing* (pp. 105–112). https://doi.org/10.1109/GRID.2009.5353070

Groschupf, S., Henze, F., Voss, V., Rosas, E., Krugler, K., Bodkin, R., … Shelley, P. (2013). The Guide To Big Data Analytics. *Datameer Whitepaper*.

Hellerstein, J. M., Stonebraker, M., & Hamilton, J. (2007). Architecture of a Database System. In *Foundations and Trends® in Databases* (Vol. 1, pp. 141–259). Now Publishers, Inc. https://doi.org/10.1561/1900000002

Intel Corporation. (2012, June). Einführung in Big Data: Die Analyse unstrukturierter Daten. http://www.intel.de/content/dam/www/public/emea/de/de/pdf/unstructured-data-analytics-paper.pdf

Khatchadourian, S., Consens, M. P., & Siméon, J. (2011a). Having a ChuQL at XML on the Cloud. In *AMW*. Citeseer.

Khatchadourian, S., Consens, M., & Siméon, J. (2011b). ChuQL: Processing XML with XQuery Using Hadoop. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research* (pp. 74–83). Toronto, Ontario, Canada: IBM Corp. http://dl.acm.org/citation.cfm?id=2093889.2093897

Kunfang, S., & Lu, H. (2016). Efficient Querying Distributed Big-XML Data using MapReduce. In *International Journal of Grid and High Performance Computing (IJGHPC)* (Vol. 8, pp. 70–79). IGI Global.

Laney, D. (2001). 3-D Data Management: Controlling Data Volume, Velocity and Variety. In *META Group Research Note, February* (Vol. 6).

Li, W.-S., Yan, J., Yan, Y., & Zhang, J. (2010). Xbase: cloud-enabled information appliance for healthcare. In *EDBT* (pp. 675–680). ACM. https://www.microsoft.com/en-us/research/publication/xbase-cloud-enabled-information-appliance-for-healthcare/

Luo, W., Liu, B., & Watfa, A. K. (2014). An open schema for XML data in Hive. In *Big Data (Big Data), 2014 IEEE International Conference on* (pp. 25–31). IEEE.

Megginson, D. (2004). SAX: Simple API for XML. http://www.saxproject.org/ (visited on 24.07.2017)

Palmer, M. (2006). Data is the New Oil. URL: http://ana.blogs.com/maestros/2006/11/data_is_the_new.html (visited on 24.07.2017)

Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., & Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (pp. 165–178). Providence, Rhode Island, USA: ACM. https://doi.org/10.1145/1559845.1559865

Refsnes Data. (2017). w3schools.com : XML DOM Tutorial. URL: https://www.w3schools.com/xml/dom_intro.asp (visited on 24.07.2017)

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., … Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. In *Proceedings of the VLDB Endowment* (Vol. 2, pp. 1626–1629). VLDB Endowment.

Vasilenko, D., & Kurapati, M. (2014). Efficient processing of xml documents in hadoop map reduce. In *International Journal on Computer Science and Engineering* (Vol. 6, p. 329). Engg Journals Publications.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., … others. (2013). Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing* (p. 5). ACM.

Yan, J., Zhang, J., Yan, Y., & Li, W.-S. (2010). Enabling XML Capability for Hadoop and Its Applications in Healthcare. In *Cloud Computing and Software Services* (p. 355).

Ylijoki, O., & Porras, J. (2016). Perspectives to definition of big data: a mapping study and discussion. In *Journal of Innovation Management* (Vol. 4, pp. 69–91).

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., … Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (pp. 2–2). USENIX Association.

Zhang, Y., Quanlin, L., & Liu, B. (2015). MapReduce implementation of XML keyword search algorithm. In *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on* (pp. 721–728). IEEE.

Zinn, D., Bowers, S., Köhler, S., & Ludäscher, B. (2010). Parallelizing XML data-streaming workflows via MapReduce. In *Journal of Computer and System Sciences* (Vol. 76, pp. 447–463). Elsevier.