Tools and data integration for eScience platforms

Romulo Goncalves[1], Niels Drost[1], Stefan Verhoeven[1], Jisk Attema[1], and Jason Maassen[1]

[1]NLeSC Amsterdam, The Netherlands {*r.goncalves,n.drost,s.verhoeven,j.attema,j.maassen*}*@esciencecenter.nl*

***Abstract*—**

I. Introduction

Data heterogeonity. Multi-disciplinary research leads to the utilization of different libraries developed for different platforms.

Tools integration. Central hub for data distribution.

The tools and data integration work of Sherlock project aims to provide easy and efficient data injection into a HDFS cluster, batch processing, interactive processing and export of results to external processing or storage systems. The decision to use HDFS as a data hub is due to its utilization in Hansken, NFI system for forensic research, but also to exploit current state-of-the-art solutions for Big Data exploration such as Spark. The diagram in Figure 2 shows how each component is interlinked with a Hadoop 2.0 cluster. It has a data import docker to convert different file types to the most appropriated file format supported by Hadoop.

The choice of its format is based on the type of processing with the aim to improve processing efficiency and have a low storage footprint. To exploit data locality and fault-tolerance on a Hadoop Cluster, external libraries are dockerized and through MRdocker, it instantiates a docker image in the Map phase of a MapReduce job, or SprDocker, it instantiates a docker image on each data node, are executed. The input, the intermediate, or the final result is made accessible through the Scala, the R, and the Python interfaces of Spark for a step-wise forensic exploration. Such interfaces allow the users to develop web-apps using R-shiny, or Python fask to feed Java Script web-pages. It provides a user-friendly visualization with all the heavy computations being done by Spark. In case the user wants to process intermediate results somewhere else, a bridge through a NFS mount or through a specialized data exporter is provided by our solution. For example, it allows the data to be loaded into external database for later re-use or be consumed by an external data visualization tool running in a docker.

How do we efficiently access data stored in different formats (possibly in a distributed setting)? How do we combine different tools into a concise workflow? How do we combine the results of each tool into a single result? How do we execute this workflow efficiently?

The remainder of the paper is as follows. Section **??** discusses the general architecture. In Section **??**, through different use case scenarios, flexibility and efficiency on exploring climate and geo-spatial data is shown. Finally the article ends with a summary in Section **??** and future plans in Section **??**.

II. Background

A. Forensics eco-system

One of the challenges encountered every day on forensic research is the large and heterogeneous collection of libraries, tools, and data available to us. Our project designs and prototypes solutions for the integration of many existing components into a single system which is flexible, robust and effective for forensic data exploration.

External libraries have may years of domain knowledge which is not easily converted into a realtional query. They are part of a long and complex pipe-line which can't all be replaced by a relational or a series of relational queries. The access to external repositories to consume data and feed results back to visualization tools or data services turns the conversion of the pipeline into series of relational queries even more complex.

B. Hadoop eco-system

C. Docker Containers

Docker combines an easy-to-use interface to Linux containers with easy-to-construct image files for those containers. In short, Docker launches very light weight virtual machines.

According to the Docker website, "Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments.", www.docker.com. For more information how how virtualization works and how docker differs from virtual machines check link and its user guide. More info is available at the Docker workshop gitHub.

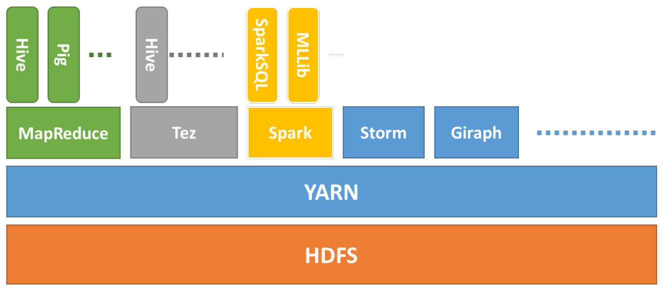

Fig. 1: Hadoop 2.0

III. Architecture

Different state-of-the-art technologies are used to encapsulate and interlink each component in an efficient manner. Through Docker containers each library and all the context on which it depends is wrapped into a light virtual image, i.e., a Docker container. Through this technology the library is easily deployed on different platforms.

A. Data Import

Different type of files will be imported using a Data Import container.

Small files Avoid data fragmentation Expose file metadata, scientific data sets are rich in metadata.

In the context of Hansken, HDFS is used as the data hub. On each step of the processing pipe-line each tool consumes input data stored in HDFS and the outputs the result back to HDFS to be visualized or simply re-used by other tools. Hence, to exploit data locality, have balanced resource utilization and fault-tolerance, the Hadoop 2.0 infra-structure is used to schedule and execute Docker instances over data stored in in HDFS.

Ofcourse, you can just copy your files using 'hdfs fs -put' commands. However, this will quickly lead to performance issues on the HDFS filesystem when you have a large number of files (fi. when each file is a twitter message) You can reduce the number of files by archiving (and zipping) them, but this makes reading them from your spark job difficult. To get the most out of HDFS, you need to store your data in a format that is a 'Resilient Distributed Dataset' or RDD. An RDD consists of a set of records with, depending on the format, extensible metadata. A RDD is split or joined automatically along the records for processing on the hadoop cluster. This will allow you to use all features like data locality, job scheduling etc. For optimal performance, it is recommended to convert your data to small number of large RDD files.

1) ASCII formats: For text data, JSON and XML like formats cannot be split at arbitrary places (ie. you need the pair of opening and closing tags for XML to stay valid). CSV or record JSON, where each line of the file is an independent record, is highly recommended.

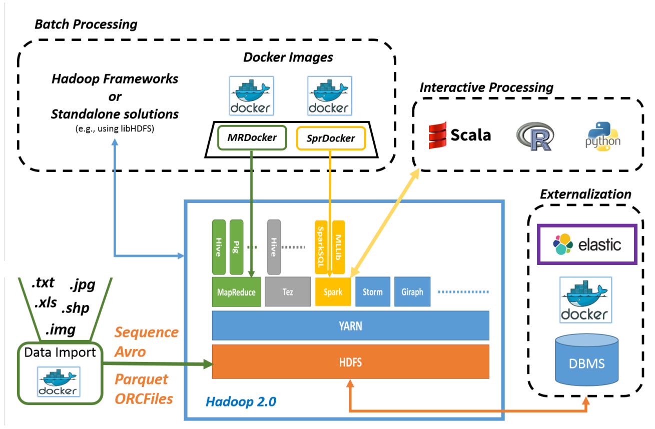

Fig. 2: Data and Tools Integration

General text data can also be converted to a number of binary formats: AVRO, ORC, Parquet, and Sequence files. A binary file can hold multiple records, which facilitates dealing with large datasets, and some formats allow you to apply compression. Other considerations are:

* amount of metadata * extensability of the the metadata schema: can you add extra fields without having to reimport all data? * Optimizations on searching on columns * Support for the format: Hive only, hadoop native, ... * Availability of supporting tools

AVRO and sequence files are easiest to work with, have native Hadoop support, and usable command line tools exist. They are not a column store, so performance is slightly lower than the other formats. ORC seems to become the default format, but support and tooling is still immature. A combination of AVRO + Parquet seems possible, too.

See the discussion on the following pages:

This link provides an indepth discussion on data practices, considering other aspects than performance and filesize:

and some more blogposts:

2) Binary formats: Support for binary data is very limited, only sequence files are a robust solution at the moment. AVRO supports fixed-length binary blobs, and the other formats can be used for small binary files with some tricks (like converting binary to base64 text, expanding the data with a factor of 2. Compression could reduce some of this overhead.)

Import from ————

Import from a Database using Apache Sqoop (<https://sqoop.apache.org/>). Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. You can use Sqoop to import data from external structured datastores into Hadoop Distributed File System or related systems like Hive and HBase. Conversely, Sqoop can be used to extract data from Hadoop and export it to external structured datastores such as relational databases and enterprise data warehouses.

Import from binary Files using forqlift (<http://www.exmachinatech.net/projects/forqlift/>). Commandline tool to work with sequence files File types: text, compressed text, BytesWritable ie. any binary file format.

Import from Text (JSON, CSV) using Avro since it has Hadoop native support: java and python bindings; Row based, not optimized for statistics / aggregates over the metadata; Only fixed-size binary blobs.

Avro files are quickly becoming the best multi-purpose storage format within Hadoop. Avro files store metadata with the data but also allow specification of an independent schema for reading the file. This makes Avro the epitome of schema evolution support since you can rename, add, delete and change the data types of fields by defining new independent schema. Additionally, Avro files are splittable, support block compression and enjoy broad, relatively mature, tool support within the Hadoop ecosystem.

Apache Avro^{TM} is a data serialization system, providing: Rich data structures; A compact, fast, binary data format; A container file, to store persistent data; Remote procedure call (RPC); Simple integration with dynamic languages; Code generation is not required to read or write data files nor to use or implement RPC protocols; Code generation as an optional optimization, only worth implementing for statically typed languages; From Python: avro, avroknife (pip).

Apache orc, ORC: optimized row columnar can add user metadata per file

ORC is a self-describing type-aware columnar file format designed for Hadoop workloads. It is optimized for large streaming reads, but with integrated support for finding required rows quickly. Storing data in a columnar format lets the reader read, decompress, and process only the values that are required for the current query. Because ORC files are type-aware, the writer chooses the most appropriate encoding for the type and builds an internal index as the file is written. Predicate pushdown uses those indexes to determine which stripes in a file need to be read for a particular query and the row indexes can narrow the search to a particular set of 10,000 rows. ORC supports the complete set of types in Hive, including the complex types: structs,

Apache Parquet, a columnar storage format Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.

B. Batch Processing

1) MapReduce: The first wave of Mappers will be slow because they will have to download the libraries for each Docker instance. However, such image is then cached. We need to configure dockter to not use too much space and thus clean the cache once new type of instances are run.

As first step towards a Map-Docker function was implemented deployed using MapReduce processing paradigm. Such function runs a simple Docker instance using a Java "System call" and through Hadoop streaming API it reads from stdin and writes to the stdout. The second option is to explore the Docker Java API to implement more advanced options to load input data and output the results, but also to start and stop the Docker instance. A set of templates on how to do it can be found at hadoop-streaming-docker.

2) Spark: In search for efficiency, the second step was to deploy Docker instances using Spark on the same YARN cluster. After setting up a SPARK cluster, using the following instructions, the deployment of Spark jobs was studied in the context of Latent Dirichlet Allocation (LDA) which is part of the machine learn library of Spark. LDA is used by the topic modeling team to classify emails by topic. To use LDA efficiently the input data was re-structured into a SequenceFile archives using Forqlift. With the data stores as SequenceFile archives the data preparation phase of LDA takes advantage of data locality and uses all the available resources.

To run a Docker instance a spark app called *spark_docker* was developed. All the instructions on how to run Docker from a Spark app can be found at spark-docker. The *spark_docker* was tested in the context of image classification. The library uses neural networks to classify a set of images and it was developed by the deep learning group. Using a Docker image stored at DockerHub, *spark_docker* instantiated one per node which used as input the local images and outputted into a CSV file the name of the file and the classification result. Such final result is then readable by the visualization layer using the NFS mount.

How to do it for HDFS cluster, we run docker containers in Spark. When using HDFS, better to use Spark because data locality comes for free.

In Spark 1.2, we've taken the next step to allow Spark to integrate natively with a far larger number of input sources. These new integrations are made possible through the inclusion of the new Spark SQL Data Sources API.

https://databricks.com/blog/2015/01/09/spark-sql-data-sources-api-unified-data-access-for-the-spark-platform.html

C. Interactive Processing

1) Python:

2) R-enviroment:

D. Externalization

Furthermore, to make all the files stored in HDFS accessible as POSIX two options were considered: FUSE and NFS. With both solutions HDFS will be mounted as a directory to be provided as input to the Docker instance. After trying both options, NFS seems to be easier to setup and used by a larger user community.

1) Database Management System: Such initial stage could be seen as a data gathering stage where data parallelism is evident.

An option is to use a files directly by a DBMS. For data we are also working on data vaults[1]. Drop the text.

Database with Hadoop integration such has BigSQL 3.0 and Polybase.

Impala which can then process Parquet files.

2) External services: Elastic Search

Web-services through docker containers.

IV. Orchestration

We first use Spark since data is stored in HDFS. So spark is used to exploit data locality and fault-tolerance. Currently TDI team is working on the integration of other team's tools using spark_docker into a concise workflow. For efficient resource utilization, efficient scheduling while co-existing with other applications running on the same cluster, TDI team is looking at Mesos and Kubernete. Furthermore, for flexible heterogeneous data integration and the use of external tables as input data is other direction under consideration where the data injection stands on the same principles as our internal Data Vaults project.

The map of such workflow into the required resources should be delegated to an outside systems with a complete overview of the existent ecosystem.

We schedule different docker containers through spark. How to connect them using Spark?

Then how other scheduling schemas can be integrated?

Different types of schedulers:

Data locality could be questioned and then by using swarm filters we could schedule dockers to specific locations, however, the scheduling algorithm would be of our responsability.

Keubernet, Spark and Zepplin

Comparisson between Mesos and Keubernet, fantastic list of differences and why we will pick Keubernet.

V. System in Action

A. Deep Learning

The Convolutional Neural Networks (CNN) are a type of deep learning networks, Y. Bengio, "Learning deep architechures for AI". CNNs are a family of multi-layer neural networks particularly designed for use on two-dimensional data, such as images. For a quick introduction to CNNs, please refer to section 2.3 of the “Large-scale Computer Vision” overview. For a more elaborate introduction, please refer to the online book “Neural Networks and Deep Learning” and to the vast resources on deeplearning.net. Sherlock

Image classification in forensics is a very common request by the digital forensic investigators.

B. Analyzing document corpora

While searching a device for incriminating evidence, text documents are commonly encountered. Analyzing the content of such documents is a challenging task which would require investigators to read through large volumes of text, without any information to guide them through the search. What is this document about? How does it relate to other documents? Are there any trends amongst them? Do I need to read through this pile of documents? Or could I spot a particular document which is more interesting than the rest?

We will tackle these problems using natural language processing tools. The aim of this group is to use NLP tools to group documents with similar content and provide investigators with the tools to quickly gain insight into large collections of text.

In the field of Natural Language Processing (NLP), topic modeling has become a popular technique for analyzing large sets of documents. Topic models attempt to infer a set of topics (basically, bags of words) and then identify the probability of a topic being contained in a document (e.g. Document1 is made up of 0.5 * Football topic + 0.2 * General sports topic + 0.3 * Drugs and crime topic).

However there are still many open questions when using topic modeling: document pre-processing, optimum number of topics to use, visualization and interpretation of the results, etc.

One focus are is investigating techniques for visualizing results from topic models in the most useful and meaningful way. The ultimate aim is to be able to take a list set of documents (e.g. emails from an inbox) and cluster them into different groups of similar emails (e.g. emails work, family, leisure). To achieve this we will need to use existing natural language processing technique, investigate suitable similarity metrics between documents, and develop (interactive?) visualization techniques.

We will be using the infamous Enron email data set as our document collection.

VI. Future work

—-Towards Generating ETL Processes for Incremental Loading— We presented an approach for generating incremental load processes for data warehouse refreshment from declarative schema mappings.

The basis of our work has been provided by Orchid, a prototype system developed at IBM Almaden Research Center, that translates schema mappings into ETL processes and vice versa. Orchid-generated ETL processes, however, are limited to initial load scenarios, i.e. source data is exhaustively extracted and the warehouse is completely (re)built. Incremental load processes, in contrast, propagate changes from the sources to the data warehouse. This approach has clear performance benefits. Change Data Capture (CDC) and Change Data Application (CDA) techniques are used to capture changes at the sources and refresh the data warehouse, respectively. We defined a model for change data to characterize both, the output of CDC techniques and the input of CDA techniques. Since CDC techniques may su from limitations we introduced a notion of partial change data. We discussed the propagation of partial change data within incremental load processes.

Our approach allows for reasoning on how limitations of CDC techniques determine the set of applicable CDA techniques. That is, it allows inferring satisticable CDA reuirements from given CDC limitations and, the other way round, acceptable CDC limitations from given CDA requirements. We further, demonstrated the exploitation of properties of data sources (such as schema constraints) to reduce the complexity of incremental load processes.

A. Stepping way from HDFS

Outside HDFS cluster we use keubernet to schedule docker containers over each node. A spark cluster can be created, but also a set of containers running othere tools. Using messos for resource management, several processing tools can co-exist at the same time.

The challenge becomes to access heterogenous data and find a common ground to share data. The idea is to exploit data vaults when using a DBMS or Spark. Specialized libraries work with a single data format. They are used often in data preparation phase where classificaiton happens.

Data vaults is not only about importing data, it is also about exporting data. We are inverting the paradigm with this platform, it is not a DBMS on top of Spark or Hadoop, but both co-exist side by side where the query plan abstraction goes on level up. With this query interaction is more user friendly, instead of gathering information we are now extracting knowledge.

Using Kubernet it is the first step towards running Spark with access to external file repositories. For that we will use and extend the latest API released: In Spark 1.2, we've taken the next step to allow Spark to integrate natively with a far larger number of input sources. These new integrations are made possible through the inclusion of the new Spark SQL Data Sources API.

Integration of data vaults into Spark. Access only to Hive Tables.

Monunt external data files as Data Frames. A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.

In Spark not possible to make and RDD out of NetCDF. There are specialized modules for each tool, but the combination of different data sets is not possible.

VII. Summary

References

[1] M. Ivanova, Y. Kargin, and et al. Data Vaults: A Database Welcome to Scientific File Repositories. SSDBM, 2013.