Efficient Data Analysis Pipelines

Teemu Koivisto
teemu.koivisto@helsinki.fi
Department of Computer Science
University of Helsinki
Helsinki, Finland

ABSTRACT

This report shows the many different parts of data analysis pipelines and how they act together especially when they are used to model big data. From the data source, where the data is incubated, it's moved through a connecting link such as internet to data collectors which in turn ensure, that the data is being gathered properly. From data collection it's often times persisted on a database or file storage, depending on the type, size and the amount of the data. Preprocessing is later used to standardize the data to the modelling layer, where it might be used as a part of data lake or data warehouse architecture to gain insights into the data as it evolves. To further simplify this process, cloud services or other higher level frameworks might be used to reduce the complexity of pipeline creation.

KEYWORDS

data pipeline, data engineering, data analysis, big data

1 INTRODUCTION

In any software system the control of the flow of information throughout the system by a series of different processing elements is an integral part of it. This specific process of manipulating data with different processing functions like filtering or sanitizing is more commonly referred to as a **pipeline**, analogous to the physical pipelines. Whereas if the data is used for some specific purpose such as data analysis, one can simply refer to it as a data analysis pipeline; a chain of processes to move and manipulate data designed to be served to a statistical model [4].

This report portrays a quick overview of data pipelines especially for data analysis with two example pipelines.

2 COMPONENTS OF A DATA ANALYSIS PIPELINE

On a technical level some have tried to give semantic definitions for the parts of the pipeline [5]. At its core, a pipeline can be thought of as a **Directed Acyclic Graph (DAG)** composed of different processing **operators**. A **source** produces data whereas a **sink** consumes it and for example a **merge** combines the outputs of two pipes into one output.

DSNS '19, Spring 2019, Helsinki, Finland

This definition however, lacks the wider perspective how a pipeline works as a complete software system. To understand it better, we'll go through the different parts of a data analysis pipeline as layers to illustrate the concept.

2.1 Data source layer

The basis of any pipeline are the producers of the data: the **data sources**. These could be anything from the users of a web service to IoT devices to the sensors of Large Hadron Collider. With the current modern technology this massive data, **big data**, is being produced and collected with unprecedented size and quantity [10].

Which brings a lot of new opportunities and also problems, as the previous ways of managing the data are no longer viable. There also lies a the question, is there anything useful in the data? With such massive data, the signal we are measuring might become increasingly difficult to find. And taking into consideration the resources required to process that amount of data, store it and transport it, the benefits of data collection must be carefully weighted.

Data itself can be of many different shapes and forms which are commonly described as structured, semi-structured and unstructured in big data setting. Each of these types will determine which type of analysis and storage suits them best, which should be analyzed when implementing a data analysis pipeline [4].

2.2 Data collection layer

The output of the data layer are then often sent across internet using TCP/IP to be stored or further processed. Collection could also be done using radio waves or other signals, which might happen more and more often with the increasing amount of Internet of Things (IoT) devices as a part of a big data [10].

A problem with any type of collection is, the assurance that the data will be collected. With a large website such as Google Search or Facebook the servers are receiving billions of visits each day [1] [2]. To handle such traffic, the collection services must also arise to the demands of massive traffic within a moment's notice without crashing the whole system down.

A load balancer, or more often load balancers, serve as a entry point to many web sites, which proxy the incoming traffic to for example a set of application servers. This enables distributing the load from a single server to many enabling much larger throughput and fault tolerance beyond a single application server. The scheduling itself could be done using a Round Robin algorithm [13].

[1] https://searchengineland.com/google-now-handles-2-999-trillion-searches-per-year-250247
[2] https://www.statista.com/statistics/346167/facebook-global-dau/

But even after receiving the message successfully, it's in turn another service's job to process the request. While the application servers might be scaled up to match the need, for very high-frequency data a more specialized service might suit better. Rather, you'd use something like a **Apache Kafka** or **RabbitMQ**. They are **stream processing softwares** which manage the many caveats of ingesting massive amounts of continuous data, streams, and ensure they are processed in a robust manner.

Kafka for example has many useful features such as its own distributed commit log, which enables replaying of streams alongside other useful things like data aggregation. Kafka allows other services to subscribe to certain topics which notify them only for certain events in the stream et cetera. A common use-case for Kafka messages is to off-load them to a HDFS filesystem as a part of Hadoop cluster for further processing [2].

2.3 Persistence layer

After the data has been collected, it's commonplace to persist the data somewhere. This as mentioned can be done using a HDFS filesystem, but more often with regular web services a **relational database management system (RDBMS)** such as **PostgreSQL** is used to persist most of the application data.

Relational databases provide as their name indicates, a relational representation of the data served often from memory allowing the quick joining and combination of different data. They virtually all use also SQL query language which gives a lot of flexibility for querying and manipulating the data. This alongside their strong guarantees on the data's atomicity as in ACID make them suitable for most of data needs of a small-scale web-application, and still serve as a backbone for many larger scale applications [12].

But however, there are problems with a RDBMS on a large distributed scale, where their performance deteriorates under heavy load and thus are not as suitable as other, more scalable solutions. To address this specific problem big data can cause, **NoSQL** databases were invented to allow easier scalability. NoSQL has been described to scale "horizontally" where growing the database happens by adding more servers, whereas traditional SQL databases scale "vertically" where single master is often first upgraded to its maximum [11].

And for NoSQL there are many subcategories such as key-value store, document-oriented databases, wide column store and graph databases [14]. Common to them all is their unrelational data structure and their scalability on a large scale distributed setting [11].

Yet for data such as files or other binary formats a database might not not necessarily be needed. Static file storage such as SSD or HDD drives can be assembled in **RAID** arrays or **Network Attached Storage (NAS)** for enhanced performance and fault-tolerance [7, 15]. However, for completely distributed file system a **Hadoop Distributed File System (HDFS)** might be used which in unison with **MapReduce** computation can process vasts quantities of file data with commodity hardware. Also to bybass the complexities involved with maintaining these system by yourself, cloud providers such as AWS or GCP are commonly utilized to further reduce the burden for managing such a sprawling system. AWS Simple Storage Service (S3) is a popular choice for a highly scalable and robust Object Storage Service (OSS) [4].

2.4 Preprocessing layer

Once the data has been persisted and before it's used for modelling, it often has to be transformed or otherwise manipulated into some form for the model to use.

With big data it's common, that the data in hand comes from a wide variety of sources which each might have to be standardized to some joint format. Preprocessing could include checks for consistency as in missing data or duplicates. Also outlier detection is useful to reduce possible noise from the data [4].

This mayhaps tedious work might comprise 80% of the total work for data analysis which highlights the importance of efficient preprocessing [8].

2.5 Modeling layer

Once the data has been processed into its desirable form, it's ready to supplied for the data analysis model. In the simplest setting a modelling could be done using a Jupyter notebook importing data from a file. However, as the data grows and comes to span a wide variety of different formats, sources and latencies, the problems of big data analytics come evident.

Traditionally the algorithms used in data mining have assumed access to the whole dataset, but with high data volumes and streaming is only provided as fractions of the entire dataset. Also the data streams as unbounded sequences of data points create non-stationary flows, in which the true distribution of the data evolves over time [4].

Therefore special type of computing is required, such as the well-known MapReduce, which has been an integral part of popularizing big data. MapReduce offers a way to do extremely parallelizable computing in a cost-efficient fashion. Other existing solutions for doing large scale computation could be proprietary Google Cloud BigQuery or AWS Athena which allow the querying of massive datasets using SQL language, thus offering a well-known interface for accessing the data [10].

The often problematic integration between different datasets can be managed through a **data lake (DL)** infrastructure. DL is a single repository for storing vast amounts of heterogeneous data in their original formats such as relational tables, XML, pictures or other files. Commonly the storage layer in DL deployments is a distributed file system such as HDFS in which the data is proccesed in parallel with eg. MapReduce [4]. A **data warehouse (DW)** on the other hand is a single store for keeping and managing the data, for example AWS Redshift[3] data warehouse can be used to load the contents of multiple databases into one for only the time they are needed [6].

3 IMPLEMENTATION

As the high overview analysis of the components showed, the actual complexity of the pipelines is vast and complex area, which requires a lot of domain knowledge to understand all parts of it.

To showcase how an actual data analysis pipeline works in practise, I'll view two example data analysis pipelines with first one using a very specific system and the latter a more holistic, general approach.

[3] https://docs.aws.amazon.com/redshift/latest/mgmt/welcome.html

3.1 Example 1: Twitter pipeline

Kejriwal et al. analyzed Twitter tweets for real-life crisis informatics using the Las Vegas shootings as example with a scalable and simple end-to-end pipeline [9]. Its goal was to find a better way to do rapid acquisition, preprocessing, analysis and visualization of a crisis-specific dataset in the aftermath of a new crisis.

Their data source consisted of tweets from Twitter API which the authors wanted to filter and curate for tweets regarding the attack using a combination of unsupervised text embeddings and limited-label active learning to construct a crisis-specific training set.

Large part of the problem in this research was the filtering of unrelated, "negative", tweets from the "positive" ones. While a user could by optimistically specifying keywords and hashtags eg. "las vegas shooting" acquire a high-recall dataset, it would still not be enough high precision. Using a high precision keyword such as "las vegas shooting" would then again miss out on a lot of other, relevant data.

The time restrictiveness of the data is also a problem. Most of the relevant tweets would happen in a short period during and after the shooting and decline quickly over time. If however, such a dataset *could* be acquired in real-time, it would significantly help the crisis workers to gain situational awareness into the crisis as it's unfolding.

To summarize the suggested pipeline, it is mostly focused on the various preprocessing tasks such as filtering and classification of tweets based on keywords. A very important aspect of it is, that it doesn't require user to engineer features or label thousands of tweets. Instead, a series of unsupervised text-embeddings and active learning is done to acquire a dataset with minimal human engineering [9].

3.2 Example 2: Genome sequencing pipeline

Whereas the Twitter pipeline was mostly focused on the specifics of implementing unsupervised and active learning on a fairly small dataset, genome sequencing instruments can produce several terabytes of raw data in one day. To transport and store this large data, a lot of engineering must be put into the construction of the required data pipeline [15].

Three different solutions are proposed, that are either a private cloud solution, where all the infrastructure will be on-premise. The data produced by the sequencing instruments is stored on a large NAS drive, which the nodes in the computing cluster can read as part of their processing pipelines. These high-performance nodes will run the heavy computation tasks supplied as DAGs with a high-throughput networking connections between the devices to enable low latencies.

For a public cloud solution the instruments and the NAS will remain on-premise, but the rest of the infrastructure will be moved to a public cloud such as Alibaba Cloud. There, a fleet of nodes using container technologies will compute the tasks, utilizing a OSS to store the results and the sequencing data. Public cloud's services are also used to allow access from other parties to run their own models on the data.

The final solution presented is a hybrid cloud, where bulk of the computing will be done on-premise using local computing nodes and additional nodes are provisioned from the public cloud as needed. By balancing the cost effectiveness of on-premise hardware and the scalability and services of cloud provider, a suitable compromise is found to minimize the resource costs and operational overhead in this particular setting [15].

4 AIMING TOWARDS A GENERAL APPROACH

As the previous examples showed, when the data analysis pipeline progresses and increases in size, multiple different types of models and variations on the data are used [15]. This makes using a more general purpose tool for creating the pipelines more desirable to lower the managerial overhead involved.

Extract Transform Load (ETL) has been a general method for describing the workflow of transferring data from one source to another. They enable defining modular interfaces which can then be processed together as a chain of processes [6].

ETL however is only a name for the general procedure, and the actual implementation and composition of ETL jobs has to done using a suitable ETL software. Because this field of **data engineering** is still very new, the tools around it are also fairly recent with Apache Beam (2016)[4], Apache Airflow (2014)[5] and Luigi[6] (2014) [4]. Apache Beam is a general-purpose tool for implementing the ETL parts, with support for Spark and Hadoop directly [3].

Apache Airflow and Luigi are more abstract designing tools of the pipelines, where the programs itself are not included but are used with operators such as BashOperator, SQLOperator and PythonOperator to compose the DAGs. This allows using the same programming language to describe all the operations and keeping it in the same version control [1, 4].

5 CONCLUSION

In conclusion, the creation of efficient data analysis pipelines is an arduous task comprising of several complex parts, that can be difficult to manage. To reduce the complexity one can use fully-managed or other proprietary cloud services which enable the analyst to focus their time on the actual data modelling. The pipeline itself might be good to describe using a higher level framework such as Apache Airflow, which make the development of bigger pipelines easier without sacrificing flexibility. The whole pipeline and its layers: data source, collection, storage, preprocessing and modelling, must be thought of as a whole to fully maximize the efficiency of the pipeline.

REFERENCES

[1] Apache Airflow authors. [n. d.]. Apache Airflow Documentation. http://airflow.apache.org/ accessed at 5.4.2019.
[2] Apache Kafka authors. [n. d.]. Apache Kafka is a distributed streaming platform. https://kafka.apache.org/intro accessed at 18.3.2019.
[3] Apache Beamauthors. [n. d.]. Apache Beam: An advanced unified programming model. https://beam.apache.org/ accessed at 5.4.2019.
[4] Paolo Ceravolo, Antonia Azzini, Marco Angelini, Tiziana Catarci, Philippe Cudré-Mauroux, Ernesto Damiani, Alexandra Mazak, Maurice Van Keulen, Mustafa Jarrar, Giuseppe Santucci, Kai-Uwe Sattler, Monica Scannapieco, Manuel Wimmer, Robert Wrembel, and Fadi Zaraket. 2018. Big Data Semantics. *Journal on Data Semantics* 7, 2 (01 Jun 2018), 65–85. https://doi.org/10.1007/s13740-018-0086-2

[4] https://beam.apache.org/
[5] https://airflow.apache.org/
[6] https://github.com/spotify/luigi

[5] Maurizio Drocco, Claudia Misale, Guy Tremblay, and Marco Aldinucci. 2017. A Formal Semantics for Data Analytics Pipelines. *CoRR* abs/1705.01629 (2017). arXiv:1705.01629 http://arxiv.org/abs/1705.01629

[6] Shaker H Ali El-Sappagh, Abdeltawab M Ahmed Hendawi, and Ali Hamed El Bastawissy. 2011. A proposed model for data warehouse ETL processes. *Journal of King Saud University-Computer and Information Sciences* 23, 2 (2011), 91–104.

[7] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. 2009. DiskReduce: RAID for Data-intensive Scalable Computing. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage (PDSW '09)*. ACM, New York, NY, USA, 6–10. https://doi.org/10.1145/1713072.1713075

[8] Daniel Haas, Sanjay Krishnan, Jiannan Wang, Michael J. Franklin, and Eugene Wu. 2015. Wisteria: Nurturing Scalable Data Cleaning Infrastructure. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 2004–2007. https://doi.org/10.14778/2824032.2824122

[9] Mayank Kejriwal and Yao Gu. 2018. A Pipeline for Post-Crisis Twitter Data Acquisition. *CoRR* abs/1801.05881 (2018). arXiv:1801.05881 http://arxiv.org/abs/1801.05881

[10] Zaigham Mahmood. 2016. *Data Science and Big Data Computing: Frameworks and Methodologies* (1st ed.). Springer Publishing Company, Incorporated.

[11] A. B. M. Moniruzzaman and Syed Akhter Hossain. 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *CoRR* abs/1307.0191 (2013). arXiv:1307.0191 http://arxiv.org/abs/1307.0191

[12] Rabi Prasad Padhy, Manas Ranjan Patra, and Suresh Chandra Satapathy. 2011. RDBMS to NoSQL: reviewing some next-generation non-relational databaseâĂŹs. *International Journal of Advanced Engineering Science and Technologies* 11, 1 (2011), 15–30.

[13] Sreelekshmi S and K. R. Remesh Babu. 2018. Synchronized Multi-Load Balancer with Fault Tolerance in Cloud. *CoRR* abs/1811.01319 (2018). arXiv:1811.01319 http://arxiv.org/abs/1811.01319

[14] Roberto V. Zicari, Marten Rosselli, Todor Ivanov, Nikos Korfiatis, Karsten Tolle, Raik Niemann, and Christoph Reichenbach. 2016. *Setting Up a Big Data Project: Challenges, Opportunities, Technologies and Optimization*. 17–47. https://doi.org/10.1007/978-3-319-30265-2_2

[15] Jitao Yang. 2017. Hybrid Cloud Computing Solution for Streamlined Genome Data Analysis. In *Proceedings of the 9th International Conference on Management of Digital EcoSystems (MEDES '17)*. ACM, New York, NY, USA, 173–180. https://doi.org/10.1145/3167020.3167047