Degree Project in ?

Second cycle, 30 credits

Faster Delta Lake operations using Rust

How Delta-rs beats Spark in a small scale Feature Store

GIOVANNI MANFREDI

Faster Delta Lake operations using Rust

How Delta-rs beats Spark in a small scale Feature Store

GIOVANNI MANFREDI

Master's Programme, ICT Innovation, 120 credits
Date: July 22, 2024

Supervisors: Sina Sheikholeslami, Fabian Schmidt
Examiner: Vladimir Vlassov
School of Electrical Engineering and Computer Science
Host company: Hopsworks AB
Swedish title: Detta är den svenska översättningen av titeln
Swedish subtitle: Detta är den svenska översättningen av undertiteln

Abstract

Here I will write an abstract that is about 250 and 350 words (1/2 A4-page) with the following components:

- What is the topic area? (optional) Introduces the subject area for the project.
- Short problem statement
- Why was this problem worth a Bachelor's/Master's thesis project? (*i.e.*, why is the problem both significant and of a suitable degree of difficulty for a Bachelor's/Master's thesis project? Why has no one else solved it yet?)
- How did you solve the problem? What was your method/insight?
- Results/Conclusions/Consequences/Impact: What are your key results/ conclusions? What will others do based on your results? What can be done now that you have finished - that could not be done before your thesis project was completed?

Keywords

Canvas Learning Management System, Docker containers, Performance tuning First keyword, Second keyword, Third keyword, Fourth keyword

Sammanfattning

Här ska jag skriva ett abstract som är på ca 250 och 350 ord (1/2 A4-sida) med följande komponenter:

- Vad är ämnesområdet? (valfritt) Presenterar ämnesområdet för projektet.
- Kort problemformulering
- Varför var detta problem värt en kandidat-/masteruppsats? (*i.e.*, varför är problemet både betydande och av en lämplig svårighetsgrad för ett kandidat-/masteruppsats-projekt? Varför har ingen annan löst det än?)
- Hur löste du problemet? Vad var din metod/insikt?
- Resultat/slutsatser/konsekvenser/påverkan: Vilka är dina viktigaste resultat/
  slutsatser? Vad kommer andra att göra baserat på dina resultat? Vad kan göras nu när du är klar - som inte kunde göras innan ditt examensarbete var klart?

Nyckelord

Canvas Lärplattform, Dockerbehållare, Prestandajustering Första nyckelordet, Andra nyckelordet, Tredje nyckelordet, Fjärde nyckelordet

Sommario

Qui scriverò un abstract di circa 250 e 350 parole (1/2 pagina A4) con i seguenti elementi:

- Qual è l'area tematica? (opzionale) Introduce l'area tematica del progetto.
- Breve esposizione del problema
- Perché questo problema meritava un progetto di tesi di laurea/master? (Perché il problema è significativo e di un grado di difficoltà adeguato per un progetto di tesi di laurea/master? Perché nessun altro l'ha ancora risolto?)
- Come avete risolto il problema? Qual è stato il vostro metodo/intuizione?
- Risultati/Conclusioni/Conseguenze/Impatto: Quali sono i vostri risultati chiave/ conclusioni? Cosa faranno gli altri sulla base dei vostri risultati? Cosa si può fare ora che avete finito - che non si poteva fare prima che il vostro progetto di tesi fosse completato?

parole chiave

Prima parola chiave, Seconda parola chiave, Terza parola chiave, Quarta parola chiave

Acknowledgments

I would like to thank xxxx for having yyyy.

Stockholm, June 2024
Giovanni Manfredi

Contents

List of acronyms and abbreviations

ACID	Atomicity, Consistency, Isolation and Durability
AI	Artificial Intelligence
API	Application Programming Interface
BI	Business Intelligence
D	Deliverable
DBMS	Data Base Management System
ELT	Extract Load Transform
ETL	Extract Transform Load
G	Goal
HDFS	Hadoop Distributed File System
IN	Industrial Need
JVM	Java Virtual Machine
ML	Machine Learning
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing
PA	Project Assumption
RDD	Resilient Distributed Dataset
RQ	Research Question
SDG	Sustainable Development Goal

Chapter 1

Introduction

Lakehouse systems are increasingly becoming the primary choice for running analytics in large-sized companies (that have more than 1000 employees) [1].

This recent architecture design called Lakehouse [2] is preferred over old paradigms, i.e. data warehouses and data lakes, as it builds upon the advantages of both systems, having the scalability properties of data lakes while preserving the Atomicity, Consistency, Isolation and Durability (ACID) properties typical of data warehouses [2]. Additionally, Lakehouse systems include partitioning, which reduces query complexity significantly and provides "time travel" capabilities, enabling users to access different versions of data, versioned over time [3].

Three main implementations of this paradigm emerged over time [4]:

1. **Apache Hudi**: first introduced by Uber, now primarily backed by Uber, Tencent, Alibaba, and Bytedance.

2. **Apache Iceberg**: first introduced by Netflix and now primarily backed by Netflix, Apple, and Tencent.

3. **Delta Lake**: first introduced by Databricks and now primarily backed by Databricks and Microsoft.

While large communities back all three projects, Delta Lake is acknowledged as the de-facto Lakehouse solution [4]. This is mainly thanks to Databricks, which first promoted this new architecture over data lakes among their clients around 2020 [5].

As a data query and processing engine, Delta Lake is typically used with Apache Spark [6]. This approach is effective when processing large quantities of data (1 TB or more) over the cloud, but whether this approach is effective on small quantities of data (100 GB or less) remains to be investigated [7].

DuckDB [8], a Data Base Management System (DBMS) and Polars [9], a DataFrame library, highlighted the limitations of Apache Spark. When the data volume is small (between 1 GB and 100 GB) and the architecture is processing data locally, an Apache Spark cluster performs worse than alternatives. This ultimately brings an increase in costs and computation time [10, 11].

Another aspect to keep in mind is that thanks to its ease of use and high abstraction level, Python has become the most used programming language in the data science space [12]. Python is currently also the most popular general purpose programming language [13, 14] and it is by far the most used language for Machine Learning (ML) and Artificial Intelligence (AI) applications [15], this is mainly thanks to its strong abstraction capabilities and accessibility. This trend can also be observed by looking at the most popular libraries among developers, where two Python libraries make the podium: NumPy and Pandas [14]. In this scenario, creating a Python client for Delta Lake would be beneficial as it would not have to resort to Apache Spark and its Python Application Programming Interface (API) (PySpark). This approach with small-scale (between 1 GB and 100 GB) use cases would improve performance significantly.

This native Python interface for Delta Lake directly benefits Hopsworks AB, the host company of this master thesis. Hopsworks AB develops a Feature Store for ML, a centralized, collaborative data platform that enables to store and access of reusable features [16]. This architecture also supports point-in-time correct datasets from historical feature data [17].

This project here presented, aims to increase the data throughput for reading and writing on Delta Lake tables that acts as an Offline Feature Store in Hopsworks. Currently, the pipeline is Apache Spark-based and the key hypothesis of the project is that a faster non-Apache Spark alternative is possible. Ultimately if effective, Hopsworks will consider integrating this system implementation into the Hopsworks Feature Store (open source version), greatly improving the experience of Python users working on small quantities of data (between 10 GB and 100 GB). More generally, this work will outline the possibility for Apache Spark alternatives in small scale (between 10 GB and 100 GB) use cases.

1.1 Background

A clear understanding of the background of this project comes from appreciating three different key aspects: Lakehouse development, Apache

Spark relevance and flows, and Python as an emergent language.

Lakehouse is a term coined by Databricks in 2020 [18], to define a new design standard that was emerging in the industry that combined the capability of data lakes in storing and managing unstructured data, with the ACID properties typical of Data warehouses. Data warehouses became a dominant standard in the '90s and early 2000s [19], enabling companies to generate Business Intelligence (BI) insights, managing different structured data sources. The problems related to this architecture were highlighted in the 2010s when the need to manage large quantities of unstructured data rose [20]. So Data lakes became the pool where all data could be stored, on top of which a more complex architecture could be built, consisting of data warehouses for BI and ML pipelines. This architecture, while more suitable for unstructured data, introduces many complexities and costs, related to the need of having replicated data (data lake and data warehouse), and several Extract Load Transform (ELT) and Extract Transform Load (ETL) computations. Lakehouse systems solved the problems of Data lakes by implementing data management and performance features on top of open data formats such as Parquet [21]. This paradigm was enabled by three key technologies: (i) a metadata layer for data lakes, tracking which files are part of different tables, (ii) a new query engine design, providing optimizations such as RAM/SSD caching, and (iii) an accessible API access for ML and AI applications. This architecture design was first open-sourced with Apache Hudi in 2017 [22] and then Delta Lake in 2020 [5].

Spark is a distributed computing framework used to support large-scale data-intensive applications [23]. Spark builds from the roots of MapReduce and its variants. MapReduce is a distributed programming model first designed by Google that enables the management of large datasets [24]. The paradigm was later implemented as an open-source project by Yahoo! engineers under the name of Hadoop MapReduce [25]. Spark significantly improved the performance of Hadoop MapReduce (10 times better in its first iteration) [23] thanks to its use of Resilient Distributed Datasets (RDDs) [26]. RDDs is a distributed memory abstraction that enables a lazy in-memory computation that is tracked through the use of lineage graphs, ultimately increasing fault tolerance [26]. This means that Spark avoids going back and forth between storage disks to store the computation results, as represented in Figure 1.1.

Spark, which is open-sourced under the Apache foundation as Apache Spark [27] (from now on simply Spark), has seen widespread success and adoption in various applications, becoming the de-facto data-intensive

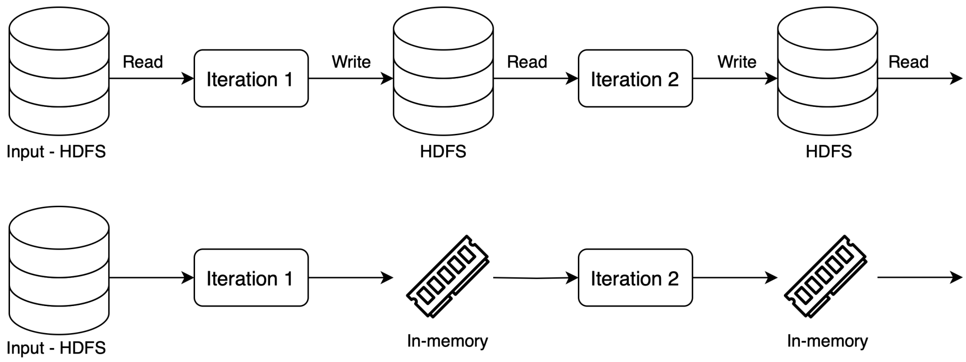

Figure 1.1: Difference between a Hadoop MapReduce execution and an Apache Spark execution. Every step in Hadoop MapReduce must be saved, while Apache Spark operates in-memory.

computing platform for the distributed computing world. While Spark is often used as a comprehensive solution [6], different solutions might be better suited for a specific scenario. An example of this is the case of Apache Flink [28], designed for real-time data streams, which prevails over Spark where low latency real-time analytics are required. Similarly, Spark might not be the best tool for lower-scale applications where the high-scaling capabilities of Spark may not be required. This is the case of DuckDB [8] and Polars [9], that by focusing on low scale (10GB-100GB) provide a fast On-Line Analytical Processing (OLAP) embedded database and DataFrame management system respectively offering an overall faster computation compared to starting a Spark cluster for to perform the same operations. This shows the possibility for improvements and new applications that substitute the current Spark-based systems in specific applications such as real-time data streaming or small-scale computation. In this project, the latter application is going to be explored.

Python can be considered the primary programming language among data scientists [29]. Python was first adopted by many thanks to its focus on ease of use, high abstraction level, and readability. This helped create a fast-growing community behind the project, which led to the development of a great number of libraries and APIs. So now, more than 30 years after its creation, it has become the de-facto standard for data science thanks to many daily used Python libraries such as TensorFlow, NumPy, SciPy, Pandas, PyTorch, Keras and many others.

Python is also considered to be the most popular programming language, according to the number of results by search query (+"*<language>*

programming") in 25 different search engines [30]. This is computed yearly in the TIOBE Index [13]. Looking at the 2024 list, it can be noted that Python has a rating of 15.16%, followed by C which has a rating of 10.97%. The index also shows the trends of the last years, clearly displaying the rise of Python over historically very popular languages such as C and JAVA, which were both outranked by Python between 2021 and 2022. This shows the importance of offering Python APIs for programmers and data scientists in particular to increase the engagement and possibilities of a framework.

1.2 Problem

The Hopsworks Feature Store [16] when querying the Delta Lake-based Offline Feature Store, uses Spark as a query engine, i.e. executes the query (read, write or delete) on the Offline Feature Store.

If no Spark job was started before (as is always the case in the open-source server-less Hopsworks app), the operation, even if small in size (only retrieving 1 GB of data or less), will take a few minutes (1-2 minutes) to complete. This is mainly due to the overhead of starting a Spark cluster and running a Spark job. This overhead is less relevant for computation on larger quantities of data (1 TB or above), as it composes a smaller part of the overall computation time (

INSERT COMPUTATION PERCENTAGE TIME HERE

). Nonetheless, Hopsworks' typical use-case sits between tests on small quantities of data (scale between 1-10 GBs) and production scenarios on a larger scale, but still relatively small (scale between 10-100 GBs). As this overhead is a Spark-specific issue, it grows the need to look for Spark alternatives. Currently, Hopsworks is saving their Feature Store data on Apache Hudi and Delta Lake table formats. Delta Lake supports Spark alternatives for accessing and querying the data, and of particular interest is the delta-rs library [31] that enables Python access to Delta Lake tables, without having the time overhead given by Spark jobs. However, the delta-rs [31] does not support Hadoop Distributed File System (HDFS), and consequently HopsFS, Hopsworks' own HDFS distribution [32].

1.2.1 Research Question

This research project has the ultimate objective to evaluate and compare the performance of Spark access to Delta Lake, to a delta-rs library [31] based access, for HopsFS [32]. To achieve this, support for HDFS must be added to the delta-rs library [31], so that it can be compatible with the Hopsworks system. Thus the project addresses the following two Research Questions (RQs):

RQ1: How can we add support for HDFS to the delta-rs library?

RQ2: What is the difference in performance between a Spark-based access to Delta Lake compared a the delta-rs library-based access, in HopsFS, Hopsworks' HDFS distribution?

1.2.2 Scientific and engineering issues

Delta-rs [31], as the name suggests, is a Rust [33] library, that offers Python bindings. Rust is a compiled language, and as such it does not need an interpreter like Python or a virtual environment like Java. This means that it is particularly easy to embed and use Rust code as a library in another language such as Python.

Currently, delta-rs does not support HDFS and therefore, HopsFS[32]. This means that adding HDFS support for delta-rs becomes a requirement of this project. Additionally, it should be noted that to match the dependencies used in the repository, the object_store [34] interface of Apache DataFusion [35] should be used.

INSERT ADDITIONAL COMMENTS ON SCIENTIFIC ISSUE ON THE EVALUATIONS METRICS

.

1.3 Purpose

The purpose of this thesis project is to contribute to reducing the read and write time, and thus increasing the data throughput, for operations on the Delta Lake-based Hopsworks Offline Feature Store. This work will identify which one between a Spark pipeline and a delta-rs pipeline on Delta Lake performs better, by comparing the differences in read time, write time, and computed throughput. As a prospect for future work, if delta-rs is proven to be a more performant alternative (in terms of data throughput), Hopsworks will consider integrating this pipeline into their application.

Overall implications for this thesis work are much wider counting the popularity Spark has in the open source community (more than 2800 contributors during its lifetime [36]. This would enable developers to have a wider range of alternatives when working on "small scale" (1 GB to 100 GB) systems by choosing delta-rs over Spark.

1.4 Goals

The accomplishment of the project's purpose (namely, increasing the data throughput for reading and writing on Delta Lake tables on the Hopsworks' HDFS distribution HopsFS) is bound to a list of Goals (Gs), here set. These are also related to the set of RQs, outlining a clear structure of the various project milestones.

1. Gs aimed to answer RQ1:

   G1: Understand delta-rs library [31] architecture and dependencies.

   G2: Identify what needs to be implemented to add HDFS support to the delta-rs library [31].

   G3: Implement HDFS support in the delta-rs library [31].

2. Gs aimed to answer RQ2:

   G4: Design and choose an evaluation framework to evaluate the different read and write performances of the new Rust pipeline based on the delta-rs library [31] and the old Apache Spark-based pipeline.

   G5: Perform the experiments using the designed framework to understand if and how the two pipelines work at different data loads (from 10 GB to 1 TB).

Associated with these Gs several Deliverables (Ds) will be created.

D1: Code implementation adding support to HDFS in the delta-rs library. This D is related to the completion of goals G1–G3. This deliverable also represents the system implementation contribution of the project.

D2: Experiment results on the performance evaluation of the new Rust pipeline based on the delta-rs library [31] compared to the old Apache Spark-based pipeline. This D is related to the completion of goals G4–G5.

D3: This thesis document, provides more detail on the implementation, design decisions, and expected performance.

1.5 Ethics and Sustainability

As a systems research project, the focus of this study revolves around software and in particular, developing more efficient data-intensive computing pipelines that find wide application in machine learning and training of neural networks. Software according to the Green Software Foundation [37] can be "part of the climate problem or part of the climate solution" [38]. We can define Green Software as a software that reduces its impact on the environment by using less physical resources, and less energy and optimizing energy use to use lower-carbon sources [38]. In the context of machine learning and training of neural networks, reducing training time (and so also the read and write time operation on the dataset) has been proven to positively impact the reduction in carbon emissions [39, 40].

This project, by aiming to increase the data throughput for reading and writing on Delta Lake tables on the Hopsworks' HDFS distribution HopsFS, follows the key green software principles reducing CPU time use compared to the previous Spark-based pipeline. This leads to a lower carbon footprint, as less energy is being used.

This project contributes to the Sustainable Development Goals (SDGs) 7 (Affordable and Clean Energy) and 9 (Industry Innovation and Infrastructure) [41], more specifically the targets 7.3 (Double the improvement in energy efficiency) and 9.4 (Upgrade all industries and infrastructures for sustainability). This work achieves this by reducing the read and write time of data on Delta Lake tables, and thus increasing the data throughput. This means that the same amount of data can be read or written in a smaller amount of time, reducing the use of resources (CPU or GPU computing time), thus reducing energy usage. This decrease in energy consumption will lead to a smaller carbon footprint (if the same amount of data is read or written).

Ultimately, this leads to an improvement in energy efficiency and a reduction in the carbon footprint of the data-intensive computing pipelines that find wide application in machine learning and training of neural networks.

1.6 Research Methodology

This work starts from a few Industrial Needs (INs), provided by Hopsworks, and a few Project Assumptions (PAs) validated through a literature study.
Hopsworks's INs are:

IN1 : the Hopsworks Feature Store has a lower throughput in reading and

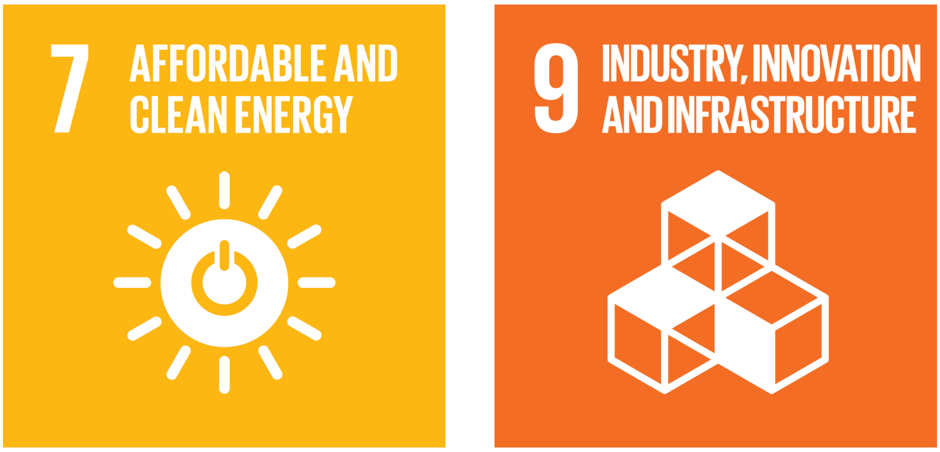

Figure 1.2: SDGs to which this thesis contributes to.

writing operations when performed on a "small scale" (1 GB - 100 GB) compared to a "large scale" (100 GB - 1 TB). This highlights the potential for improvement in the "small scale" use case.

IN2 : Hopsworks, adapting to their customer needs, supports the Delta Lake table format. Improving the speed of read and write operations on this table format, would improve a typical use case for Hopsworks Feature Store users.

PAs are:

PA1 : Python is the most popular programming language and the most used in data science workflows. ML and AI developers prefer Python tools to work. This means that Python libraries with high performance will typically be preferred over alternatives (even more efficient) that are Java Virtual Machine (JVM) or other environments based.

PA2 : Rust libraries have proven to have the chance to improve performance over C/C++ counterparts (Polars over Pandas). A Rust implementation could strongly improve reading and writing operations on the Hopsworks Feature Store.

These assumptions will be validated in the 2.

The project aims at fulfilling the INs with a system implementation approach. First, a HDFS storage support will be written for the delta-rs library to extend the Rust library support to HopsFS, Hopsworks HDFS distribution [32]. Then, an evaluation structure will be designed and used to compare the performances of the old Spark-based system and the new Rust-based pipeline. The two approaches will be tested with datasets of different sizes (between 1 GB and 1 TB). This is critical to identify if the same tool should be used for all scenarios or if they perform differently. The critical metrics that will be used to evaluate the system are read and write operations data throughout (the higher, the better). These were chosen as they most affect the computation time of pipelines accessing Delta Lake tables.

1.6.1 Delimitations

The project is conducted in collaboration with Hopsworks AB, and as such the implementation will focus on working with their HDFS distribution HopsFS. While the consideration drawn from these results cannot be generalized and be true for any system, they can still provide an insight on Apache Spark limitations, and on which tools perform better in different use cases.

1.7 Thesis Structure

Once the thesis is written, provide a outline of the thesis structure

Chapter 2

Background

This chapter provides the background information necessary to the reader to understand the project work. Starting from Delta Lake and Lakehouse paradigms, the chapter will highlight the trend in programming languages and distributed storage systems that helps to understand the relevance of this work. Furthermore, the chapter provides details on the Hopsworks Feature Store, and its architecture, complementing it with few reference to related work.

2.1 Delta Lake

In recent years the rise of Big Data, large volumes of various structured and unstructured data types at a high velocity, has showed an incredible potential but it has also posed a number of challenges [42]. These mostly impact the software architecture that needs to deal with these issues, that lead to an evolution of these technologies [43]. Delta Lake [5], is one of the most recent iteration of this evolution process, but in order to understand the tool, it is necessary to understand the challenges, starting from the beginning of the data management evolution.

Before Big Data, companies already wanted to gain insights from their data sources using an automated workflow. Here is where ETL and relational databases first came in use. An ETL pipeline as the name suggests:

1. Extracts data from APIs or other company's data sources.
2. Transforms data by removing errors or absent fields, standardises the format to match the database and validate the data to verify its correctness.

3. Loads it into a relational database (e.g. MySQL).

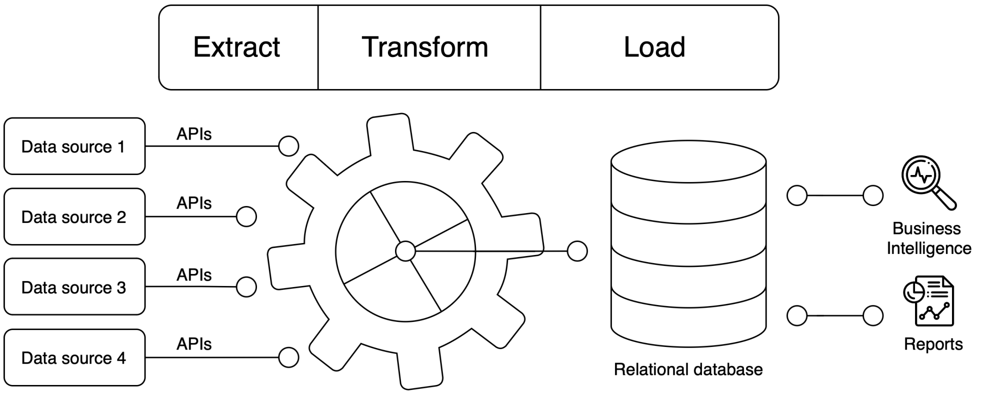

Figure 2.1: Simple ETL system with a relational database. Figure inspired by AltexSoft video [44]

This type of workflow worked for companies with no need of running complex analytical queries. This type of relational databases focusing on transactions are called On-Line Transaction Processing (OLTP) in contrast to OLAP systems. When the need to compute more complex queries rose, DBMS substituted simple database tables, optimized for running business centric complex analytical queries.

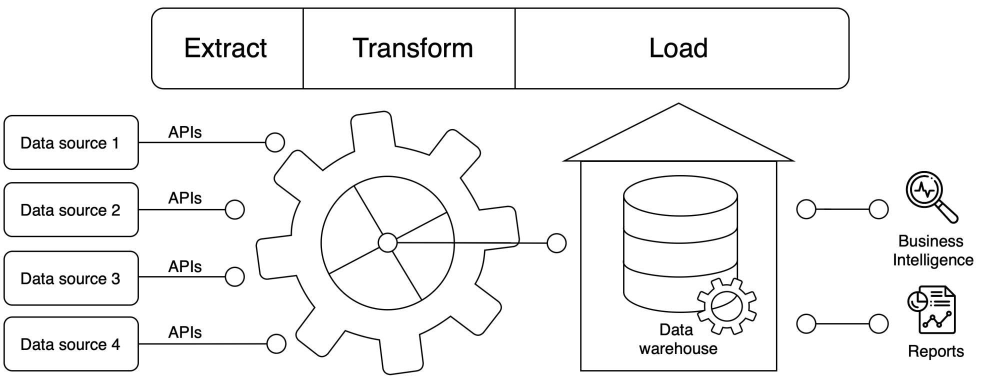

Figure 2.2: ETL system with a data warehouse. Figure inspired by AltexSoft video [44]

Here is where the first challenges caused by Big Data rose. DBMS only support structured data, while Big Data can be unstructured (e.g. images, videos). Furthermore, storing large DBMS systems is expensive and does not support any type of AI/ML workflow.

These issues were tackled by a new paradigm called Data Lake. Data Lakes are based on a low cost object storage system (see 2.3 to know more) that consist of a flat structure where all data is loaded after extraction. In Data lakes we have ELT pipelines that leave transformation customizable for specific applications. This architecture reduces storage costs, but the increase in complexity might make the architecture ultimately cost more depending on the case.

Explain the higher complexity of the new structure

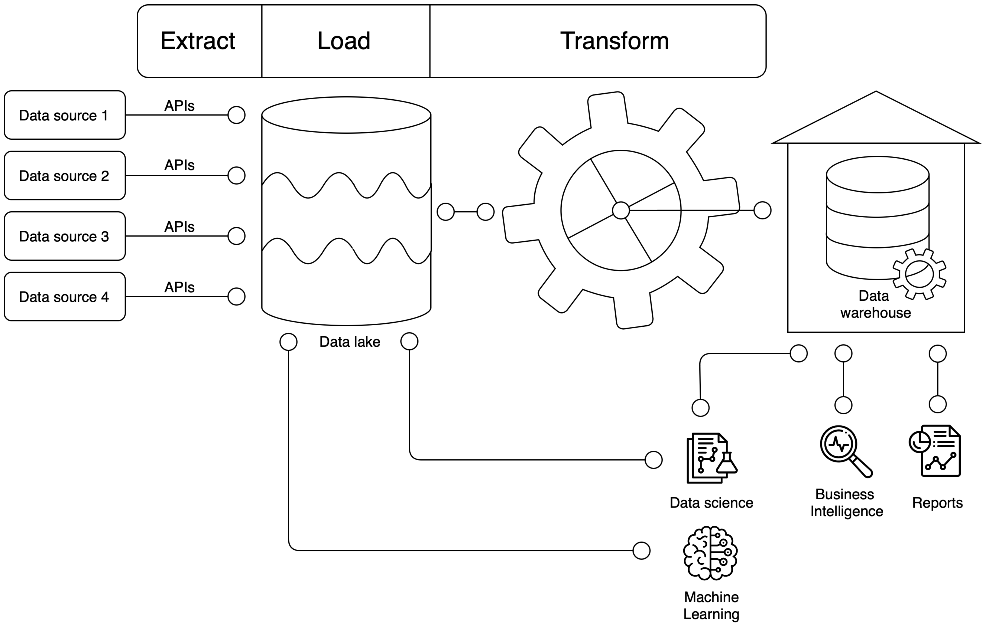

Figure 2.3: ELT system with a data lake. Figure inspired by AltexSoft video [44]

2.2 Programming languages

2.3 Modern distributed storage systems

2.4 Hopsworks Feature Store

2.5 Related Work

References

[1] "State of the Data Lakehouse," Dremio, Tech. Rep., 2024. [Page 1.]

[2] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proceedings of CIDR*, vol. 8, 2021. [Page 1.]

[3] D. Croci, "Data Lakehouse, beyond the hype," Dec. 2022. [Page 1.]

[4] "Apache Hudi vs Delta Lake vs Apache Iceberg - Data Lakehouse Feature Comparison," https://www.onehouse.ai/blog/apache-hudi-vs-delta-lake-vs-apache-iceberg-lakehouse-feature-comparison. [Page 1.]

[5] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. Van Hovell, A. Ionescu, A. Łuszczak, M. Świtakowski, M. Szafrański, X. Li, T. Ueshin, M. Mokhtar, P. Boncz, A. Ghodsi, S. Paranjpye, P. Senster, R. Xin, and M. Zaharia, "Delta lake: High-performance ACID table storage over cloud object stores," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, Aug. 2020. doi: 10.14778/3415478.3415560 [Pages 1, 3, and 11.]

[6] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016. doi: 10.1145/2934664 [Pages 1 and 4.]

[7] A. Khazanchi, "Faster reading with DuckDB and arrow flight on hopsworks : Benchmark and performance evaluation of offline feature stores," Master's thesis, KTH Royal Institute of Technology / KTH, School of Electrical Engineering and Computer Science (EECS) / KTH, School of Electrical Engineering and Computer Science (EECS), 2023. [Page 1.]

[8] M. Raasveldt and H. Mühleisen, "DuckDB: An Embeddable Analytical Database," in *Proceedings of the 2019 International Conference on Management of Data*. Amsterdam Netherlands: ACM, Jun. 2019. doi: 10.1145/3299869.3320212. ISBN 978-1-4503-5643-5 pp. 1981–1984. [Pages 2 and 4.]

[9] R. Vink, "I wrote one of the fastest DataFrame libraries," https://www.ritchievink.com/blog/2021/02/28/i-wrote-one-of-the-fastest-dataframe-libraries/, Feb. 2021. [Pages 2 and 4.]

[10] "Benchmark Results for Spark, Dask, DuckDB, and Polars — TPC-H Benchmarks at Scale," https://tpch.coiled.io/. [Page 2.]

[11] T. Ebergen, "Updates to the H2O.ai db-benchmark!" https://duckdb.org/2023/11/03/db-benchmark-update.html, Nov. 2023. [Page 2.]

[12] A. Nagpal and G. Gabrani, "Python for Data Analytics, Scientific and Technical Applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Feb. 2019. doi: 10.1109/AICAI.2019.8701341 pp. 140–145. [Page 2.]

[13] "TIOBE Index," https://www.tiobe.com/tiobe-index/. [Pages 2 and 5.]

[14] "Stack Overflow Developer Survey 2023," https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023. [Page 2.]

[15] M. Raschka and S. Vahid, *Python Machine Learning (3rd Edition)*. Packt Publishing, 2019. ISBN 978-1-78995-575-0 [Page 2.]

[16] "Hopsworks - Batch and Real-time ML Platform," https://www.hopsworks.ai/, 2024. [Pages 2 and 5.]

[17] A. Pettersson, "Resource-efficient and fast Point-in-Time joins for Apache Spark : Optimization of time travel operations for the creation of machine learning training datasets," Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS) / KTH, School of Electrical Engineering and Computer Science (EECS), 2022. [Page 2.]

[18] "What Is a Lakehouse?" https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html, Thu, 01/30/2020 - 09:00. [Page 3.]

[19] S. Chaudhuri and U. Dayal, “An overview of data warehousing and OLAP technology,” *ACM SIGMOD Record*, vol. 26, no. 1, pp. 65–74, Mar. 1997. doi: 10.1145/248603.248616 [Page 3.]

[20] EDER, “Unstructured Data and the 80 Percent Rule,” Aug. 2008. [Page 3.]

[21] “Dremel made simple with Parquet,” https://blog.x.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet. [Page 3.]

[22] P. Rajaperumal, “Uber Engineering’s Incremental Processing Framework on Hadoop,” https://www.uber.com/blog/hoodie/, Mar. 2017. [Page 3.]

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud’10. USA: USENIX Association, 2010, p. 10. [Page 3.]

[24] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *OSDI’04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004, pp. 137–150. [Page 3.]

[25] T. T. E. White, “Hadoop : The definitive guide,” in *Hadoop : The Definitive Guide*, first edition. ed. Sebastopol, California: O’Reilly Media, Inc., 2009. ISBN 1-306-81746-3 [Page 3.]

[26] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-82, Jul. 2011. [Page 3.]

[27] “Apache Spark™ - Unified Engine for large-scale data analytics,” https://spark.apache.org/. [Page 3.]

[28] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache Flink™: Stream and Batch Processing in a Single Engine.” [Page 4.]

[29] G. van Rossum, "Python tutorial," Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Tech. Rep. CS-R9526, May 1995. [Page 4.]

[30] "TIOBE Index," https://www.tiobe.com/tiobe-index/programminglanguages_definition/. [Page 5.]

[31] "Delta-io/delta-rs," Delta Lake, May 2024. [Pages 5, 6, and 7.]

[32] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt, and M. Ronström, "HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017. ISBN 978-1-931971-36-2 pp. 89–104. [Pages 5, 6, and 9.]

[33] "Rust Programming Language," https://www.rust-lang.org/. [Page 6.]

[34] "Object_store - Rust," https://docs.rs/object_store/latest/object_store/. [Page 6.]

[35] "Apache DataFusion — Apache DataFusion documentation," https://datafusion.apache.org/. [Page 6.]

[36] "The Apache Spark Open Source Project on Open Hub," https://openhub.net/p/apache-spark. [Page 7.]

[37] "Green Software Foundation," https://greensoftware.foundation/. [Page 8.]

[38] "What is Green Software?" https://greensoftware.foundation/articles/what-is-green-software, Oct. 2021. [Page 8.]

[39] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon Emissions and Large Neural Network Training," 2021. [Page 8.]

[40] D. Patterson, J. Gonzalez, U. Hölzle, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink," 2022. [Page 8.]

[41] "| Sustainable Development," https://sdgs.un.org/. [Page 8.]

[42] H. E. Pence, "What is Big Data and Why is it Important?" *Journal of Educational Technology Systems*, vol. 43, no. 2, pp. 159–171, Dec. 2014. doi: 10.2190/ET.43.2.d [Page 11.]

[43] I. Gorton and J. Klein, "Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems," *IEEE Software*, vol. 32, no. 3, pp. 78–85, May 2015. doi: 10.1109/MS.2014.51 [Page 11.]

[44] AltexSoft, "How Data Engineering Works - Youtube," https://www.youtube.com/watch?v=qWru-b6m030&t=485s, 2021. [Pages 12 and 13.]