KTH ROYAL INSTITUTE
OF TECHNOLOGY

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

Stockholm, Sweden, 2024

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: May 29, 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
BI	Business Intelligence
ELT	Extract Load Transform
ETL	Extract Transform Load
HDFS	Hadoop Distributed File System
IN	Industrial Need
JVM	Java Virtual Machine
ML	Machine Learning
OLAP	On-Line Analytical Processing
PA	Project Assumption
RDD	Resilient Distributed Dataset

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 [2] is preferred over old paradigms, i.e. data warehouses and data lakes, because it takes the best of both worlds, having scalability properties of data lakes, while preserving the Atomicity, Consistency, Isolation and Durability (ACID) properties typical of data warehouses. Additionally, Lakehouse systems include partitioning, that reduces query significantly and time travel enabling users to access different versions of data, versioned over time [3].

Three 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 all three projects are backed by large communities, it is Delta Lake that is more acknowledged as a Lakehouse solution [4]. This is mainly thanks to Databricks, that first promoted this new architecture over data lakes among their clients around 2020 [5].

Delta Lake is typically used in combination with Apache Spark [6] that acts as a data query and processing engine. This approach is effective when processing large quantities of data (1 TB or more) over the cloud, but is this approach still effective in other scenarios such as local computing over small quantities of data (100 GB or less)?

In recent years, in other data storage cases, DuckDB [7] and Polars [8], showed as under a certain number of data volume, and in particular if the architecture is local, starting a Spark cluster, might actually reduce performance, increasing costs and the computation time. Spark alternatives at a smaller scale (10 GB - 100 GB) generally perform much better in these scenarios [9, 10].

Another aspect to keep in mind when developing in this field is which programming language data scientists like to use, and this is Python. Python is currently the most popular programming language [11] and it is by far the most used language for Machine Learning (ML) and Artificial Intelligence (AI) applications [12], this is mainly thanks to its strong abstraction capabilities and accessibility. This can be also observed by looking at the mentioned libraries, DuckDB, Polars and Spark, that all offer Python interfaces. In this scenario, creating a Python client for Delta Lake would be beneficial as it would not have to resort to Spark and its Python library (PySpark). This approach with small-scale 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 reusable features [13]. This architecture also supports point-in-time correct datasets from historical feature data [14].

This project here presented, aims to speed up their read and write operations on the Feature Store, currently Spark-based. Ultimately, this system implementation will become part for their Feature Store product (open source version), greatly improving the experience of Python users working on small quantities of data (between 10 GB and 100 GB).

1.1 Background

A cleared understanding of the background of this project comes from appreciating three different key aspects: Lakehouse development, Spark relevance and flows, Python as emergent language.

Lakehouse is a term coined by Databricks in 2020 [2], to define a new design standard that was emerging in the industry, that combined the capability of data lakes of storing and managing unstructured data, with the ACID properties typical of Data warehouses. Data warehouses became a dominant standard in the 90s early 2000s, enabling companies to generate Business Intelligence (BI) insights, managing different structured data sources. The problems related to this architecture rose in the 2010 years when it became clear the need to manage unstructured data in large quantities [15]. 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 a lot of Extract Load Transform (ELT) and Extract Transform Load (ETL) computations. Lakehouses solved the problems of Data lakes by implementing data management and performance features on top of open data formats such as Parquet [16]. This paradigm was successful thanks to three key components: a metadata layer for data lakes, a new query engine design, a declarative access for ML and AI. This architecture design was first open-sourced with Apache Hudi in 2017 [17] and then Delta Lake in 2020 [5].

When talking about Spark origins and reasons behind its creation we need to look into Google needs for advancing in the internet search and indexing. In particular these needs led to the creation of MapReduce [18] a distributed programming model that enables the management of large datasets. This paradigm became then part of the Hadoop ecosystem [19]. From the roots of MapReduce, Spark was created [6], improving both on the performance (10 times better in its first iteration), and on the fault tolerance, using Resilient Distributed Datasets (RDDs). RDDs are a distributed memory abstraction that enables a lazy in-memory computation that is tracked though the use of lineage graphs, ultimately increasing fault tolerance [20]. Spark, now Apache Spark under the Apache foundation [21], has seen widespread success and adoption in various applications, becoming a de-facto standard of the distributed computing world. As Spark becomes older, other approaches appear and compete with Spark in specific areas, achieving better results. This is the case of Apache Flink [22], designed for true stream processing prevails over Spark in this area. The same can be said for lower scale applications where the high scaling capabilities of Spark are not at use, and the overhead of starting a Spark application is not compensated by other factors. This is the case of DuckDB [7] and Polars [8], that focusing on low scale (10GB-100GB) they provided 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.

The data science world speaks Python [23]. Python was first adopted by many thanks to its focus on ease of use, high abstraction level and readability.

This helped created a fast-growing community behind the project, that lead to the development of a great number of libraries and APIs. So now, after more than 30 years after its creation, it became the de-facto standard for data science thanks to its many libraries such as Tensorflow, NumPy, SciPy, Pandas, PyTorch, Keras and many others.

Python is also the most popular programming language. This appears clear if we refer to TIOBE Index 2024 [11] we see that Python has a rating of 15.16%, followed by C that 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, that were both outranked by Python between 2021 and 2022. This shows the importance of offering Python interfaces for programmers and data scientist in particular to increase the engagement and possibilities of a framework.

1.2 Problem

Hopsworks Feature Store [13], currently has a large time overhead when starting Spark (around 5 minutes) even for the most simple operation. 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 larger scale, but still relatively small (scale between 10-100 GBs). As this overhead is caused by starting up Spark jobs, we 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, of particular interest is the library delta-rs [24] that enables Python access to Delta Lake tables, without having the time overhead given by Spark jobs. However, the delta-rs [24] does not support Hadoop Distributed File System (HDFS), thus not even HopsFS, Hopsworks own HDFS distribution [25].

1.2.1 Research Question

The objective of this systems research project is designing and implementing using delta-rs [24] a Python access to Delta Lake tables for the Hopsworks Feature Store. Additionally, this research aims to evaluate and compare the current system based on Apache Spark and the newly implemented one based on the delta-rs library [24]. Hence, the research question is the following: "What is the difference in performance between a Apache Spark based access to Delta Lake compared to an access based on the delta-rs library?

1.2.2 Scientific and engineering issues

Delta-rs [24] as the name suggests is a Rust [26] library, that offers Python bindings. Rust is a compiled language, and as such it does not need an interpreter as Python or virtual environment as Java. This means that it is particularly easy to embed and use Rust code as library in another language such as Python.

Currently delta-rs does not support HDFS (or the Hopsworks distribution, HopsFS [25]). This means that adding support to HDFS to delta-rs becomes a requirement of this project. Additionally, it should be noted that to meet development standards to the repository, the object_store [27] interface of the Datafusion [28] should be used.

INSERT ADDITIONAL COMMENTS ON SCIENTIFIC ISSUE ON THE EVALUATIONS METRICS

.

1.3 Purpose

The purpose of the project is to reduce the read and write time for operations on the Hopsworks Feature Store built on top of Delta Lake tables. The main use case in consideration is a small scale amount of data (10 GB - 100 GB), as Hopsworks users mostly work with this amounts of data, when using the platform. Thanks to this improvement, developers working with the Feature Store will see an improvement in productivity and reduce friction with the development tool.

While managers of other professions typically focus on employee productivity, in software engineering it a good approach is to focus on limiting friction between developers and tools, so that they can focus on value creation. For such effective environment to grow a micro-feedback loops structure is created, enabling developers to work with continuous feedback [29]. Faster read and write operations on the Feature Store would enable the creations of such feedback loops.

On top of reducing work time for a developer, an efficient working environment can be a positive attractor [30] incrementing the happiness and thus performance of the developer.

1.4 Goals

The goal of this systems research project is the to design and implement a Delta Lake read and write Rust interface that has faster read and write operations compared to the current Spark-based one. This goal has been divided into the following three sub-goals:

1. Implement the object_store interface for HDFS, thus adding support for Hopsworks storage solution.
2. Add support for the HDFS object_store in the delta-rs library, creating a working pipeline to read and write on Delta Lake tables on top of which Hopsworks Feature Store operates.
3. Compare the new Rust pipeline to the old Spark-based one at different data loads (from 10 GB to 1 TB) using read and write operation time as primary metric.

1.5 Ethics and Sustainability

As a systems research project the focus of this study revolves around software. Software according to the Green Software Foundation [31] can be "part of the climate problem or part of the climate solution" [32]. We can define Green Software as a software that reduces its impact on the environment by using less physical resources, less energy and optimizing energy use to use lower-carbon sources [32].

This project, by reducing the time required for reading and writing on Delta Lake tables in the Hopsworks Feature Store, 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.

1.6 Research Methodology

This work starts from few Industrial Needs (INs), provided by Hopsworks, and a few Project Assumptions (PAs) validated though a literature study.

Hopsworks's INs are:

IN1 : the Hopsworks Feature Store currently suffers from slow read and write operations on their platform. Reducing the overhead to start the Spark jobs could streamline the computation by a large factor.

IN2 : Hopsworks, adapting to their customer needs, supports 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 be 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 operation on the Hopsworks Feature Store.

These assumptions will be validated in the 2.

The project aims at fulfilling the INs with an 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 [25]. Then, once a working implementation to read and write on Hopsworks Feature Store though delta-rs is in place, an evaluation framework 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 size (between 1 GB and 1 TB). The critical metrics that will be used to evaluate the system are read and write operations time (the lower, the better).

1.7 Delimitations

The project is conducted in collaboration with Hopsworks AB, and as such the implementation will focus on working with their Feature Store. 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.

1.8 Structure of the thesis

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

Chapter 2

Background

Bakgrund

When you do your literature study, you should have a nearly complete Chapters 1 and 2.
You may also find it convenient to introduce the future work section into your report early – so that you can put things that you think about but decide not to do now into this section.
Note that later you can move things between this future work section and what you have done as you may change your mind about what to do now versus what to put off to future work.

What does a reader (another x student – where x is your study line) need to know to understand your report? What have others already done? (This is the "related work".) Explain what and how prior work/prior research will be applied on or used in the degree project/work (described in this thesis). Explain why and what is not used in the degree project and give valid reasons for rejecting the work/research.

This chapter provides basic background information about xxx. Additionally, this chapter describes xxx. The chapter also describes related work xxxx.

Vilken viktig litteratur och (forsknings-)artiklar har du studerat inom området (litteraturstudie)?

2.1 Major background area 1

Viktigt bakgrundsområde 1

There are xxx characteristics that distinguish yyy from other information and communication technology (ICT) system, as shown in Figure 2.1. Table 2.1 summarizes these characteristics.

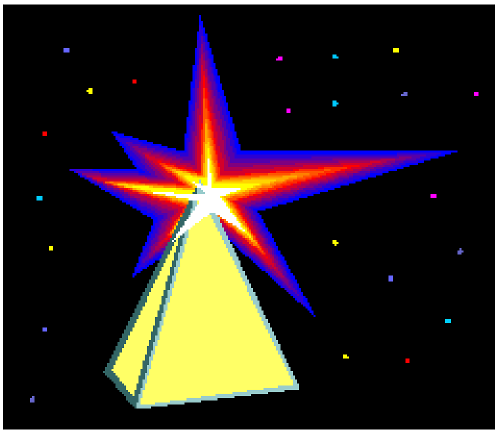

Figure 2.1: Lots of stars (Inspired by Figure x.y on page z of [xxx])

Massor av stärnor (Inspirerad av figur x.y på sidan z i [xxx])

Table 2.1: xxx characteristics

Characteristics α	**Description** β
1	1 110.1
2	10.1
3	23.113 231

Egenskaper

Beskrivning

2.1.1 Subarea 1.1

Entangled states are an important part of quantum cryptography, but also relevant in other domains. This concept might be relevant for neutrinos, see for example.

2.1.2 Subarea 1.1.2

Computational methods are increasingly used as a third method of carrying out scientific investigations. For example, computational experiments were used to find the amount of wear in a polyethylene liner of a hip prosthesis in. . . .

2.1.3 Subarea 1.1.2

Using the nearest data center may improve performance

2.1.4 Link layer Encapsulation

See Figure 2.2 which uses the bytefield LaTeX package.

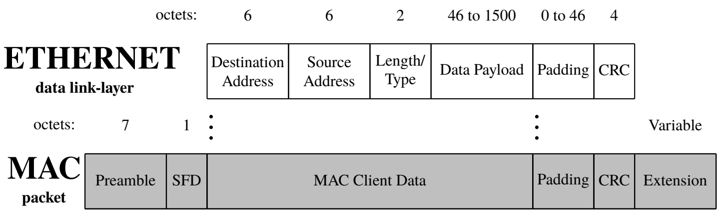

Figure 2.2: Ethernet data link layer protocol encapsulated into a IEEE 802.3 MAC packet

2.1.5 IP packet headers

The data link layer will receive a packet from the IP layer. The layout of an IPv4 packet is shown in Figure 2.3. This should be contrasted with the IPv6 header shown in Figure 2.4.

2.1.6 Test for accessibility of formulas

As can be seen in these equations: $c = 2 \cdot \pi \cdot r$ or

$$\int_a^b x^2 \, dx$$

a chemical formula: $(C_5O_2H_8)_n$ ...

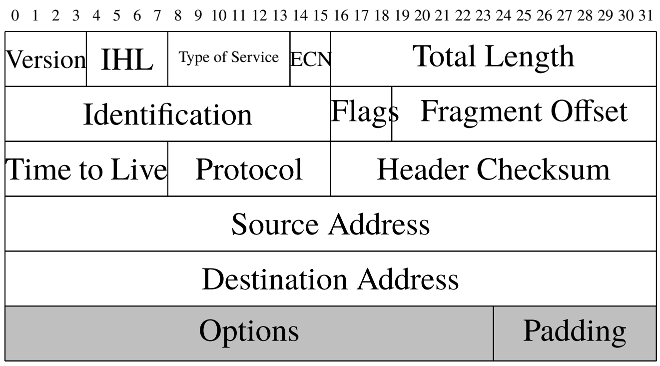

Figure 2.3: IPv4 datagram header. Light grey coloured fields are optional.

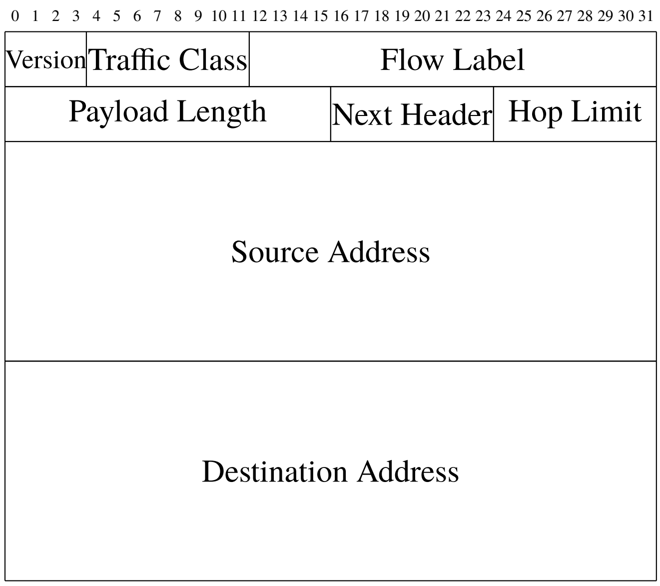

Figure 2.4: IPv6 datagram header

2.2 Major background area 2

Viktigt bakgrundsområde 2

...

2.2.1 WLAN Security

2.2.2 Network layer security

...

2.3 Related work area

Relaterade arbeten

2.3.1 Major related work 1

Relaterade arbeten 1

Carrier clouds have been suggested as a way to reduce the delay between the users and the cloud server that is providing them with content. However, there is a question of how to find the available resources in such a carrier cloud. One approach has been to disseminate resource information using an extension to OSPF-TE, see Roozbeh, Sefidcon, and Maguire.

2.3.2 Major related work n

Relaterade arbeten

2.3.3 Minor related work 1

Mindre relaterat arbete 1

...

2.3.4 Minor related work n

Mindre relaterat arbete n

2.4 Summary

Sammanfattning

Det är trevligt om detta kapitel avslutas med en sammanfattning. Till exempel kan du inkludera en tabell som sammanfattar andras idéer och fördelar och nackdelar med varje - så som senare kan du jämföra din lösning till var och en av dessa. Detta kommer också att hjälpa dig att definiera de variabler som du kommer att använda för din utvärdering.

It is nice to have this chapter conclude with a summary. For example, you can include a table that summarizes other people's ideas and benefits and drawbacks with each - so as later you can compare your solution to each of them. This will also help you define the variables that you will use for your evaluation.

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. [Pages 1 and 2.]

[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 and 3.]

[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 3.]

[7] 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 3.]

[8] 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 3.]

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

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

[11] “TIOBE Index,” https://www.tiobe.com/tiobe-index/. [Pages 2 and 4.]

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

[13] “Hopsworks - Batch and Real-time ML Platform,” https://www.hopsworks.ai/, 2024. [Pages 2 and 4.]

[14] 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.]

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

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

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

[18] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. doi: 10.1145/1327452.1327492 [Page 3.]

[19] “Apache Hadoop,” https://hadoop.apache.org/. [Page 3.]

[20] 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.]

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

[22] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink™: Stream and Batch Processing in a Single Engine." [Page 3.]

[23] G. van Rossum, "Python tutorial," Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Tech. Rep. CS-R9526, May 1995. [Page 3.]

[24] "Delta-io/delta-rs," Delta Lake, May 2024. [Pages 4 and 5.]

[25] 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 4, 5, and 7.]

[26] "Rust Programming Language," https://www.rust-lang.org/. [Page 5.]

[27] "Object_store - Rust," https://docs.rs/object_store/latest/object_store/. [Page 5.]

[28] "Apache DataFusion — Apache DataFusion documentation," https://datafusion.apache.org/. [Page 5.]

[29] T. Cochran, "Maximizing Developer Effectiveness," https://martinfowler.com/articles/developer-effectiveness.html, Jan. 2021. [Page 5.]

[30] D. Graziotin, "Towards a Theory of Affect and Software Developers' Performance," Jan. 2016. [Page 6.]

[31] "Green Software Foundation," https://greensoftware.foundation/. [Page 6.]

[32] "What is Green Software?" https://greensoftware.foundation/articles/what-is-green-software, Oct. 2021. [Page 6.]