usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Time Travel and Provenance for Machine Learning Pipelines

Alexandru A. Ormenisan, *KTH - Royal Institute of Technology and Logical Clocks AB;* Moritz Meister, Fabio Buso, and Robin Andersson, *Logical Clocks AB;* Seif Haridi, *KTH - Royal Institute of Technology;* Jim Dowling, *KTH - Royal Institute of Technology and Logical Clocks AB*

https://www.usenix.org/conference/opml20/presentation/ormenisan

This paper is included in the Proceedings of the
2020 USENIX Conference on Operational Machine Learning.
July 28–August 7, 2020
978-1-939133-15-1

Open access to the Proceedings of the 2020 USENIX Conference on Operational Machine Learning is possible thanks to the generous support of

NetApp®

Time Travel and Provenance for Machine Learning Pipelines

Alexandru A. Ormenisan
KTH - Royal Institute of Technology
Logical Clocks AB

Moritz Meister
Logical Clocks AB

Fabio Buso
Logical Clocks AB

Robin Andersson
Logical Clocks AB

Seif Haridi
KTH - Royal Institute of Technology

Jim Dowling
KTH - Royal Institute of Technology
Logical Clocks AB

Abstract

Machine learning pipelines have become the defacto paradigm for productionizing machine learning applications as they clearly abstract the processing steps involved in transforming raw data into engineered features that are then used to train models. In this paper, we use a bottom-up method for capturing provenance information regarding the processing steps and artifacts produced in ML pipelines. Our approach is based on replacing traditional intrusive hooks in application code (to capture ML pipeline events) with standardized change-data-capture support in the systems involved in ML pipelines: the distributed file system, feature store, resource manager, and applications themselves. In particular, we leverage data versioning and time-travel capabilities in our feature store to show how provenance can enable model reproducibility and debugging.

1 From Data Parallel to Stateful ML Pipelines

Bulk synchronous parallel processing frameworks, such as Apache Spark [4], are used to build data processing pipelines that use idempotence to enable transparently handling of failures by re-executing failed tasks. As data pipelines are typically stateless, they need lineage support to identify those stages that need to be recomputed when a failure occurs. Caching temporary results at stages means recovery can be optimized to only re-run pipelines from the most recent cached stage. In contrast, database technology uses stateful protocols (like 2-phase commit and agreement protocols like Paxos [10]) to provide ACID properties to build reliable data processing systems. Recently, new data parallel processing frameworks have been extended with the ability to make atomic updates to tabular data stored in columnar file formats (like Parquet [3]) while providing isolation guarantees for concurrent clients. Examples of such frameworks are Delta Lake [8], Apache Hudi [1], and Apache Iceberg [2]. These ACID data lake platforms are important for ML pipelines as they provide the ability to query the value of rows at specific points in time in the past (time-travel queries).

The Hopsworks Feature Store is built on the Hudi framework, where data files are stored in HopsFS [11] as parquet files and available as external tables in a modified version of Apache Hive [6] that shares the same metadata layer as HopsFS. HopsFS and Hive have a unified metadata layer, where Hive tables and feature store metadata are extended metadata for HopsFS directories. Foreign keys and transactions in our metadata layer ensure the consistency of extended metadata through Change-Data-Capture(CDC) events.

Just like data pipelines, ML pipelines should be able to handle partial failures, but they should also be able to reproducibly train a model even if there are updates to the data lake. The Hopsworks feature store with Hudi enables this, by storing both the features used to train a model and the Hudi commits (updates) for the feature data, see figure 1.

In contrast to data pipelines, ML pipelines are stateful. This state is maintained in the metadata store through a series of CDC events as can be seen in figure 1. For example, after a model has been trained and validated, we need state (from the metadata store) to check if the new model has better performance than an existing model running in production. Other systems like TFX [5] and MLFlow [14] also provide a metadata store to enable ML pipelines to make stateful decisions. However, they do so in an obtrusive way - they make developers re-write the code at each of the stages with their specific component models. In Hopsworks [9], however, we provide an unobtrusive metadata model based on implicit provenance [12], where change capture APIs in the platform enable metadata about artifacts and executions to be implicitly saved in a metadata store with minimal changes needed to user code that makes up the ML pipeline stages.

2 Versioning Code, Data, and Infrastructure

The defacto approach for versioning code is git, and many solutions are trying to apply the same process to the versioning of data. Tools, such as DVC [7] and Pachyderm [13] version data with git-like semantics and track immutable files, instead of changes to files. An alternative to git-like versioning that we chose is to use an *ACID Data-Lake* with time-travel query

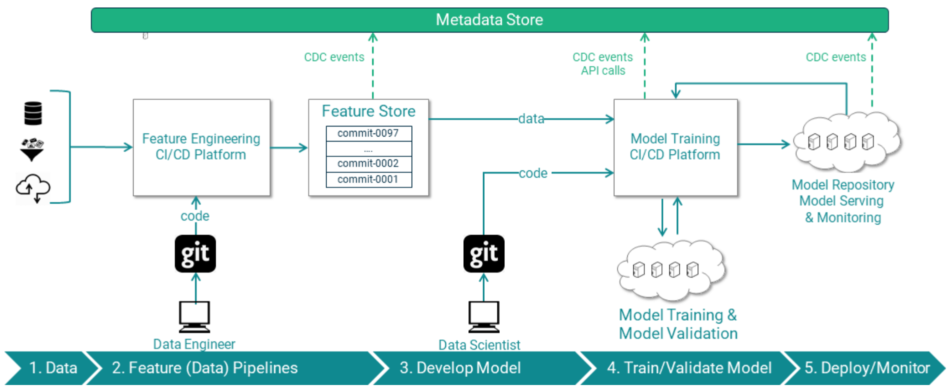

Figure 1: Hopsworks ML Pipelines with Implicit Metadata.

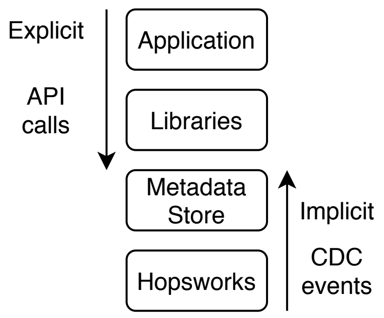

Figure 2: Implicit vs Explicit Provenance.

capabilities. Recent platforms such as Delta-Lake, Apache Hudi, Apache Iceberg extend data lakes with ACID guarantees using Apache Spark to perform updates and queries. These platforms store version and temporal information about updates to tables in a commit log (containing the row-level changes to tables). As such, you can issue time-travel queries such as "what was the value of this feature on this date-time", or "select this feature data for the time range 2017-2018". These type of queries are traditionally not possible in data warehouses that typically only store the latest values for rows in tables. Efficient time travel queries are enabled by temporal indexes (bloom filters in Hudi) over the parquet files that store the commits (updates to tables). Aside from code and data versioning, we also consider the versioning of infrastructure. With much of the ML work being done in Python, it is crucial to know the Python libraries and their versions when you ran a pipeline to facilitate importing/exporting and re-executing the pipeline, reproducibly.

3 Provenance in Hopsworks

TFX and MLFLow track provenance, by asking the user to explicitly mark the operation on a ML artifact that should be saved in their separate metadata store. We call this approach explicit, as the user has to explicitly say what would be tracked though calls to dedicated APIs. In Hopsworks, we instrument our distributed file systems - HopsFS, our resource manager - HopsYarn and our feature store to implicitly track the file operations, including the tagging of files with additional information. Our metadata store is also tightly coupled with the file system and thus we can capture metadata store operations in the same style. As we can see in figure 2, implicit provenance [12] relies on bottom-up propagation of Change-Data-Capture(CDC) events, while explicit provenance relies on user invocation of specific API and has a up-down propagation. With implicit provenance we can track which application used or created a specific file and who is the owner of the application. This together with file naming conventions and tagging of files allows us to automatically create relations between files on disk representing machine learning artifacts.

4 Reproducible ML Pipelines

With versioned code, data and environments, as well as provenance information to mark the usages of the particular version, it is easy to reproduce executions of the same pipeline. With the help of provenance information we augment this reproducibility scenario with additional functionality such as determining whether your current setup will provide similar results to the original and warn the user in case the input datasets, the environment or the code differ to the original. We also provide an environment that encourages exploration and learning through the ability to search through full text elasticsearch queries for most used datasets, most recent pipeline using a specific dataset or the persons who ran the latest version of a particular pipeline successfully. Provenance can also be used to warn a user not to expect a similar result to previous runs due to changes in the code, data or environment.

5 Provenance for Debugging ML Pipelines

Implicit provenance provides us with links between ML artifacts used and created in each of the stages of the pipeline. We know at what time they were used, by whom and in what application. This allows us to determine the footprint of each stage of the pipeline, as the files that were read, modified, created or changed in any way during the execution of the pipeline stages. From the footprint of each of the pipeline stages we can also determine the impact of previous stages. We defined the impact of a pipeline stage as the union of footprints for pipeline stages that use the output of the current stage as input. Since artifacts can be shared across pipelines, and since pipelines can be partially re-executed and forked into new pipelines, the impact of a pipeline can be quite large.

6 Summary

In this paper, we discussed how the implicit model for provenance can be used next to a feature store with versioned data to build reproducible and more easily debugged ML pipelines. Our future work will provide development tools and visualization support that can help developers more easily navigate and re-run pipelines based on the architecture we have built.

Acknowledgments

This work was funded by the EU Horizon 2020 project Human Exposome Assessment Platform (HEAP) under Grant Agreement no. 874662.

References

[1] Apache Hudi. https://hudi.apache.org/. [Online; accessed 25-February-2020].

[2] Apache Iceberg. https://iceberg.apache.org/. [Online; accessed 25-February-2020].

[3] Apache Parquet. https://parquet.apache.org/. [Online; accessed 25-February-2020].

[4] Apache Spark. https://spark.apache.org/. [Online; accessed 25-February-2020].

[5] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395. ACM, 2017.

[6] Fabio Buso. Sql on hops. Master's thesis, KTH, School of Information and Communication Technology (ICT), 2017.

[7] Data Version Control. https://dvc.org/. [Online; accessed 25-February-2020].

[8] Delta Lake. https://delta.io/. [Online; accessed 25-February-2020].

[9] Mahmoud Ismail, Ermias Gebremeskel, Theofilos Kakantousis, Gautier Berthou, and Jim Dowling. Hopsworks: Improving user experience and development on hadoop with scalable, strongly consistent metadata. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2525–2528. IEEE, 2017.

[10] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[11] Salman Niazi, Mahmoud Ismail, Seif Haridi, Jim Dowling, Steffen Grohsschmiedt, and Mikael Ronström. Hopsfs: Scaling hierarchical file system metadata using newsql databases. In *15th {USENIX} Conference on File and Storage Technologies ({FAST} 17)*, pages 89–104, 2017.

[12] Alexandru A. Ormenisan, Mahmoud Ismail, Seif Haridi, and Jim Dowling. Implicit provenance for machine learning artifacts. In *Proceedings of MLSys'20: Third Conference on Machine Learning and Systems.*, 2020.

[13] Pachyderm. https://www.pachyderm.com/. [Online; accessed 25-February-2020].

[14] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.