Implicit Provenance for Machine Learning Artifacts

Alexandru A. Ormenisan
aaor@kth.se
KTH - Royal Institute of Technology
Logical Clocks
Stockholm, Sweden

Mahmoud Ismail
maism@kth.se
KTH - Royal Institute of Technology
Logical Clocks
Stockholm, Sweden

Seif Haridi
haridi@kth.se
KTH - Royal Institute of Technology
Stockholm, Sweden

Jim Dowling
jdowling@kth.se
KTH - Royal Institute of Technology
Logical Clocks
Stockholm, Sweden

ABSTRACT

Machine learning (ML) presents new challenges for reproducible software engineering, as the artifacts required for repeatably training models are not just versioned code, but also hyperparameters, code dependencies, and the exact version of the training data. Existing systems for tracking the lineage of ML artifacts, such as TensorFlow Extended or MLFlow, are invasive, requiring developers to refactor their code that now is controlled by the external system. In this paper, we present an alternative approach, we call implicit provenance, where we instrument a distributed file system and APIs to capture changes to ML artifacts, that, along with file naming conventions, mean that full lineage can be tracked for TensorFlow/Keras/Pytorch programs without requiring code changes. We address challenges related to adding strongly consistent metadata extensions to the distributed file system, while minimizing provenance overhead, and ensuring transparent eventual consistent replication of extended metadata to an efficient search engine, Elasticsearch. Our provenance framework is integrated into the open-source Hopsworks framework, and used in production to enable full provenance for end-to-end machine learning pipelines.

1 INTRODUCTION

Modern software engineering methodologies, such as agile and test-driven development, strive to continuously deliver new versions of stable working systems by automated building and testing of incremental changes to software. Machine learning (ML) is a relatively new software engineering discipline, where we strive to continuously deliver new versions of models, and, in the event of performance, security, or behavioural regressions of a model, be able to discern the source of those regressions by investigating all parts of the model's lineage from the code, its dependencies, and the features used to train the model. In addition, the provenance information for all ML artifacts used to train models is important for model governance, interpretability, debugging, and sharing of artifacts between teams.

Machine learning applications run as pipelines that ingest data from a source, such as a data lake, and compute features, discover hyperparameters, train model(s), validate and deploy model(s). Pipelines typically run on workflow engines, such as AirFlow [1], TensorFlow Extended [3] , MLFlow [14], KubeFlow [8], and Argo [2].

2 PROVENANCE TRANSPARENCY

Existing approaches to model provenance and governance involve a generic API-based instrumentation approach, where each step in a workflow is logged in a database to act as a recorded history of the execution. On one hand this approach allows for a very flexible use, where the user can determine exactly what she wants to log. On the other hand, it is prone to users omitting to log certain artifacts, only logging use of artifacts in one stage of the pipeline but not another or having parts of the code that need to be changed when they want to switch to a different provenance tracking system. We call, this type of provenance tracking, explicit provenance, since the users have to explicitly point what to log in the provenance. In our platform, we decided to track provenance at a lower level, at the platform layer, in a transparent and non-invasive way towards the user's machine learning code. We call this provenance tracking method implicit provenance, because we log the operations at the filesystem and resource manager level, and thus we implicitly gain the provenance information without interfering with the ML code. By transferring the provenance tracking responsibility from the user to the platform, we reduce the probability of inconsistent handling of provenance tracking between different stages, pipelines and even different users.

3 HOPSWORKS PROVENANCE TRACKING

At its simplest, a pipeline stage can be defined as an application, running user defined code in a well defined environment, based on a set of inputs including data and parameters and producing output data. In order to allow for a robust pipeline, resilient to failures and also to simplify our current implementation, we consider all input and output data as being persisted files or directories on the filesystem. Additionally, the environment (installed/used libraries) in which applications are run is persisted and updated on the filesystem as a json file. In order to allow for a slightly more flexible setup, we also allow a metadata layer where we can attach metadata to existing files.

Considering the above simplification of pipeline stages, the relevant components of a platform capable of running a ML pipeline involves a resource manager for running applications capable of analysing and processing Big Data, a distributed file system capable of storing Big Data and a metadata layer. In the case of our platform, Hopsworks [6], as we can see in Figure 1 we use HopsYARN,

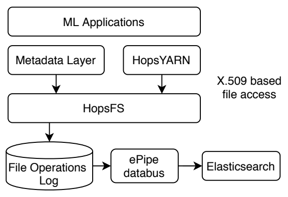

Figure 1: Hopsworks Provenance Architecture.

a modified version of YARN, as a resource manager, HopsFS [10], a modified version of HDFS [4], as a filesystem and a metadata layer based on HDFS Extended Attributes.

The metadata layer, based on the HDFS Extended Attributes mechanism, allows users to attach metadata to files. The file operations for attaching, updating and removing metadata is thus tightly coupled with the filesystem and is part of the set of operations that can be performed on files. As such, tracking provenance in the metadata layer is reduced to tracking provenance at the filesystem level. Another benefit we get from this mechanism is consistent metadata. As the operations for metadata operations are actually file system operations the files and metadata itself are always kept consistent and deleting the file for example, will also remove the attached metadata. As we can see in Figure 1 file operations including our extended metadata operations are logged and then asynchronously replicated by our databus service ePipe [7], to a consistent replica in Elasticsearch [5], for full text search capabilities.

At the resource manager level, the main provenance tracking mechanism is related to identifying exactly who access the files. The who in this case is which application and on which user's behalf the application is performing file operations. Our modified version of YARN accesses files by using X.509 certificates which contain the userId and applicationId.

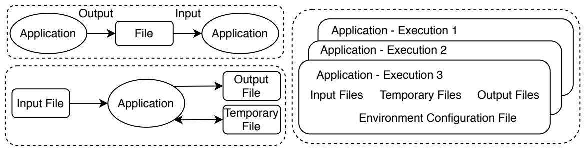

Figure 2: Contexts around files and applications.

At the file system level, we track CRUD operations as well as Extended Attribute operations. These operations are logged in our internal Mysql Cluster [9] database. As a distributed file system, HopsFS has multiple namenodes performing the file system operations. Thus, in order to extract a usable file lineage we require ordering of the file operations and this is achieved with the help of logical clocks. In order to have minimal overhead on performance, the logical clocks are implemented on a per inode level. This mechanism allows us to have the history of changes to a file as an ordered stream of events persisted in our log. The application identifier present in each of the file operations provides us with additional context. As we can see in Figure 2 this context allows us to infer additional dependencies between files, files and application, and applications. Old lineage is archived on the filesystem when trying to reduce the size of the provenance log.

Once input and output files are determined, we rely on two mechanism to translate this to Machine Learning artifacts. The first mechanism is the usage of location patterns. These locations patterns can be defined within out platform as filesystem paths where particular artifact types, such as models, features or experiments are saved. If location patterns are not used, the user can chose to use our metadata layer in the form of explicit provenance within the code itself, or they can tag the artifacts in our dataset browser after the execution of the application when the input and output files are presented to her.

4 USE CASES

The implicit provenance extracted from the executions of applications together with the tracking of file system operations, including the metadata layer tagging mechanism allows us to provide enough lineage information on the files to discover relevant dependencies for most common use cases. Using the implicit provenance we can detect changes to the artifacts of interest in order to trigger automatic pipeline execution, warning or suggestions for garbage collection. Having a context around applications and files can help in developing, understanding and debugging new pipelines. Having a full history of file changes from different pipelines as well a clear identifier for the issuer of operations allows us to easily support Data Governance and Auditing related use cases. With clear dependencies for each of the stages of a pipeline, we can easily reproduce results or inform the user that a change occurred that might end up with results that might be logically different from previous runs.

5 RELATED WORK

ModelDB was the first model management platform that had significant adoption [13]. The model governance work from Sridhar et al, introduces an interesting abstraction, the Intelligence Overlay Network, as a logical layer that is used to orchestrate ML workflows and store the creation path, subsequent usage, and consequent outcomes of ML models [12]. This approach, however, has no visibility of the internal ML pipeline stages, and whole training pipelines execute as only a single step in their provenance graphs. The use of metadata stores for tracking provenance is present in well known systems with MLFlow [14] requiring users to log operations on demand, TensorFlow Extended(TFX) [3] including the metadata logging as part of the calls to their libraries and the automatic metadata logging through extractors embedded in the user's code in the work of Schelte et al [11].

6 SUMMARY

In this paper, we described the provenance tracking mechanism used in Hopsworks. The implicit provenance is a novel system that is transparent and non-intrusive to user code and applies universally to all applications and file accesses, removing the human element as a possible source of error or omission when logging provenance. Provenance is tracked within the resource manager, the file system and the metadata layer and thus can accommodate running any application in any of the supported machine learning frameworks with no change to the code.

ACKNOWLEDGMENTS

This work is supported by the ExtremeEarth[1] project funded by European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 825258.

REFERENCES

[1] Airflow [n.d.]. Apache Airflow. https://airflow.apache.org. [Online; accessed 16-January-2020].

[2] Argo [n.d.]. Argo. https://argoproj.github.io/. [Online; accessed 16-January-2020].

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

[4] Dhruba Borthakur et al. 2008. HDFS architecture guide. *Hadoop Apache Project* 53, 1-13 (2008), 2.

[5] Elasticsearch [n.d.]. Elasticsearch. https://www.elastic.co/products/elasticsearch. [Online; accessed 16-January-2020].

[6] Mahmoud Ismail, Ermias Gebremeskel, Theofilos Kakantousis, Gautier Berthou, and Jim Dowling. 2017. Hopsworks: Improving User Experience and Development on Hadoop with Scalable, Strongly Consistent Metadata. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2525–2528.

[7] Mahmoud Ismail, Mikael Ronström, Seif Haridi, and Jim Dowling. 2019. ePipe: Near Real-Time Polyglot Persistence of HopsFS Metadata. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-May 17, 2019*. 92–101.

[8] Kubeflow [n.d.]. Kubeflow. https://www.kubeflow.org/. [Online; accessed 16-January-2020].

[9] MySQL Cluster [n.d.]. MySQL Cluster CGE. http://www.mysql.com/products/cluster/. [Online; accessed 16-January-2020].

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

[11] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. 2017. Automatically tracking metadata and provenance of machine learning experiments. In *Machine Learning Systems workshop at NIPS*.

[12] Vinay Sridhar, Sriram Subramanian, Dulcardo Arteaga, Swaminathan Sundararaman, Drew Roselli, and Nisha Talagala. 2018. Model governance: Reducing the anarchy of production {ML}. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 351–358.

[13] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. Model DB: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 14.

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

[1]Project website: http://earthanalytics.eu .