A Systems Perspective to Reproducibility in Production Machine Learning Domain

Sindhu Ghanta **Lior Khermosh** **Sriram Subramanian** **Vinay Sridhar**

Swaminathan Sundararaman **Dulcardo Arteaga** **Qianmei Luo** **Drew Roselli**

Dhananjoy Das **Nisha Talagala**

Abstract

Reproducibility of ML models and pipelines deployed in production requires capturing both the current and historic state. Capturing state in ML pipelines is complex due to the inherent nature of typical production deployments. We present a system that addresses these issues from a systems perspective, enabling ML experts to track and reproduce ML models and pipelines in production. This enables quick diagnosis of issues that occur in production.

1 Introduction

We focus on Machine Learning (ML) application reproducibility, with two specific elements. First, we focus on Production Reproducibility: the ability to faithfully reproduce ML behaviors that occur in production, a task with some similar and some different challenges from ML reproduction in development or research environments. Second, we strive to leverage lessons learned in the distributed systems space to address this problem. The combination has led us to the timeline mechanism that we describe in this paper. While our approach can be broadly used, our focus here is on using this approach for ML production diagnostics (which requires reproduction of and analysis of the issue in question). The challenges we face are: similar to reproducibility in other ML environments, production ML predictions depend on many factors: input data, pipeline configurations, historical predictions, trained models, training data, initial random seeds, etc. Furthermore, in production, ML applications are frequently logically distributed across machines and time. Finally, non-deterministic elements exist in production (such as incoming live data or hardware/software/systems errors) which complicate reproducibility. Approaches of storing all input configuration parameters is not enough in such cases and diagnostics can require replay (i.e., the ability to re-live the actual event rather than re-execute it).

We advocate for an application aware timeline capture, designed for capturing and managing events in realistic ML applications. Our contributions are: We describe the state capture problem in realistic complex ML applications and propose a novel approach. We compare and contrast our proposed approach with known systems approaches for reproducing other complex distributed applications. We demonstrate how this approach enables us to reproduce production ML outcomes by (a) capturing the sufficient configuration information to re-execute a pipeline if necessary, (b) capturing such information in conjunction pipeline dependencies such that even a sequence of dependent pipelines can be re-executed and (c) enabling a user to "re-live" a previous execution if factors within it such as live data or unexpected system errors make it difficult to faithfully re-execute for diagnostics. We also present an example of the working system.

Reproducibility in ML Workshop at the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden.

2 Complexity of Machine Learning Environments in Production

In this section we describe the complexities associated with practical production machine learning pipelines and the limitations of existing tools.

2.1 Machine Learning Applications

ML pipelines may run on analytic engines or as standalone programs. Figure 1 shows the complexity inherent in a realistic ML application. As discussed in the Introduction, diagnosing issues in such environments are challenged by *C*omplex Dependencies between Pipelines, *D*istribution and Heterogeneity, and *C*hanging Temporal State.

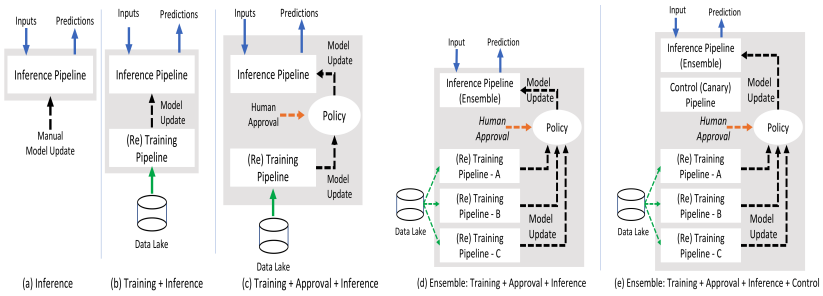


Figure 1: **Evolution of ML Pipelines in production.** *(a) The prediction pipeline is executing in production and is using a model trained offline and uploaded. (b) A more dynamic scenario where the model itself is retrained on a schedule. (c) A newly trained models undergo an approval process prior to production deployment. (d) An ensemble model is used for prediction (requiring each sub-model to be trained individually). Finally, (e) shows the scenario if a control pipeline (*canary*) is used in production to ensure that the primary prediction pipeline is behaving stably. The control pipeline could also be running a surrogate model to improve explainability.*

2.2 Limitations of Existing Techniques

We strive to leverage existing techniques and outline the popular ones below. File/Storage/Database Snapshots can version datasets. VM/Container snapshots can capture pipeline and system state. Source Control Systems can version pipeline components and (depending on pipeline construction) whole pipelines. Analytic engine checkpoints (e.g., Flink Checkpoints (6)) can capture pipeline execution state. Lineage-aware Storage Systems (e.g., Tachyon (13)) can link pipeline state with data state. Change Data Capture (33) and CDP techniques (9; 5) can capture dataset state.

None of these techniques capture the complete ML application as described in Figure 1. Even if all of these techniques are used, their relation to the complete picture is still missing. Also, infrastructure based state captures cannot differentiate between pipelines simultaneously executing within the same analytic engine instance. Launching a separate instance of an analytic engine (like Spark) for each pipeline, and packaging both as a container, can create a pipeline specific state capture. However, given that ML workflows execute as a collection of loosely coupled parallel programs, even such a per-pipeline state capture does not capture the dependencies between distributed dependent pipelines. Finally, a point in time state capture does not provide history to root cause or reproduce issues if those issues are non-deterministic (such as system errors).

Today, one would have to manually combine and extend these tools, an expensive, error prone and non-scalable process for deploying ML at scale.

3 System Design and Approach

We propose a system that captures complete ordered event sequences of ML applications, including:

ML Awareness: all state (code, pipelines, models, human actions, events errors, dataset access points, etc.), and ML dependencies that need to be captured.

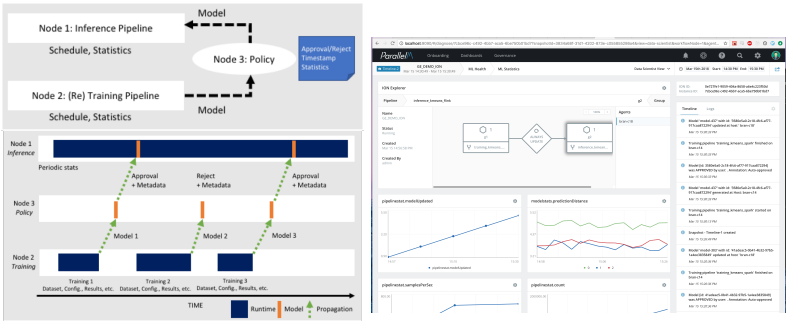
Heterogeneity: across execution locations, programming languages, and data stores.

Launch and Re:Launch: the full configuration of all dependent pipelines are captured, and reproducing a multi-pipeline execution can be done with a single step.

"After the Fact" capture: Our timeline captures are creatable "after the fact", i.e., a user can request a capture of a past (configurable) time frame. This is useful to record non-deterministic events.

Browsing: To "re-live" past executions, our system includes a Browse capability for all Timelines.

3.1 The Intelligence Overlay Network



(a) ION for Flow in Figure 1(c) and ION timeline.

(b) Browsable View of an ION.

Figure 2: ION flow and timeline.

The Intelligence Overlay Network (ION), a two-level logical graph of ML/data pipelines, policy modules and messages. Each node can itself be a Directed Acyclic Graph (DAG) of components. Figure 2a shows how Figure 1(c)'s pattern maps to an ION. The Inference and Training nodes are pipelines (or DAGs), any policies that dictate how these pipelines interact (such as a model transfer) is a Policy node, and the edges show the path of a model. Note that a model sent between nodes does not imply anything about when the nodes run. In this example, Node 1 (Inference) is a continuous streaming pipeline, Node 2 (Training) is a periodic batch pipeline, and Node 3 is an event driven policy module.

The ION contains sufficient information (code, pipeline structures, configuration parameters etc.) to reproduce the execution of not just an individual pipeline but the entire collection of dependent pipelines. Our system enables a single button launch (and re-launch) to recreate/reproduce execution.

We employ an Agent/Server infrastructure to deploy IONs. Due to lack of space, we do not elaborate on the details on this design/implementation: a more detailed description, justification and trade-off discussion is available in (29).

For ML pipelines, the server gathers everything from configuration state to runtime statistics (including ML specific stats, accuracy, and input data patterns) via agents. Each agent monitors pipelines via an engine-specific interaction and APIs are provided for pipelines to instrument for additional reporting. Due to lack of space, we do not elaborate further here on the specific mechanisms used to gather the statistics. More details are available in (26).

3.2 Managing Timeline Captures

A timeline capture is a persistent record of the merged temporal log of all pipelines, policies, events and other objects tracked above (for each ION). To create a ML timeline capture, we merely add a marker to ensure that certain sections of the logical log are not deleted. The marker is stored in our system with a name to help browse it.

Timeline Captures meets our stated reproducibility goals. First, state that is captured at the launch of an ION is sufficient to re-execute the entire ION on demand. Second, individual pipelines can be re-executed on demand. Finally, for non-deterministic events (such as live data changes, hardware errors etc.), the Timeline Capture browse enables users to re-live the original execution's progress.

3.3 Integrating with Existing Tools

Our system integrates with several state capture tools that exist in typical production environments. *Code versioning:* If pipeline code is in a source repository, we capture the pathname and commit number at ION definition or execution time, ensuring record of the precise version used in any computation. *Dataset versioning:* At every pipeline start, we capture any input dataset pathnames. *Model versioning:* If the underlying ML engine (such as TF (2)) versions models explicitly, our Agent fetches the model from the local version and stores it as a new version of the model in our database. Otherwise, we maintain our own versioning for models.

3.4 Re-Living and Sharing Timelines

All timeline captures can be graphically viewed over time (see 2b) in the same way as live data (providing a "virtual-reality" like ability to explore the past) (1). Named timeline captures exist till

deleted. Users can view the captures created by other users. An export serializes it into CSV/JSON files grouped into a tarball. They can be downloaded into a data scientist's laptop and imported into environments such as Juypter notebook, and Matlab.

4 Evaluation

We evaluated our system on the following: (a) ability to re-execute pipelines, (b) ability to re-execute in the presence of dependent pipelines, and (c) ability to "re-live" a previous execution. We have been using our system for more than a year and have run tens of thousands of IONs.

Re-execute Pipelines: In total, we have run more than thousand IONs and verified the ability to re-run the pipelines using our system. More importantly, we used our database (containing recorded statistics, models, predictions, events, etc.) to verify that the re-executed pipelines have the same output (or behavior) during re-execution.

Re-execute Dependent Pipelines: To verify the ability to re-execute dependent pipelines, we executed more than fifty pipelines with patterns similar to Figure 1 (b), (c), (e). We also included IONs that had A/B testing component in them. The minimum number of dependent pipelines were 2 and the maximum was 10. In all our experiments, we were able to validate the correctness by comparing the results of re-execution with the original run.

Re-live Executions: We have manually verified that the timeline capture's "browse" feature to re-live execution matches with the original execution. We are currently working on automatically validating the correctness of this features and the initial results are promising. Figure 3 shows an screenshot of timeline "browse" for an ION in our system.

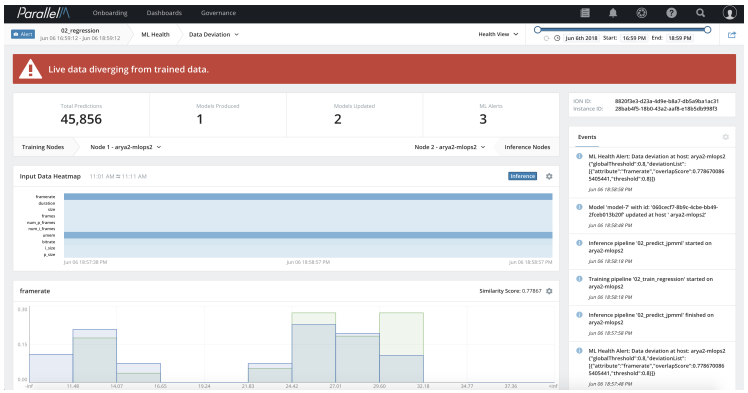


Figure 3: **Example Timeline browse screen for an ION.** The top left portion denotes the timeline name and the top right portion enables you to select the window of time to browse.

5 Related Work

Timelines and timeline capture systems exist (11; 31; 14; 15; 8; 16); the closest to ours is (18). To the best of our knowledge, we are the first to create comprehensive timelines for distributed ML applications, including all ML aspects such as pipelines, models, human approvals, and heterogeneous analytic engines. Analytic engines provide checkpoints (or snapshots) (34; 35; 6; 25; 4) for migrate/restart which do not track interdependencies between pipelines as we do. These are complementary and can be included within our system. Storage timelines, checkpoints, and snapshots are covered in (12; 32; 18; 23; 21; 27; 3; 20; 22). Our work can leverage logical clocks as these works do.

Also related is provenance/lineage in ML Model Management (19; 28; 10; 7; 17; 24; 30). This work and ours have several differences. Our focus is a logical timeline covering not just model lineage but all events, runtime statistics, human actions, errors, etc. for diagnostics. We have also focused on browsability, sharing, and export. Our design can also be used for Model Provenance, an area we are exploring that is out of the scope of this paper.

6 Future Work

Our future work is to further explore the performance and scalability of our design as well as experiment with additional analytic engines. We also expect that additional state captures such as VM snapshots can be integrated into our system.

References

[1] How to use time machine to back up or restore your mac. Internet draft, 2017. https://support.apple.com/en-us/HT201250.

[2] ABADI, M., AND ET AL. Tensorflow: Large-scale machine learning on heterogeneous systems. Internet draft, 2015.

[3] AGUILERA, M. K., SPENCE, S., AND VEITCH, A. Olive: Distributed point-in-time branching storage for real systems. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3* (Berkeley, CA, USA, 2006), NSDI'06, USENIX Association, pp. 27–27.

[4] DATMO. Ml snapshots. Internet Draft, 2018.

[5] FALCONSTOR SOFWARE, I. Continuous data protector (cdp).

[6] FLINK, A. Checkpoints in flink. Internet Draft, 2018.

[7] GLAVIC, B., AND DITTRICH, K. R. Data provenance: A categorization of existing approaches. In *BTW* (2007).

[8] HADZILACOS, V., AND TOUEG, S. Distributed systems (2nd ed.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993, ch. Fault-tolerant Broadcasts and Related Problems, pp. 97–145.

[9] IBM. Ibm tivoli continuous data protection for files. Internet Draft, 2018.

[10] KULKARNI, D. A provenance model for key-value systems. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance* (Berkeley, CA, USA, 2013), TaPP '13, USENIX Association, pp. 12:1–12:4.

[11] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July 1978), 558–565.

[12] LEE, E. K., AND THEKKATH, C. A. Petal: Distributed virtual disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 1996), ASPLOS VII, ACM, pp. 84–92.

[13] LI, H., GHODSI, A., ZAHARIA, M., SHENKER, S., AND STOICA, I. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, 2014), SOCC '14, ACM, pp. 6:1–6:15.

[14] LOWELL, D. E., CHANDRA, S., AND CHEN, P. M. Exploring failure transparency and the limits of generic recovery. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4* (Berkeley, CA, USA, 2000), OSDI'00, USENIX Association.

[15] LYNCH, N. A. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[16] LYNCH, N. A., AND SHVARTSMAN, A. A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing* (June 1997), pp. 272–281.

[17] MISSIER, P., DEY, S., BELHAJJAME, K., CUEVAS-VICENTTIN, V., AND LUDÄSCHER, B. D-prov: Extending the PROV provenance model with workflow structure. In *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13)* (Lombard, IL, 2013), USENIX Association.

[18] MOH, C.-H., AND LISKOV, B. Timeline: A high performance archive for a distributed object store. In *First Symposium on Networked Systems Design and Implementation (NSDI)* (San Francisco, CA, Mar. 2004).

[19] MUNISWAMY-REDDY, K.-K., HOLLAND, D. A., BRAUN, U., AND SELTZER, M. Provenance-aware storage systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference* (Berkeley, CA, USA, 2006), ATEC '06, USENIX Association, pp. 4–4.

[20] ÖZSOYOĞLU, G., AND SNODGRASS, R. T. Temporal and real-time databases: A survey. *Knowledge and Data Engineering 7*, 4 (1995), 513–532.

[21] QUINLAN, S., AND DORWARD, S. Awarded best paper! - venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2002), FAST '02, USENIX Association.

[22] REN, K., DIAMOND, T., ABADI, D. J., AND THOMSON, A. Low-overhead asynchronous checkpointing in main-memory database systems. In *Proceedings of the 2016 International Conference on Management of Data* (New York, NY, USA, 2016), SIGMOD '16, ACM, pp. 1539–1551.

[23] SANTRY, D. S., FEELEY, M. J., HUTCHINSON, N. C., VEITCH, A. C., CARTON, R. W., AND OFIR, J. Deciding when to forget in the elephant file system. *SIGOPS Oper. Syst. Rev. 34*, 2 (Apr. 2000), 18–19.

[24] SIMMHAN, Y. L., PLALE, B., AND GANNON, D. A survey of data provenance in e-science. *SIGMOD Rec. 34*, 3 (Sept. 2005), 31–36.

[25] SPARK, A. Checkpointing in spark. Internet Draft, 2018.

[26] SRIDHAR, V., SUBRAMANIAN, S., SUNDARARAMAN, S., ROSELLI, D., AND TALAGALA, N. Model governance: Reducing the anarchy of production ml. In *USENIX Annual Technical Conference 2018* (Boston, MA, USA, 2018).

[27] SUBRAMANIAN, S., SUNDARARAMAN, S., TALAGALA, N., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Snapshots in a flash with iosnap. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 23:1–23:14.

[28] SULTANA, S., AND BERTINO, E. A comprehensive model for provenance. In *Proceedings of the 2012 International Conference on Advances in Conceptual Modeling* (Berlin, Heidelberg, 2012), ER'12, Springer-Verlag, pp. 121–130.

[29] TALAGALA, N., SUNDARARAMAN, S., SRIDHAR, V.AND ARTEAGA, D., LUO, SUBRAMANIAN, S., GHANTA, S., KHERMOSH, L., AND ROSELLI, D. Eco: Harmonizing edge and cloud with ml orchestration. In *HotEdge 2018* (Boston, MA, USA, 2018).

[30] TAN, Y. S., KO, R. K. L., AND HOLMES, G. Security and data accountability in distributed systems: A provenance survey. In *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing* (Nov 2013), pp. 1571–1578.

[31] TANSEL, A. U., CLIFFORD, J., GADIA, S., JAJODIA, S., SEGEV, A., AND SNODGRASS, R., Eds. *Temporal Databases: Theory, Design, and Implementation*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.

[32] THEKKATH, C. A., MANN, T., AND LEE, E. K. Frangipani: A scalable distributed file system. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1997), SOSP '97, ACM, pp. 224–237.

[33] VAN DE WIEL, M. *Asynchronous Change Data Capture Cookbook*. Oracle Corporation, 2007.

[34] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 2–2.

[35] ZAHARIA, M., XIN, R. S., WENDELL, P., DAS, T., ARMBRUST, M., DAVE, A., MENG, X., ROSEN, J., VENKATARAMAN, S., FRANKLIN, M. J., GHODSI, A., GONZALEZ, J., SHENKER, S., AND STOICA, I. Apache spark: A unified engine for big data processing. *Communications of the ACM 59*, 11 (Oct. 2016), 56–65.