An Experimentation and Analytics Framework for Large-Scale AI Operations Platforms

Thomas Rausch[1,2], Waldemar Hummer[2], Vinod Muthusamy[2]
[1] *TU Wien*, [2] *IBM Research AI*

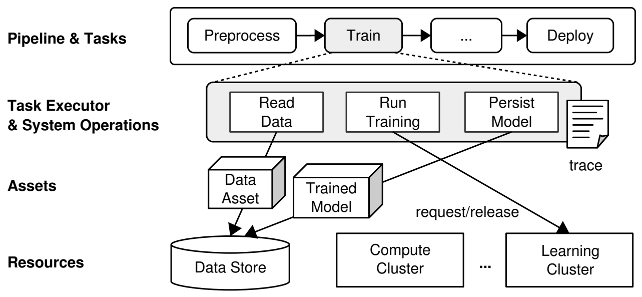
Abstract

This paper presents a trace-driven experimentation and analytics framework that allows researchers and engineers to devise and evaluate operational strategies for large-scale AI workflow systems. Analytics data from a production-grade AI platform developed at IBM are used to build a comprehensive system and simulation model. Synthetic traces are made available for ad-hoc exploration as well as statistical analysis of experiments to test and examine pipeline scheduling, cluster resource allocation, and similar operational mechanisms.

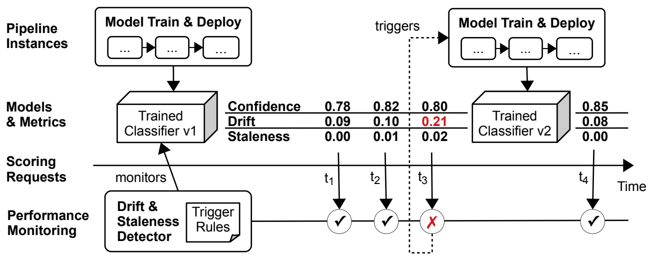
1 Introduction

Operationalizing AI has become a major endeavor in both research and industry. Automated, operationalized pipelines that manage the AI application lifecycle will form a significant part of infrastructure workloads [6]. AI workflow platforms [1, 6] orchestrate the heterogeneous infrastructure required to operate a large number of customer-specific AI pipelines. It is challenging to fine-tune operational strategies that achieve application-specific cost-benefit tradeoffs while catering to the specific domain characteristics of ML models, such as accuracy, or robustness. A key challenge is to determine the cost trade-offs associated with executing a pipeline, and the potential model performance improvement [5–7].

We present a trace-driven experimentation and analytics environment that allows researchers and engineers to devise and evaluate such operational strategies for large-scale AI workflow systems. Traces from a production-grade AI platform developed at IBM, recorded from several thousand pipeline executions over the course of a year are used to build a comprehensive simulation model. Our simulation model describes the interaction between pipelines and system infrastructure, and how pipeline tasks affect different ML model metrics. We implement the model in a standalone, stochastic, discrete event simulator, and provide a toolkit for running experiments. By integrating a time-series database and analytics front-end, we allow for ad-hoc exploration as well as statistical analysis of experiments.



(a) Build-time view



(b) Run-time view

Figure 1: Conceptual system model of AI ops platforms.

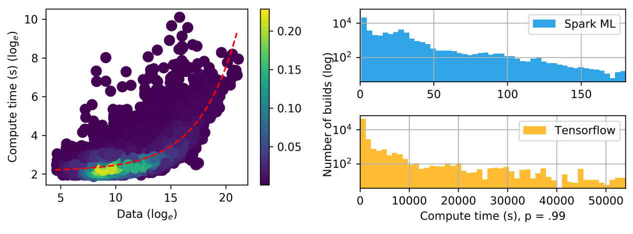
2 System Description

Conceptual Model Automated AI ops pipelines integrate the entire lifecycle of an AI model, from training, to deployment, to runtime monitoring [1, 6, 10]. To manage risks and prevent models from becoming stale, pipelines are triggered automatically by monitoring runtime performance indicators of deployed models. It is therefore essential to model both build-time and run-time aspects of the system. In general, we say that a model has a set of *static* and *dynamic* properties. Static properties are assigned by the pipeline at build-time, such as the prediction type (e.g., classification, or regression) or the model type (e.g., random forests or DNN). Dynamic properties change during runtime, such as model performance or robustness scores [13].

Build-time view: AI *pipelines* are workflows, i.e., graph-structured compositions of *tasks*, that create or operate on machine learning models [6]. At build time, an AI pipeline generates or augments a *trained model* by operating on *data assets* and using underlying infrastructure *resources* (e.g., data store, cluster compute or GPU nodes).

Run-time view: The outcome of a successful pipeline execution is usually a deployed model that is being served by the platform and used by applications for scoring. At runtime, the deployed model has associated performance indicators that change over time. Some indicators can be measured directly, by inspecting the scoring inputs and outputs (e.g., confidence), whereas other metrics (e.g., bias, or drift [4, 11]) require continuous evaluation of the runtime data against the statistical properties of the historical scorings and training data.

Synthesizing & Simulating Pipelines We synthesize plausible pipelines from three common pipeline structures we have identified by analyzing both commercial and research use cases [2, 3, 6, 9]. The generated pipelines vary in parameters such as the type of model they generate, the ML frameworks they use, or the number of tasks. The parameters are sampled from distributions we have fitted over analytics data. In the same way, we sample the metadata of data assets (such as the number of dimensions and instances or the size in bytes) as input for pipelines. We currently model the following pipeline steps: (a) data pre-processing (performing manipulation operations on the training data), (b) model training, (c) model validation, (d) model compression (e.g., removing layers from DNNs [12], changing the model size).



(a) Data preprocessing task (b) Training task

Figure 2: Observations of compute time for data preprocessing and training tasks based on other known properties.

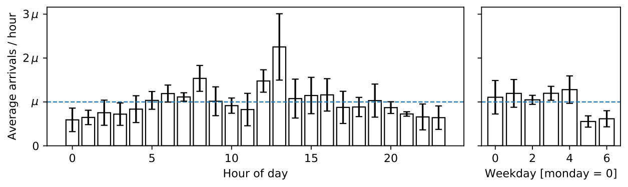
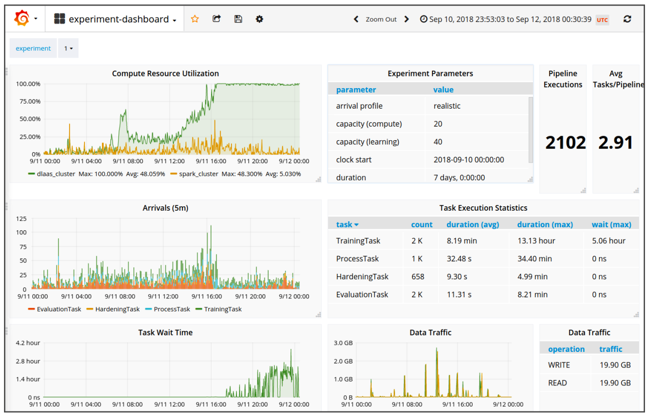


Figure 3: Average arrivals per hour stratified by hour of day and weekday ($n = 210\,824$). μ shows the average arrivals per hour overall. Error bars show one standard deviation.


experiment-dashboard
Zoom Out
Sep 10, 2018 23:53:03 to Sep 12, 2018 00:30:39 UTC
experiment
1
Compute Resource Utilization
dlaas_cluster Max: 100.000% Avg: 48.059%
spark_cluster Max: 48.300% Avg: 5.030%
Arrivals (5m)
EvaluationTask
HardeningTask
ProcessTask
TrainingTask
Task Wait Time
Data Traffic

Experiment Parameters

parameter	value
arrival profile	realistic
capacity (compute)	20
capacity (learning)	40
clock start	2018-09-10 00:00:00
duration	7 days, 0:00:00

Pipeline Executions: 2102

Avg Tasks/Pipeline: 2.91

Task Execution Statistics

task	count	duration (avg)	duration (max)	wait (max)
TrainingTask	2 K	8.19 min	13.13 hour	5.06 hour
ProcessTask	1 K	32.48 s	34.40 min	0 ns
HardeningTask	658	9.30 s	4.99 min	0 ns
EvaluationTask	2 K	11.31 s	8.21 min	0 ns

Data Traffic

operation	traffic
WRITE	19.90 GB
READ	19.90 GB

Figure 4: Experiment analytics dashboard showing infrastructure and pipeline execution metrics of an experiment run

For simulating pipeline executions, we have developed a stochastic, discrete-event simulator called PipeSim, which implements the conceptual system model and data synthesizers. We simulate task execution time and resource use based on our traces. Figure 2 shows examples of ways to simulate the compute time of data preprocessing and training tasks. For example, we correlate the size of a data asset with the preprocessing time Figure 2(a). For a training task, we stratify the observations into the frameworks they used, and the data asset size they processed. Figure 2(b) shows a distribution of compute times for Tensorflow and SparkML tasks.

To generate random workload we model the interarrivals of pipeline triggers in seconds as a random variable and sequentially draw from a fitted exponentiated Weibull distribution, which we found to produce a good fit. We variate means based on arrival patterns from our observations. Figure 3 shows pipeline triggers per hour averaged over several hundred thousand pipeline executions.

Experiment Runner & Explorer PipeSim is based on SimPy [8] and persists synthetic traces into InfluxDB. Resource allocation and scheduling algorithms are integrated as Python code into the simulator, which can then be evaluated by running PipeSim. The analytics frontend shown in Figure 4 allows exploratory analysis of experiment results. It displays the experiment parameters, general statistics about individual task executions and wait time. The graphs show the resource utilization of compute resources, individual tasks arrivals, network traffic, and overall wait time of pipelines, which allows us to quickly observe the impact of resource utilization on pipeline wait times.

We modeled the production system with PipeSim and analyzed the active scheduling policies. Experiments allowed us to approximate the increased execution times of pipelines given the projected user growth for the next year, and identify GPU cluster resource bottlenecks. We are now working to simulate and visualize aggregated model metrics to examine the effect of pipeline scheduling on overall model performance.

References

[1] 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.

[2] IBM Corporation. Medtronic builds a cognitive mobile personal assistant app to assist with daily diabetes management, 2017. IBM Case Studies.

[3] IBM Corporation. An AI-powered assistant for your field technician. Technical report, 2018.

[4] Jaka Demšar and Zoran Bosnić. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, 92:546–559, 2018.

[5] Sindhu Ghanta, Sriram Subramanian, Lior Khermosh, Harshil Shah, Yakov Goldberg, Swaminathan Sundararaman, Drew Roselli, and Nisha Talagala. MPP: Model performance predictor. In *2019 USENIX Conference on Operational Machine Learning*, OpML 19, pages 23–25, 2019.

[6] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, and Kaoutar El Maghraoui. Modelops: Cloud-based lifecycle management for reliable and trusted ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E'19)*, Jun 2019.

[7] Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *Proc. VLDB Endow.*, 11(5):607–620, January 2018.

[8] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009), 2008.

[9] Thomas Rausch, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. Towards a serverless platform for edge AI. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[10] Evan R Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J Franklin, and Benjamin Recht. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 535–546. IEEE, 2017.

[11] Yange Sun, Zhihai Wang, Haiyang Liu, Chao Du, and Jidong Yuan. Online ensemble using adaptive windowing for data streams with concept drift. *International Journal of Distributed Sensor Networks*, 12(5):4218973, 2016.

[12] Dharma Teja Vooturi, Saurabh Goyal, Anamitra R. Choudhury, Yogish Sabharwal, and Ashish Verma. Efficient inferencing of compressed deep neural networks. *CoRR*, abs/1711.00244, 2017.

[13] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.