`tensorflow-tracing`: A Performance Tuning Framework for Production

Sayed Hadi Hashemi[+△], Paul Rausch, Benjamin Rabe[+△]
Kuan-Yen Chou[+], Simeng Liu[+△], Volodymyr Kindratenko[△], Roy H Campbell[+]

1 Introduction

The growing popularity of Deep Neural Networks (DNN) within the mainstream [8] has had a rapid transformative effect on clusters and data centers. DNN training jobs are becoming one of the largest tenants within clusters, and often take hours to weeks to complete; and even a slight performance improvement can save substantial runtime costs. Despite this fact, the DNN specific performance tuning tools are yet to keep up with the needs of the new changes in production environments.

On one hand, the existing application-agnostic resource-level tools such as top, Nvidia Nsight (for GPU utilization), IPM (for MPI network monitoring) are too limited to predict or explain the behavior and performance of a job accurately. In DNN applications, there exists a complex relationship among resources. Even though measuring coarse metrics such as bandwidth, latency, and GPU/CPU utilization can draw an overall picture of cluster performance, these metrics are not easily translatable to application-level metrics and do not provide actionable insights on how to handle performance bottlenecks.

On the other hand, the short list of application-aware tools, such as MLModelScope [6], TensorBoard [1], and `tf.RunOptions` [2], while able to provide actionable insights, are mainly designed for the need of application developers and are not intended for production use. Such tools require substantial modification to applications, and early planning as to what, when and how data should be collected.

In this article, we introduce `tensorflow-tracing` to fill the gap between these two classes of performance tuning tools. To achieve this goal, `tensorflow-tracing` addresses the following technical challenges:

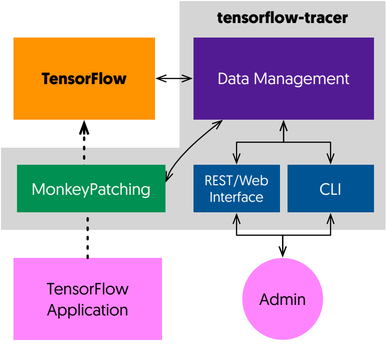


Figure 1: The architecture of `tensorflow-tracing`

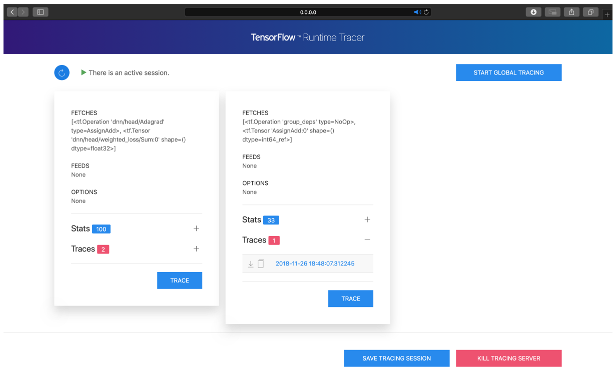
- Collecting the application-level runtime metrics, such as the timing of each operation or the iteration time, needs explicitly expressed in the training job source code. To makes it possible to trace ML jobs without requiring any application modification, `tensorflow-tracing` *monkeypatches* the `tensorflow` library at the system level.
- Collecting some metrics is expensive and have a significant overhead on the runtime. `tensorflow-tracing` treats metrics differently; it collects low-overhead metrics automatically, while expensive ones are collected on demand through an admin interface.
- There is no easy way to exchange runtime metrics among users and admins — `tensorflow-tracing` facilities this through a portable file format and supporting tools to explore these metrics offline.

The `tensorflow-tracing` is publicly available under `Apache-2.0` license[1]. It supports native TensorFlow [3], Horovod [7], and IBM PowerAI [5] applications.

[+] University of Illinois at Urbana-Champaign
[△] National Center for Supercomputing Applications

This material is based on work supported by the National Science Foundation under Grant No. 1725729.

[1] `https://github.com/xldrx/tensorflow-tracer`


0.0.0.0

TensorFlow™ Runtime Tracer

▶ There is an active session.

START GLOBAL TRACING

FETCHES
[<tf.Operation 'dnn/head/Adagrad' type=AssignAdd>, <tf.Tensor 'dnn/head/weighted_loss/Sum:0' shape=() dtype=float32>]

FEEDS
None

OPTIONS
None

Stats 100 +

Traces 2 +

TRACE

FETCHES
[<tf.Operation 'group_deps' type=NoOp>, <tf.Tensor 'AssignAdd:0' shape=() dtype=int64_ref>]

FEEDS
None

OPTIONS
None

Stats 33 +

Traces 1 −

2018-11-26 18:48:07.312245

TRACE

SAVE TRACING SESSION

KILL TRACING SERVER

Figure 2: The main web interface of `tensorflow-tracing`. Each entry represents a separate task in the DNN session.

2 Design and Implementation

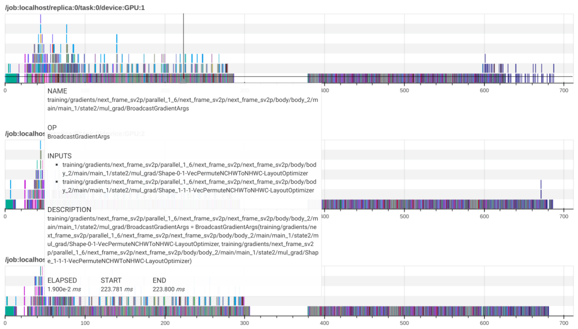
Figure 1 shows the building blocks of `tensorflow-tracing`:

MonkeyPatching In order to provide tracing without code modification, `tensorflow-tracing` injects a proxy function to the `tensorflow` library using a *monkey-patching* scheme to intercepts the calls to certain functions and redirects them to the *Data Management* module. While *monkeypatching* the library at the system-level automatically enables tracing for any DNN application, `tensorflow-tracing` also supports per application patching.

Data Management This module is responsible for collecting profiling and tracing data as well as making online decisions as to whether a task should be traced[2]. This module is also responsible for serializing/deserializing tracing sessions from/to a file.

REST/Web Interface This interface is the main portal for interacting with the system. `tensorflow-tracing` starts a web server whenever an application is executed which is accessible either through a web browser or a REST API client (possibly from terminal). The interface provides two logical views:

1. *Main Interface* shows the list of tasks and their associated profiling/tracing data. This interface allows request tracing. (Figure 2)
2. *Timeline Interface* visualizes an instance of a task trace as a series of timelines, one for every resources (e.g. CPU, GPU, Network Interface) on each machine. Each box represent an operation in the DataFlow DAG of DNN application. (Figure 3)



NAME
training/gradients/next_frame_sv2p/parallel_1_6/next_frame_sv2p/next_frame_sv2p/body/body_2/main/main_1/state2/mul_grad/BroadcastGradientArgs

OP
BroadcastGradientArgs

INPUTS
- training/gradients/next_frame_sv2p/parallel_1_6/next_frame_sv2p/next_frame_sv2p/body/body_2/main/main_1/state2/mul_grad/Shape-0-1-VecPermuteNCHWToNHWC-LayoutOptimizer
- training/gradients/next_frame_sv2p/parallel_1_6/next_frame_sv2p/next_frame_sv2p/body/body_2/main/main_1/state2/mul_grad/Shape_1-1-1-VecPermuteNCHWToNHWC-LayoutOptimizer

DESCRIPTION
training/gradients/next_frame_sv2p/parallel_1_6/next_frame_sv2p/next_frame_sv2p/body/body_2/main/main_1/state2/mul_grad/BroadcastGradientArgs = BroadcastGradientArgs(training/gradients/next_frame_sv2p/parallel_1_6/next_frame_sv2p/next_frame_sv2p/body/body_2/main/main_1/state2/mul_grad/Shape-0-1-VecPermuteNCHWToNHWC-LayoutOptimizer, training/gradients/next_frame_sv2p/parallel_1_6/next_frame_sv2p/next_frame_sv2p/body/body_2/main/main_1/state2/mul_grad/Shape_1-1-1-VecPermuteNCHWToNHWC-LayoutOptimizer)

ELAPSED	START	END
1.900e-2 *ms*	223.781 *ms*	223.800 *ms*

Figure 3: The timeline interface provides details of the execution of one iteration step. Each box represent an operation in the DataFlow DAG. There is a timeline for every resources on each machine. The trace is collected from `next_frame_sv2p` [4] model in `tensor2tensor` library [10].

CLI It loads a tracing sessions offline and enables exploring through a web interface.

3 `tensorflow-tracing` in action

Overhead We observe no performance hit on collecting low-overhead metrics such as iteration times, 'session.run' call names and frequencies. We observe less than 3% runtime overhead to iteration time when individual operations in a call are traced. CPU Memory requirements varies for different models. For example: an *Inception v3* [9] trace consumes 718KB while **next_frame_sv2p** [4] consumes 2.4MB.

Case Study We have used `tensorflow-tracing` on different workloads to find the performance issues on application, framework, and infrastructure level. For example, our work TicTac **??** addresses the communication timing issue we found in the tracing of a distributed TensorFlow job with parameter server.

4 Limitation and Future Work

The correctness of the `tensorflow-tracing`'s distributed traces relies on the precision of the clocks on the different machines. Currently it relies on external sources to synchronize the clocks.

`tensorflow-tracing` traces the network activities just in the user space. This will miss the events such as packet drops or queue latencies. We are planning to expand this capability by adding network stack events from kernel space.

[2] `MonitoredSession.run` function calls

References

[1] Tensorboard: Visualizing learning. https://www.tensorflow.org/guide/summaries_and_tensorboard, Accessed: February 15, 2019.

[2] tf.runoptions. https://www.tensorflow.org/api_docs/python/tf/RunOptions, Accessed: February 15, 2019.

[3] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[4] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy Campbell, and Sergey Levine. Stochastic variational video prediction. 2018.

[5] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. Powerai ddl. *arXiv preprint arXiv:1708.02188*, 2017.

[6] Abdul Dakkak, Cheng Li, Abhishek Srivastava, Jinjun Xiong, and Wen-Mei Hwu. MLModelScope: Evaluate and Measure ML Models within AI Pipelines. *arXiv preprint arXiv:1811.09737*, 2018.

[7] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[8] Svetlana Sicular and Kenneth Brant. Hype cycle for artificial intelligence, 2018, July 2018.

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[10] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018.