Benchmarking RunInference with Multi-Process Shared Models

Danny McCormick (dannymccormick@google.com)

Overview

Right now, using RunInference with large models and on GPUs has several performance gaps. This document focuses on one: when running inference with large models, we often OOM because we load several copies of the model at once. This document explores using the multi_process_shared.py utility to load models and finds that we can recommend using the utility for pipelines which load a large model for inference, but not for pipelines that normally don't have memory issues.

Goals

- Benchmark the usage of shared/multi_process_shared.py in RunInference for various configurations
- Recommend improvements to leverage the outcome of these benchmarks

Background

Right now, using RunInference with large models and on GPUs has several performance gaps. When running inference with large models, pipelines risk OOMing or forcing users to significantly scale up their machines. GPUs also often run into memory challenges and can easily OOM. Anecdotally, users seem more likely to run large models on GPUs.

The cause of these OOMs is well known; RunInference uses shared.py to run inference on a single shared model per process. By default, however, many Beam execution engines run multiple processes per worker. When these processes all load a large model, it can cause an OOM. Some engines (like Dataflow) provide controls to limit the number of processes per

worker (`no_use_multiple_sdk_containers` is the Dataflow experiment that does this), but not all engines have this mechanism and even for those that do, this is a less efficient mechanism for data-parallel operations because of Python's Global Interpreter Lock (GIL). Despite this limitation, `no_use_multiple_sdk_containers` is the current recommendation for inference on GPUs in Dataflow because memory problems are so pervasive. While all GPU cores are still visible from this process, this still leads to problems with the GIL and can cause GPU cores to be underfed.

Late in 2022, a utility class for managing multiprocess shared objects was introduced to Beam, but its performance characteristics have not been measured.

Experiment 1 - Small Tensorflow Model

The next set of benchmark tests was performed against the Tensorflow imagenet example in the Beam repo. This was run against 100,000 images.

This example runs inference using 1 small Tensorflow model, partitioned by the features available.

This example was run against the following experiments and configurations and took the following amount of vCPU hours. Since machine type was constant throughout the experiment, vCPU hours is the best measure of performance.

Experiment/Configuration set	**Job links (jobs may not be visible to everyone)**	**vCPU hours (run 1)**	**vCPU hours (run 2)**	**vCPU hours (run 3)**	**vCPU hours (avg)**
- no_use_multiple_sdk_containers - use_runner_v2	- Run 1 - Run 2 - Run 3	4.451	4.503	4.520	4.491
- use_runner_v2	- Run 1 - Run 2 - Run 3	4.303	3.997	3.856	4.052
- use_sibling_sdk_workers - use_runner_v2	- Run 1 - Run 2 - Run 3	4.137	3.564	4.019	3.907
- use_sibling_sdk_workers - use_runner_v2 - **use multi_process_shared instead of shared for model storage**	- Run 1 - Run 2 - Run 3	4.863	4.883	4.500	4.749
- no_use_multiple_sdk_containers - use_runner_v2 - worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver	- Run 1 - Run 2 - Run 3	3.805	4.113	4.164	4.027
- use_runner_v2	- Run 1	3.736	3.472	3.785	3.664

- worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver	- Run 2 - Run 3				
- use_runner_v2 - worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver;use_nvidia_mps	- Run 1 - Run 2 - Run 3	3.562	3.443	3.805	3.603
- use_sibling_sdk_workers - use_runner_v2 - worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver	- Run 1 - Run 2 - Run 3	3.529	3.860	3.395	3.597
- use_sibling_sdk_workers - use_runner_v2 - worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver;use_nvidia_mps	- Run 1 - Run 2 - Run 3	3.652	3.652	3.690	3.665
- use_sibling_sdk_workers - use_runner_v2 - **use multi_process_shared instead of shared for model storage** - worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver	- Run 1 - Run 2 - Run 3	3.984	4.311	4.284	4.193
- use_sibling_sdk_workers - use_runner_v2 - **use multi_process_shared instead of shared for model storage** - worker_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver;use_nvidia_mps	- Run 1 - Run 2 - Run 3	4.148	4.338	4.198	4.228

Analysis

When run against a small Tensorflow model that can easily fit into memory, loading the model with shared.py is clearly more efficient than using multi_process_shared.py. In fact, even with no_use_multiple_sdk_containers set to true when using shared.py and allowing multiple sdk containers when using multi_process_shared.py, the shared.py version still outperforms the multi_process_shared.py version.

Experiment 2 - Large Model

The next set of experiments was run against a single large model that would normally cause a worker to OOM without using the `no_use_multiple_sdk_containers` experiment. This experiment was run against a custom pipeline using the t5-large model as described in the Large Lanugage Model Inference example. The example was extended to run against 5,000 records. It was run using the n1-highmem64 machine type. Each run was performed just once without looking at GPUs to demonstrate whether the models could be correctly loaded since the

CPU performance characteristics were not expected to be very different. In a pipeline where inference was less dominant compared to IO or GPU-level pre/postprocessing, having multiple SDK containers would be more significant.

Experiment/Configuration set	**Job links (jobs may not be visible to everyone)**	**vCPU hours**
- no_use_multiple_sdk_containers - use_runner_v2	- Run	4123.431
- use_runner_v2	- Run - OOM	N/A
- use_sibling_sdk_workers - use_runner_v2	- Run - OOM	N/A
- use_sibling_sdk_workers - use_runner_v2 - **use multi_process_shared instead of shared for model storage**	- Run	3,466

Analysis

When loading large models, using multi_process_shared.py allows us to avoid using no_use_multiple_sdk_containters and leads to superior performance. It seems reasonably safe to assume that for models which are large enough to normally cause an OOM, using multi_process_shared.py will provide at least a small benefit.

Next Steps

We will allow users to specify if their model is large as a configuration option in their ModelHandler. Given a model that consumes N memory, we will recommend that they use this option if they expect that N * num_worker_processes per machine > machine memory. If the model is large, we will use multi_process_shared.py, if not we will use shared.py.