AD for pap566：Towards Optimal Caching with Reuse Distance Prediction via Long Short-Term Memory

1. Hardware

All the hardware used in this paper is the corporate resources of Alibaba. According to the corporate security policy, there is no way to vpn-access the hardware resources from outside at this moment. But, we would like to give the detailed machine architectures and configurations as below.

All deep learning tasks were performed on a Nvidia Tesla V100 GPU card of the recent Volta architecture, which has 5120 streaming cores, 640 tensor cores and 32GB memory capacity. The CPU host is Intel Xeon CPU 8163 2.50GHz, running Linux kernel 5.0.

2. Software and Code

The code used in this paper is partially running in the production at Alibaba. According to the corporate security policy, there is no way to open-source the code at this moment. But, we would like to describe some basics of the code structures and the instructions to run.

2.1 For trace pre-processing

- extract_msr.py: convert MSR original traces into the format of memory access requests.
- count_diff.py: count the number of different data blocks in a trace.
- preprocess.py: merge partial traces into one complete trace.
- global state.py: generate the first 6 global features of each data access, and partition the trace into uniform trunks for faster file readings in later experiments.

How to run:

- python extract_msr.py trace_name volume_number
    - output: trace_name volume_number trace.tr
- python count_diff.py trace_name volume_number
    - output: the number of different data blocks in trace_name volume_number trace.tr (denoted as key i for trace name i trace.tr, when the volume number is equal to i.)
- python preprocess.py trace_name total_volume_number key_0 ... key_n
    - output: trace.csv (assuming total volume number = n + 1)

- python global_state.py trace_name trunk_size
    - output: trace st 0.csv, trace st 1.csv, . . .

2.2 For feature engineering

- feature_trace.h feature_trace.cc
    - Compute features from the converted traces above.
- feature_to_examples.h feature_to_examples.cc
    - Convert the computed features to training data sets and testing data sets, stored in .tfrecord data format.

How to build:

- We have a BUILD file for all C++ programs.
- >> bazel build -c opt :feature_trace
- >> bazel build -c opt :feature_to_examples

How to run:

- >> ./bazel-bin/feature_trace --trace_path <trace> --last_k=<a number>
    - Output: a feature vector for each access
- >> ./bazel-bin/feature_to_examples --dataset_path <trace_feature_file> --num_examples=<a number>
    - Output: a training data set or test data set in the .tfrecord data format.

2.3. For model training and inference

- seq_stacked_predictor.py
    - Train model and make predictions for given test data sets.

How to run:

- >> python3 -W ignore predictors/seq_stacked_predictor.py
  --epochs=100 \
  --seq_length=1024 \
  --batch_size=64 --data_path "data_sets/data_set_web_pred" \
  --out_path "data_sets/data_set_web_pred"

2.4 For pOPT cache

- popt.h popt.cc
    - Run Algorithm 1 with the predicted forward reuse distance

How to build:

- We have a BUILD file for all C++ programs.

- >> bazel build -c opt :popt

How to run:

- >> ./bazel-bin/popt --trace_path <trace> --forward_reuse_distance_path <rd_file> --cache_size=<a number> --out_dir <out directory>
    - Output: cache miss rate given the cache size.

2.5 For baselines

- We implemented LRU, ARC, DDRIP, OPT by ourselves based on the existing references or their definitions.
- The 2Q implementation refers to the version used in lease cache work[28].
- Parrot implementation refers to https://github.com/google-research/google-research/tree/master/cache_replacement

All these baseline implementations are not allowed to be open-sourced according to the corporate policy.

3. Workloads

In our experiment, we considered the traces from Microsoft cloud storage workloads, which can be downloaded from http://iotta.snia.org/traces/388.

We ignore the difference between reads and writes, and simplify them as indiscriminate memory access requests. Traces from volumes of the same domain are combined as one complete trace, since they access the same type of data.

4. Tools

- Bazel version: 3.5.0
- Python: 3.7.4

5 Steps to run our tools

1. Preprocess downloaded Microsoft cloud storage traces, by running scripts in 2.1.
2. Compute data features from the preprocessed traces, by following the instructions of 2.2.
3. Store these computed features into a training data set and a testing data set, by following the instructions of 2.2.
4. Train the proposed model and test the model, by following 2.3.
5. With the predicted forward reuse distance values, run the pOPT cache, by following the instructions of 2.4.

6. Lastly, we get the cache miss rate for each workload when running pOPT.