Post-hoc Training Analysis with Flor

Eric Liu, Nishita Shetty, Anusha Dandamudi, Rolando Garcia, Vikram Sreekanti, Joseph Gonzalez, Joseph Hellerstein

A System for Diagnostics and Sharing

Fast, Memoization-backed Re-execution

First Execution: Flor-fork Memoization

Parallel Re-execution

Evaluation

Serialization in the main thread is by far the biggest Flor overhead during execution. The best method of cheaply absorbing these costs is with fork. Similarly, deserialization is the biggest bottleneck during re-execution. Of the methods we found worth exploring, plasma offers the shortest runtime due to zero-copy sharing.

Larger buffer sizes decrease the frequency of fork, resulting in less overhead or thrashing. However, in real-world experiments (i.e., Resnet18), we see that runtime is less impacted by buffer size. We suspect Python has overhead for maintaining large buffers.

Since execution cost is still high, we must make up the difference with parallel re-execution.

Conclusions

- Flor achieves execution and re-execution runtime below 2x the runtime of No Flor. We minimize runtime by reducing what we log during execution and then parallelizing re-execution.
- We can parallelize re-execution because memoizing model states allows for independent parallel execution; this allows for between 4x and 32x speedup, depending on the query type.
- In real-world benchmarks (Resnet, SqueezeNet, etc.) batch fork suffers from buffer maintenance overhead, likely due to memory reorganization in Python.
- Plasma, with its zero-copy deserialization of objects, is perfect for recovering information from parallel re-execution.

We ideally want to keep the cost of execution and re-execution below 2x the cost of No Flor execution. We gain up to a 32x speedup on re-execution as well as a minor execution speedup due to parallelized logging and identifying unimportant log statements.