An Optimization Layer for Distributed Matrix Computations

Jonah Brown-Cohen, Tselil Schramm, and Ben Weitz

Motivation

Companies like Facebook, Netflix, and Google perform large-scale distributed matrix computations.

- Trade-offs: accuracy vs. time vs. money
- Humans: bad at choosing parameters
- Solution: an optimization layer to automatically manage computations
- Learn from past computations
- Meet time, accuracy, or monetary budget

Optimizer Design

Optimizer stores statistics from prior jobs
- Architecture-independent
- Parameters chosen based on statistics from prior jobs on the same architecture
- Adaptive
- Predictions for data from a specific distribution improve as the optimizer learns
- Avoids Local Optima
- When there is variation in results, there is a risk of getting stuck at a local optimum
- In "explore mode" we add randomness: parameters are chosen with probability proportional to their relative suitability

Objective

Automate the choice of algorithm parameters and number of partitions to meet time, accuracy, and monetary budget specifications

Framework

- User specifies input data, and time, error, and monetary budgets
- Optimizer then interfaces with algorithms implemented on top of a parallel framework
- All parameter selection hidden from user

Implementation

- Distributed matrix computation: Divide-Factor-Combine (DFC)
- Parallel computation framework: SparkR
- Two different base factorization methods: Stochastic Gradient Descent and Accelerated Proximal Gradient Descent
- Randomized and deterministic projection methods
- Our implementation of DFC to be incorporated into SparkR

Future Work and Acknowledgements

Directions for further work include:
- Optimize over space of algorithms
- Handle novel jobs from new distributions (coming soon)

Our very warm thanks to:
- Professors Anthony Joseph and John Kubiatowicz
- Ameet Talwalkar and Shivaram Venkataraman for mentoring us
- UC Berkeley and the NSF for providing funds and resources