An Optimization Layer for Distributed Matrix Computations

Jonah Brown Cohen, Tselil Schramm, and Ben Weitz

Motivation

- Big data companies like Facebook, Netlix, or Google perform large-scale distributed matrix computations
  - Computations experience **trade-offs** in accuracy vs. time or money.

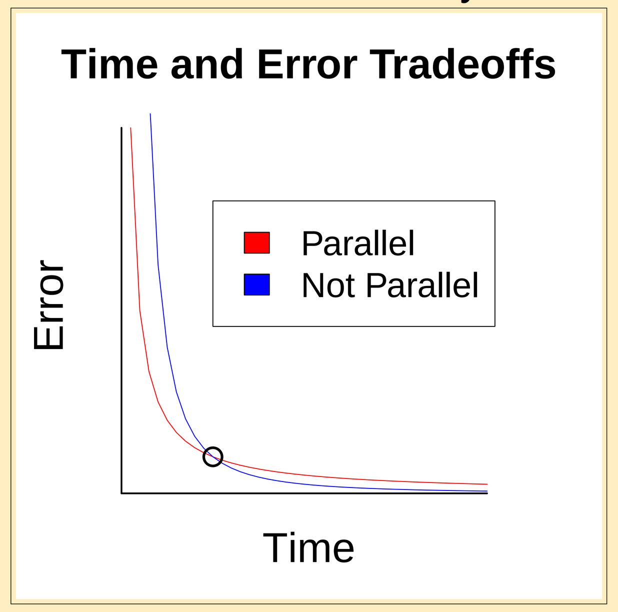


Figure : Time and Error Tradeoffs

  - Human operators manually tweak parameters and partitioning
  - Humans are prone to error and costly to hire!
- Solution: Build an optimization layer to automatically tweak and manage these computations
  - Learn to adjust parameters from past computations
  - Incoming jobs come with budgets of time or accuracy that must be met

Objective

- **Create an optimizer that automatically picks algorithm parameters and the degree of data partitioning to meet budget specifications**

Framework

- Optimizer interface to user: input data, methods to run, and time, error, and financial budgets
- Optimizer then interfaces with matrix algorithms implemented within a parallel framework (such as an algorithm wirtten in SparkR or methods from MLBase)
- All parameter selection hidden from user

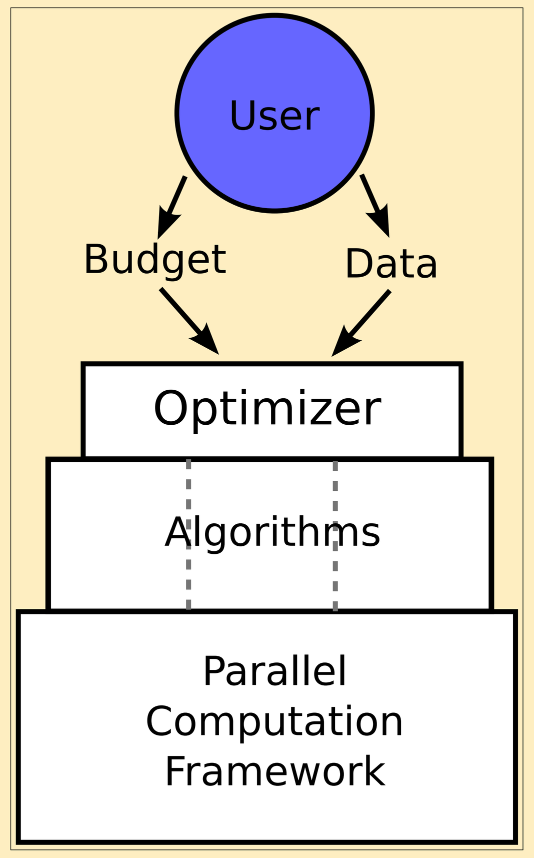


Figure : Our optimizer provides an interface to the user, through which the user specifies the algorithm and a time, error, and monetary budget. The optimizer then calls down to the matrix computation framework with the appropriate parameters.

Optimizer Design

Our optimizer is:

- Architecture-independent
  - Stores statistics from prior jobs on the architecture in question
  - Parameters chosen based on statistics for the specific architecture
- Adaptive
  - Stores statistics from previous jobs on instances with similar distribution
  - Prediction of the optimal parameters improves as the optimizer learns
- Avoids Local Optima
  - When one choice of parameters already encountered is slightly better, there is a risk of getting stuck at a local optimum
  - In "explore mode" we choose the instance parameters randomly, with probability proportional to the relative suitability of the parameters

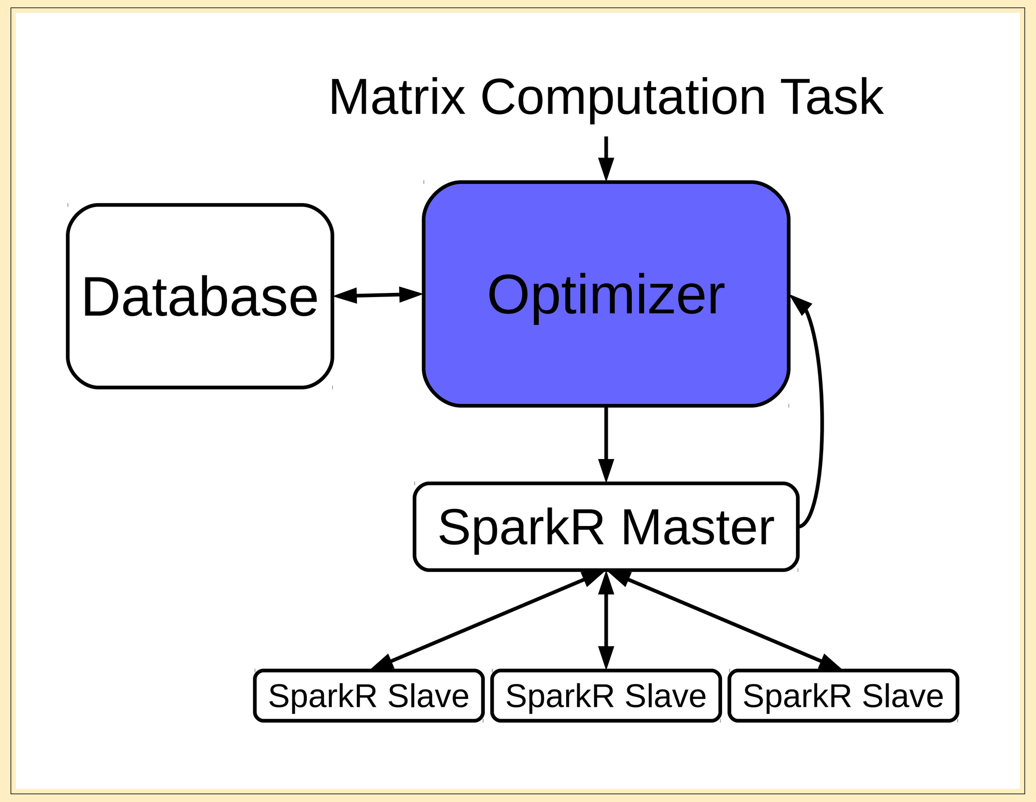


Figure : The control flow of our optimizer. The optimizer looks up relevant information in a database, then interacts with the distributed computation framework (in our current implementation, SparkR).

Implementation

- The matrix computation: Divide-Factor-Combine (DFC)

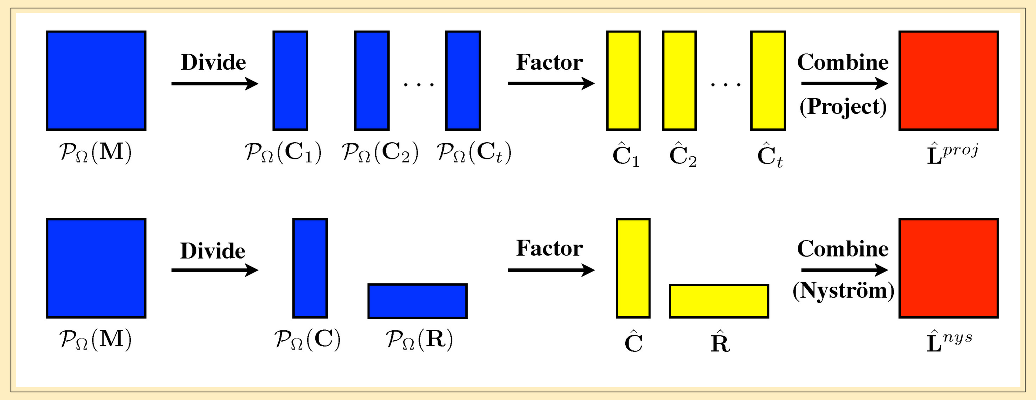


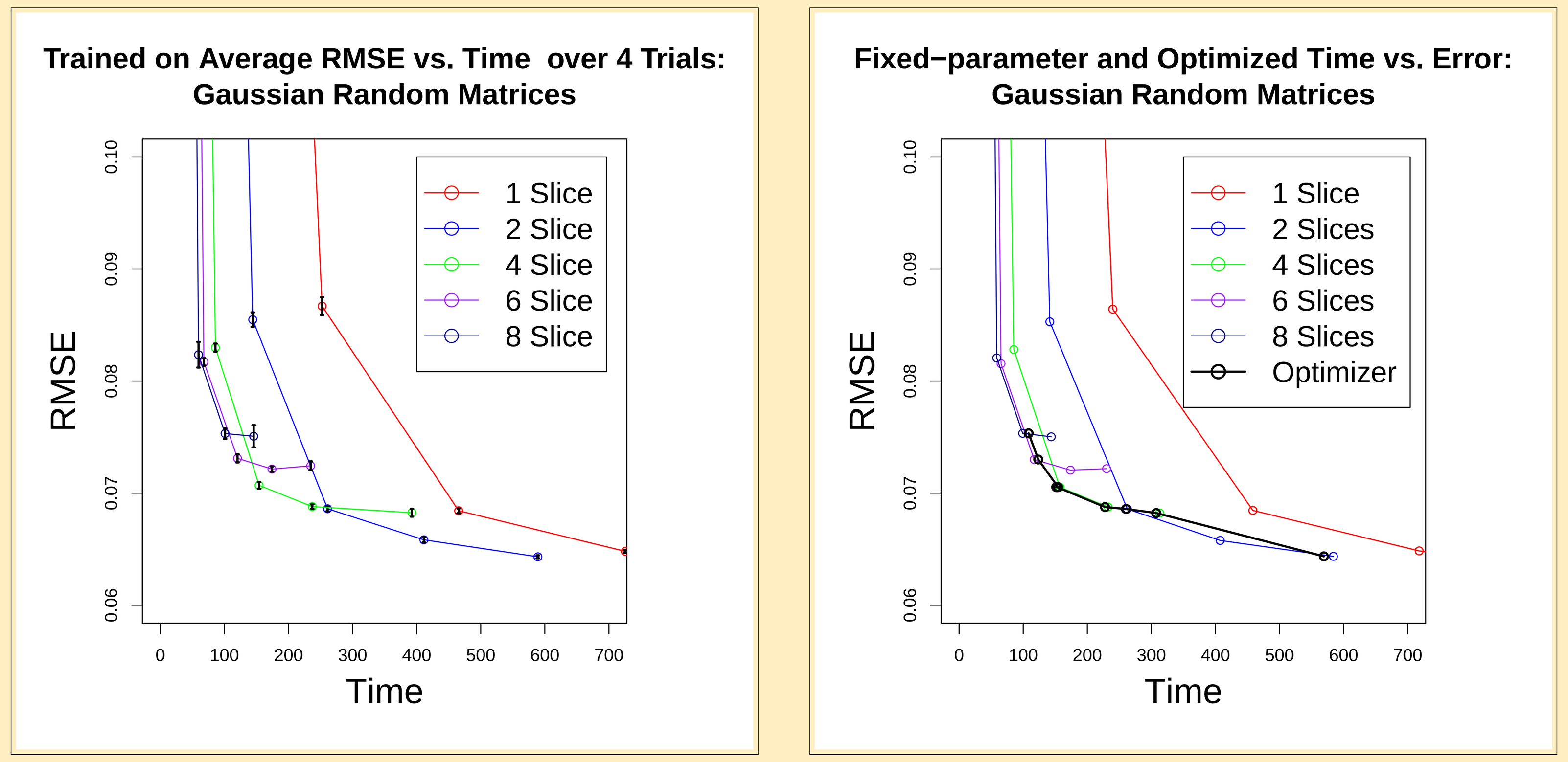
Figure : DFC is a distributed matrix factorization algorithm.

- The parallel computation framework: SparkR
  - An interface for Spark in R
- Implemented DFC to be incorporated into SparkR
- Both randomized and deterministic projection methods
- Two different base factorization methods: APG and SGD

Evaluation

Tested on Synthetic and Real-world Data

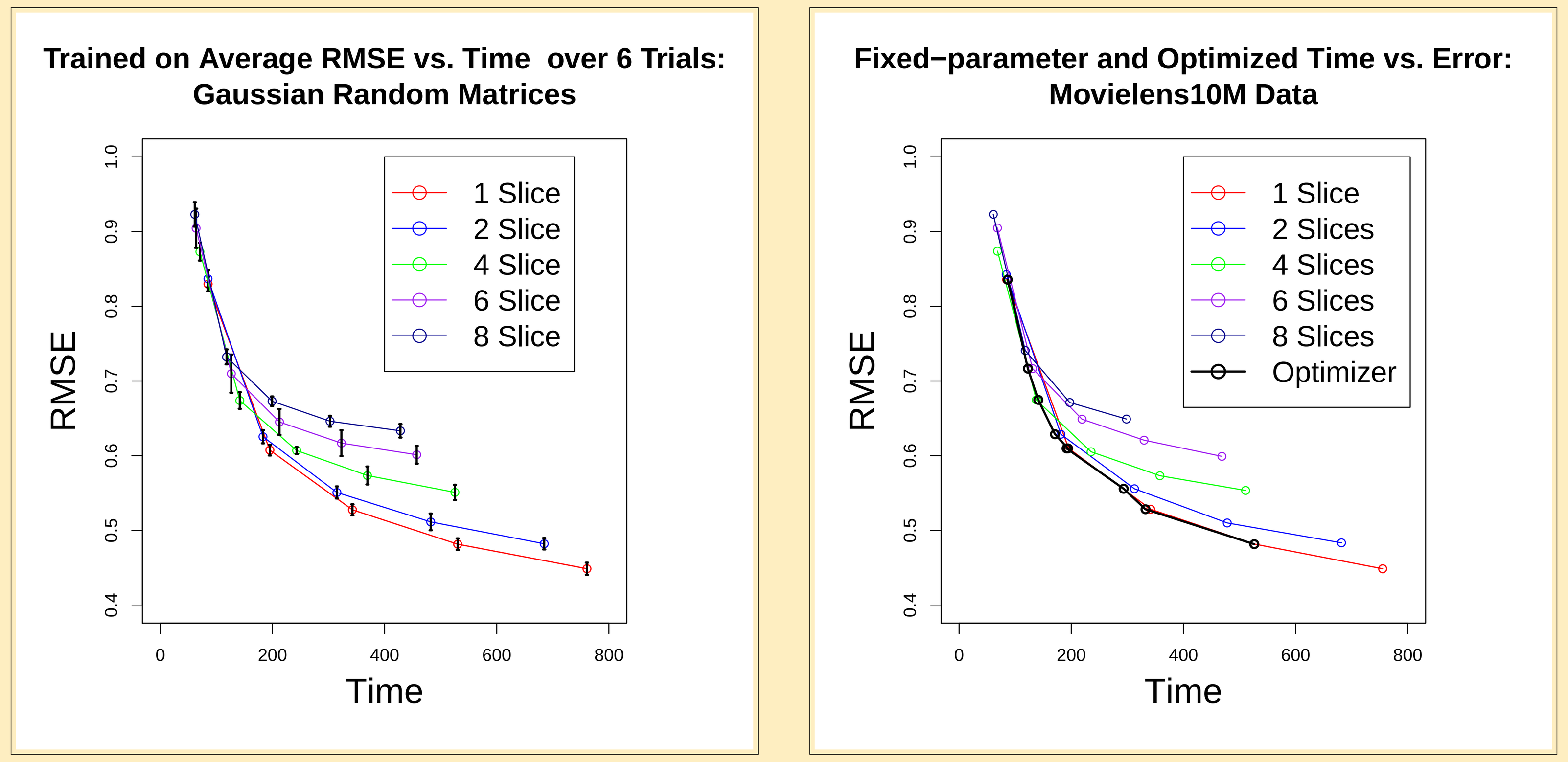
- Gaussian Random Matrices
  - Trained on four random matrices and tested on a fifth



(a) Training Data (b) Testing Data

Figure : Plots of Time and Error taken to factor a 4000 by 4000 matrix drawn from a Gaussian distribution

- MovieLens 10M Dataset
  - Partitioned dataset into 7 parts
  - Trained on all but one partition and tested on the remainder



(a) Training Data (b) Testing Data

Figure : Plots of Time and Error taken to factor a 4000 by 4000 matrix drawn from a Gaussian distribution

- Optimizer performs as well as manually setting the parameter!

Future Work and Acknowledgements

Some more directions for improvements include:

- Optimize over space of algorithms in addition to space of parameters
- Avoid RAM bottlenecks by distributing collect step
- Handle novel or different jobs

We would like to give 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