Performance Estimation of ADMM in a Distributed Environment

Johan Mathe
EE364b: Convex Optimization II Class Project

STANFORD UNIVERSITY

Goal

- Implement **ADMM consensus algorithm** with the **MapReduce** framework and compare the resulting performance with another distributed optimization method. Investigate implementation caveats.

Problem

- Entity disambiguation, *i.e* mapping a text to an ontology concept (*e.g* classification of news articles into various categories like business, politics, *etc.*).
- L^1-regularized multinomial **logistic regression problem**, with m training examples. $l_k(x) \in \mathbf{R}$ loss function for example k. $x \in \mathbf{R}^n$ the feature vector to predict.

$$\underset{x,z}{\text{minimize}} \sum_{k=1}^{m} l_k(x) + \lambda \|x\|_1 \tag{1}$$

ADMM Form

- Separable problem. N number of subproblems. $p = \frac{m}{N}$ is the number of training examples per subproblem. $x_i \in \mathbf{R}^n$ primal vector for subproblem i.

$$L_i(x_i) = \sum_{j=ip}^{ip+p} l_j(x_i) \quad i = 1, \ldots, N$$

- Convert to ADMM global variable consensus problem with L^1 regularization. $z \in \mathbf{R}^n$ is the global consensus variable:

$$\begin{array}{ll} \underset{x_i,z}{\text{minimize}} & \sum_{i=1}^{N} L_i(x_i) + g(z) \\ \text{subject to} & x_i - z = 0 \;\; i = 1, \ldots, N. \end{array}$$

Algorithm and Implementation

- ADMM scaled form. $u_i \in \mathbf{R}^n$ local dual vector. S soft thresholding. $\forall i = 1, \ldots, N$ in parallel:

$$\begin{aligned} u_i &:= u_i + x_i - z \\ x_i &:= \underset{x_i}{\text{argmin}} \left(L_i(x_i) + (\rho/2) \left\| x_i - z + u_i \right\|_2^2 \right) \\ z &:= S_{N\rho}(\bar{x} + \bar{u}) \end{aligned}$$

Wrapper Program

1: converged := false
2: **while** not converged **do**
3: RunMapReduceIteration()
4: $x_i, u_i, \hat{z}$:= ReadFromBigtable()
5: converged := HasConverged($x_i, u_i, \hat{z}$)
6: **end while**

Map Function

1: **function** Map(D_i)
2: $x_i, u_i, \hat{z}$:= ReadFromBigtable(i)
3: $z := S_{N\rho}((1/N)\hat{z})$
4: $u_i := u_i + x_i - z$
5: $x_i := \text{argmin}_x(f_i(x) + (\rho/2)\|x - z + u_i\|_2^2)$
6: Emit(key CENTRAL, record (x_i, u_i))

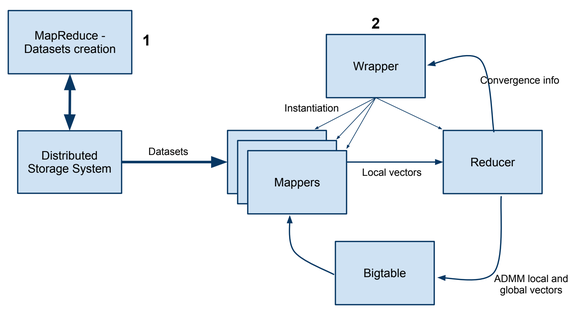
Reduce Runction

1: **function** Reduce(key CENTRAL, records $(x_1, u_1), \ldots, (x_N, u_N)$)
2: $\hat{z} := \sum_{i=1}^{N} x_i + u_i$
3: **for all** i such that $i < N$ **do**
4: S := SerializeToBigtableCell($x_i, u_i, \hat{z}$)
5: WriteToBigtable(i, S)
6: **end for**

Implementation Caveats

- **MapReduce needed to convert training examples to datasets $D_i \in \mathbf{R}^{p \times n}$.** Our implementation: Mapper is the identity. Number of output shard: N. Each reducer accumulates the values, once all values are accumulated, we flush the dataset with the reducer's id as dataset label. Given good sharding function and enough examples, we get good size distribution.
- **Original algorithm could only write to one single Bigtable row.** This method does not scale because rows are limited in size. Therefore we write subproblem information per bigtable row, with label i. This leads to more efficient distribution in the database.
- **No warm start.** It leads to numerical instability: Wolfe's conditions are not satisfied anymore (thought: we are approaching a flat region which leads to an ill-conditioned Hessian matrix).

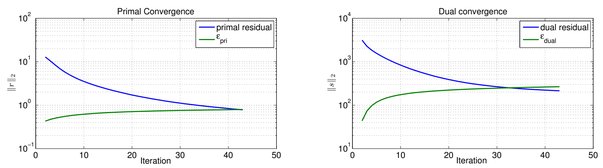
Big Picture



Tests

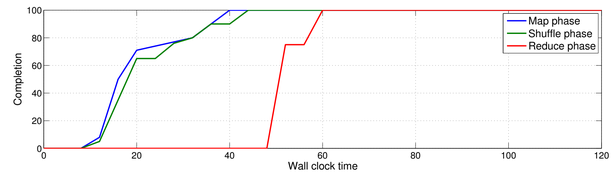
- 560MiB Dataset, 4M examples, 30K features, 200 subproblems.
- Each subproblem is a 20K $\times$ 30K size problem. Example density: 0.01%.
- Evaluate result vector with a testing set T. Success rate is the ratio of properly predicted examples over total examples in T.

Results



Method	Success rate	CPU time	Memory	Net	Time	Shards	Iterations
ADMM	0.8309	46 h	57.34GiB.h	29.9GiB	92 min	200	44
L-BFGS	0.8407	14 h	28.33GiB.h	31.95GiB	17 min	50	50

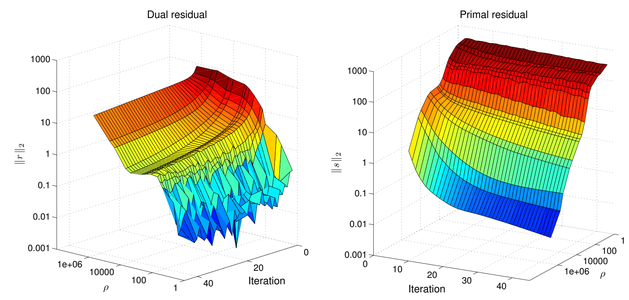
Single Iteration Details



CPU time	Input data	Memory usage	Reduce Input	Time
3731s	619MiB	5161GiB.s	31.2MiB	122s

Convergence *vs* ρ

- Each data point of the graph is a MapReduce, normalized by stopping criteria.
- ρ too small ($< 10^2$): poor primal convergence and instability due to lack of regularization.
- ρ too big ($> 10^4$): poor dual convergence.



Conclusion

Successful implementation of ADMM consensus problem for a multinomial L^1 regularized logistic regression problem. Appropriate ρ is primordial for a reasonable convergence time, and setting ρ to a small value prevents L-BFGS to converge because of a lack of L^2 regularization. Global performance of our implementation is not as good as a direct (distributed) L-BFGS method, but it is in the same order of magnitude. Further optimizations like warm start and the use of MapReduce combiners will greatly improve performance.