Optimistic Concurrency Control for Distributed Learning
Xinghao Pan, Joseph E. Gonzalez, Stefanie Jegelka, Tamara Broderick, Michael I. Jordan

Distributed Machine Learning

Larger datasets have pushed the Machine Learning community to explore parallel / distributed algorithms.

Challenge: dividing and coordinating computation across cores / machines

Model parameters

Data

Two prior approaches to parallelize algorithm:
1. Mutual exclusion: Serializable but costly locking
2. Coordination free: Low contention but possible data corruption

Concurrency: most updates in parallel
Correctness: result equivalent to some serial execution of algorithm

Objective: Provide high concurrency & correctness, through optimistic concurrency control

Optimistic Concurrency Control

View as a transactional model: Transaction \( \otimes \) Operation
1. Read shared state and data
2. Validation: detect conflicts
3. Resolution: fix conflicts

Optimistic Concurrency Control (OCC):
- Assume low conflict, proceed optimistically, serially validate when needed
- Standard OCC validate by comparing read- and write-sets, reject on conflict

\( \square \) Rare conflicts, proceed optimistically \( \rightarrow \) High Concurrency
\( \square \) Validation and Resolution mechanism \( \rightarrow \) Correctness

Example: Distributed Clustering

DP-means: Novel clustering algorithm
- Extends popular K-means approach
- Groups similar data together without need to specify # of clusters

Serial algorithm
1. Read data \( x \), and set of clusters, represented by centers \( \{\mu_i\} \)
2. Compute distance \( d_{ij} \) of \( x \) to each center \( \mu_i \)
3. If \( d_{ij} < \lambda \), assign \( x \) to nearest center; Otherwise, create new cluster with center at \( x \)

Transaction \( T \): for each data object \( x \):
1. Read cluster centers \( \{\mu_i\} \)
2. Compute distance \( d_{ij} \) of \( x \) to each center \( \mu_i \)
3. If \( d_{ij} < \lambda \), assign \( x \) to nearest center, commit immediately; Otherwise, create new cluster with center at \( x \), validate

Validate cluster creation for \( x \):
1. Read cluster centers \( \{\mu_i\} \) created since read phase of \( T_i \)
2. Compute distance \( d_{ij} \) of \( x \) to each center \( \mu_i \)
3. If \( d_{ij} < \lambda \), resolve: assign \( x \) to nearest new center; Otherwise, accept: create new cluster with center at \( x \)

Example: Feature Modeling

BP-means: Novel feature clustering algorithm
- Allows multiple cluster (feature) membership
- Each data object represented as sum of features

Serial algorithm
1. Read current set of features \( \{f_j\} \)
2. Find best representation \( x_i = \sum z_i f_j \) where \( z_i \neq 0 \) or 1
3. If distance \( x_i \) to \( \sum z_i f_j \leq \lambda \), assign \( x_i \) otherwise; create new feature \( z_i \), validate

Validate feature creation for \( f \):
1. Read features \( \{f_j\} \) created since read phase of \( T \)
2. Find best representation \( f_{\text{new}} = \sum z_i f_j \) where \( z_i \neq 0 \) or 1
3. If distance \( f_{\text{new}} \) to \( \sum z_i f_j \leq \lambda \), resolve: represent \( f \) as \( \sum z_i f_j \); Otherwise, accept: create new feature \( f_{\text{new}} = \sum z_i f_j \)

Theorem 1a: Distributed DP-means is serially equivalent to DP-means.
Theorem 1b: Expected number of data points sent for validation is less than \( P h + E[K] \lambda \), where \( P \) is the number of processors, \( h \) is the number of data points processed by each processor in 1 iteration, and \( K \) is the number of clusters.

Experiments

Matlab simulation of distributed DP-means and BP-means algorithm:
- # data points proposed but rejected as clusters independent of dataset size
- Constant additional work compared to serial execution!

Amazon EC2 experiment with 134 million data points, and 1,2,4,8 machines:
- 2x # machines decreases the run time by \( \frac{1}{2} \)
- Close to perfect scaling up to 8 machines