Optimistic Concurrency Control for Distributed Learning

Xinghao Pan, Joseph Gonzalez, Stefanie Jegelka, Tamara Broderick, Michael I. Jordan

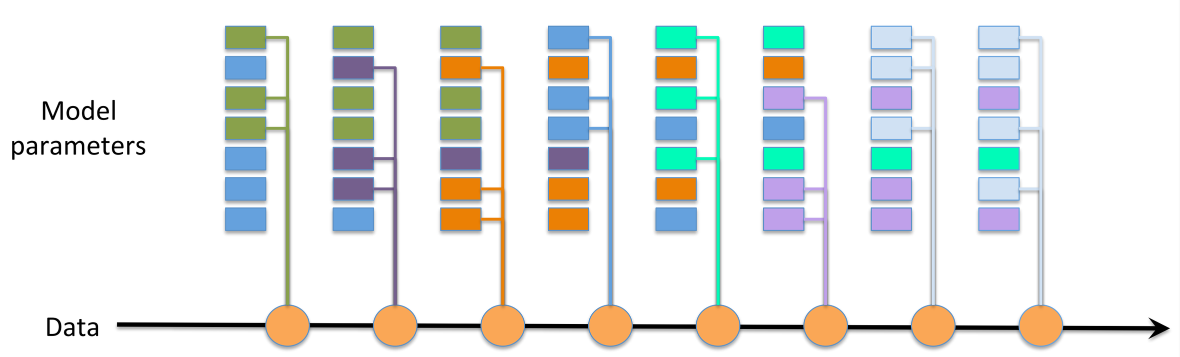
https://amplab.cs.berkeley.edu/projects/ccml/

Distributed Machine Learning

Big Data: need for distributed machine learning algorithms.

Challenge: dividing and coordinating computation across cores / machines

Algorithms access data and serially update shared state



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

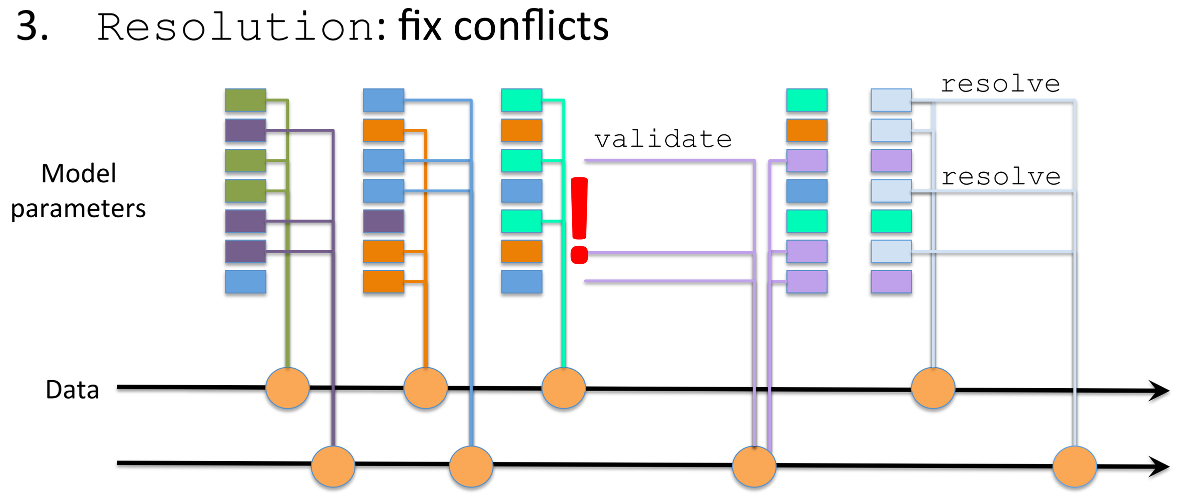
Concurrency: most updates in parallel
→ fast algorithms

Correctness: result equivalent to some serial execution
→ preserve theoretical properties

Objective: Provide high concurrency & correctness, through <u>optimistic concurrency control</u>

Optimistic Concurrency Control

View as a **transactional model**: Transaction ⇔ Operation
1. Read shared state and data
2. `Validation`: detect conflicts
3. `Resolution`: fix conflicts



Optimistic Concurrency Control (OCC) [1]:
- Assume low conflict rate, proceed optimistically, serially validate when needed
- Standard OCC validate by comparing read- and write-sets, reject on conflict

❑ Rare conflicts, proceed optimistically
→ High Concurrency

❑ Validation and Resolution mechanism
→ Correctness

Example: Distributed Clustering

DP-means: Novel clustering algorithm [2]
- Extends popular K-means approach
- Cluster data without need to specify # of clusters
- Small variance asymptotic approx. to Dirichlet Process

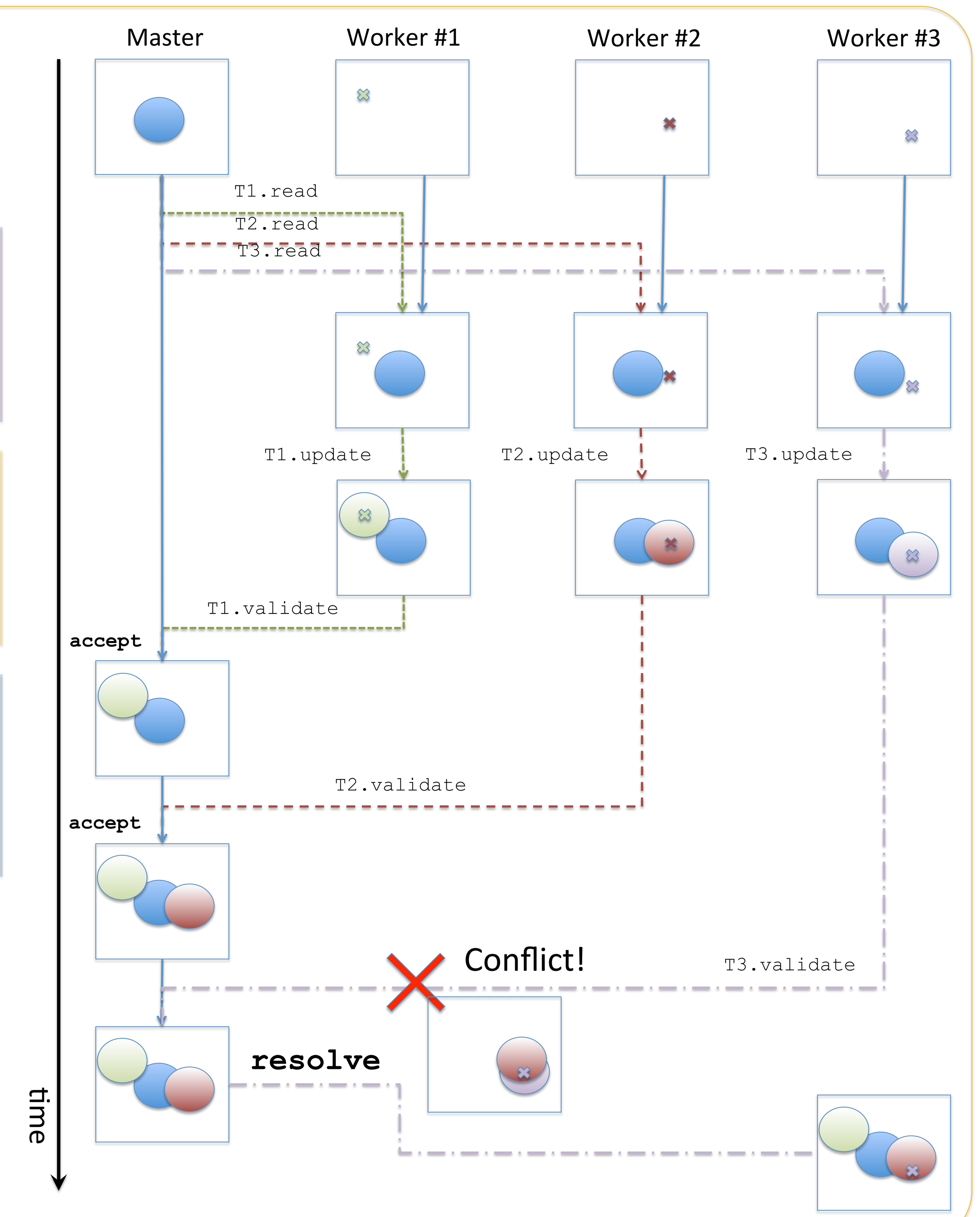
<u>Serial algorithm</u>
1. Read data x_i and set of clusters, represented by centers $\{\mu_c\}$
2. Compute distance $d = \min_c \|x_i - \mu_c\|^2$ of x_i to centers $\{\mu_c\}$
3. If $d < \lambda$, assign x_i to nearest center;
   Otherwise, create new cluster with center at x_i

<u>Transaction T_i for each data object x_i:</u>
1. **Read** cluster centers $\{\mu_c\}$
2. Compute distance $d = \min_c \|x_i - \mu_c\|$ of x_i to centers $\{\mu_c\}$
3. If $d < \lambda$, assign x_i to nearest center, **commit** immediately;
   Otherwise, create new cluster with center at x_i, **validate** x_i

<u>Validate cluster creation for x_i</u>
1. Read cluster centers $\{\mu_k\}$ created since read phase of T_i
2. Compute distance $d^* = \min_k \|x_i - \mu_k\|^2$ of x_i to centers $\{\mu_k\}$
3. If $d^* < \lambda$, **resolve**: assign x_i to nearest new center;
   Otherwise, **accept**: create new cluster with center at x_i

Theorem 1a: Distributed DP-means is serially equivalent to DP-means.

Theorem 1b: Expected number of data points sent for validation is less than $Pb + \mathbf{E}[K]$, where P is the number of processors, b is the number of data points processed by each processor in one iteration, and K is the number of clusters.



Example: Feature Modeling

BP-means: Novel feature clustering algorithm [3]
- Allows membership in multiple clusters (features)
- Each data object represented as sum of features
- Small variance asymptotic approximation of Beta Process

<u>Serial algorithm</u>
1. Read current set of features $\{f_j\}$
2. Find best representation $x_i \approx \Sigma_j z_{ij} f_j$, where z_{ij} = 0 or 1
3. If distance x_i to $\Sigma_j z_{ij} f_j < \lambda$, assign representation for x_i
   Otherwise, create new feature $x_i - \Sigma_j z_{ij} f_j$

<u>Transaction T_i for each data object x_i:</u>
1. **Read** current set of features $\{f_j\}$
2. Find best representation $x_i \approx \Sigma_j z_{ij} f_j$, where z_{ij} = 0 or 1
3. If distance x_i to $\Sigma_j z_{ij} f_j < \lambda$, **commit** immediately;
   Otherwise, create new feature $f_i^{new} = x_i - \Sigma_j z_{ij} f_j$, **validate** f_i^{new}

<u>Validate feature creation for f_i^{new}</u>
1. Read features $\{f_k\}$ created since read phase of T_i
2. Find best representation $f_i^{new} \approx \Sigma_k z_{ik} f_k$, where z_{ik} = 0 or 1
3. If distance f_i^{new} to $\Sigma_k z_{ik} f_k < \lambda$,
   resolve: represent $x_i \approx \Sigma_j z_{ij} f_j + \Sigma_k z_{ik} f_k$;
   Otherwise, **accept**: create new feature f_i^{new} - $\Sigma_k z_{ik} f_k$

Theorem 2: Distributed BP-means is serially equivalent to BP-means.

Example: Online Facility Location

Online Facility Location (OFL):
- Select facilities to min objective $\Sigma_i \min_c \|x_i - \mu_c\|^2 + \lambda^2 |\{\mu_c\}|$
- Stochastically choose data point x as facility in single pass

<u>Serial algorithm</u>
1. Read data x_i and set of facilities, represented by centers $\{\mu_c\}$
2. Compute distance $d = \min_c \|x_i - \mu_c\|^2$ of x_i to centers $\{\mu_c\}$
3. With probability $1\text{-}\min(1, d^2/\lambda^2)$, assign x_i to nearest facility;
   Otherwise, create facility at x_i

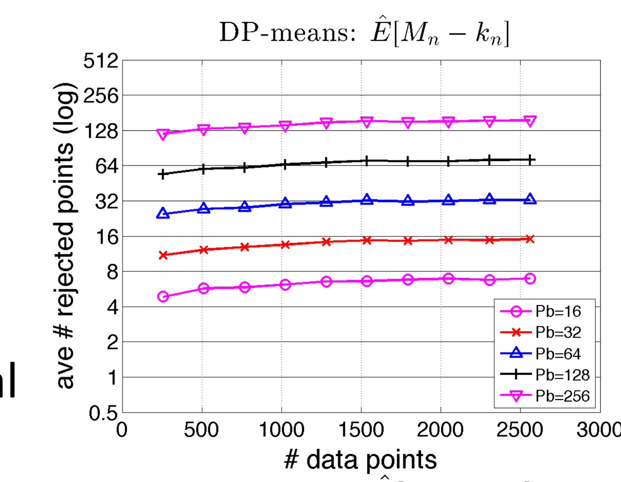
<u>Transaction T_i for each data object x_i:</u>
1. **Read** facility centers $\{\mu_c\}$
2. Compute distance $d = \min_c \|x_i - \mu_c\|^2$ of x_i to centers $\{\mu_c\}$
3. w.p. $1\text{-}\min(1, d^2/\lambda^2)$, assign x_i to nearest facility, **commit**;
   Otherwise, create new facility at x_i, **validate** (x_i, d)

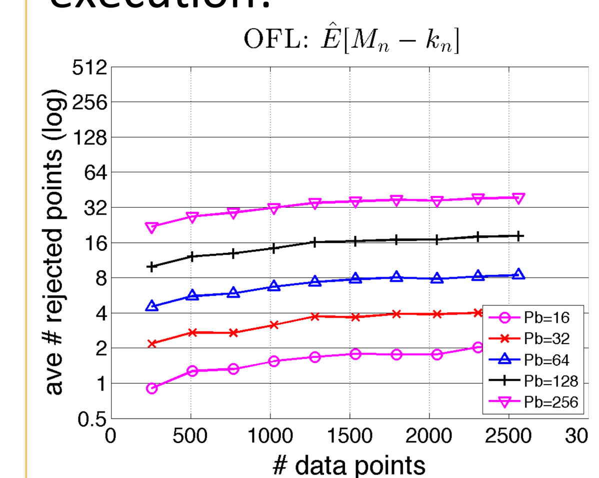
<u>Validate facility creation for (x_i, d)</u>
1. Read facility centers $\{\mu_k\}$ created since read phase of T_i
2. Compute distance $d^* = \min_k \|x_i - \mu_k\|^2$ of x_i to centers $\{\mu_k\}$
3. w.p. $1\text{-}\min(1, d^{*2}/d^2)$, **resolve**: assign x_i to nearest new facility;
   Otherwise, **accept**: create new facility at x_i

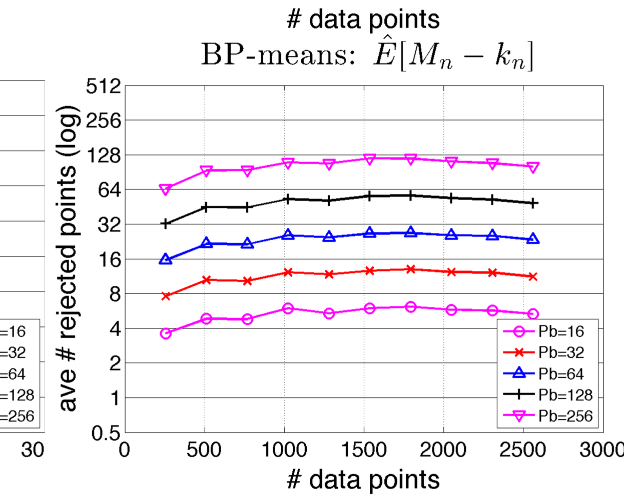
Theorem 3a: Distributed OFL is serially equivalent to OFL.
Corollary 3b: If the data is randomly ordered, then the distributed OFL algorithm provides a constant-factor approximation for the DP-means objective.

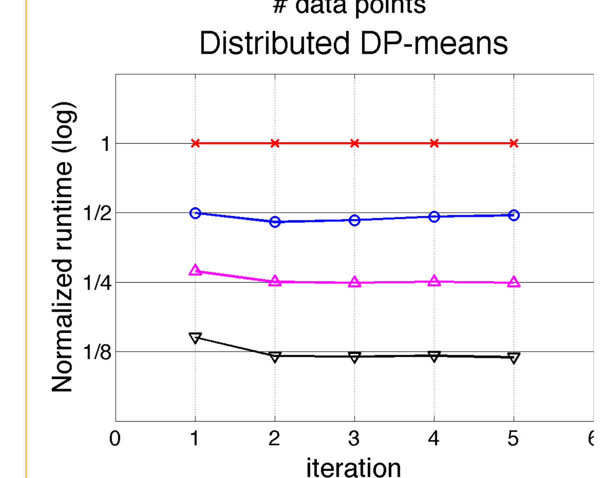
Experiments

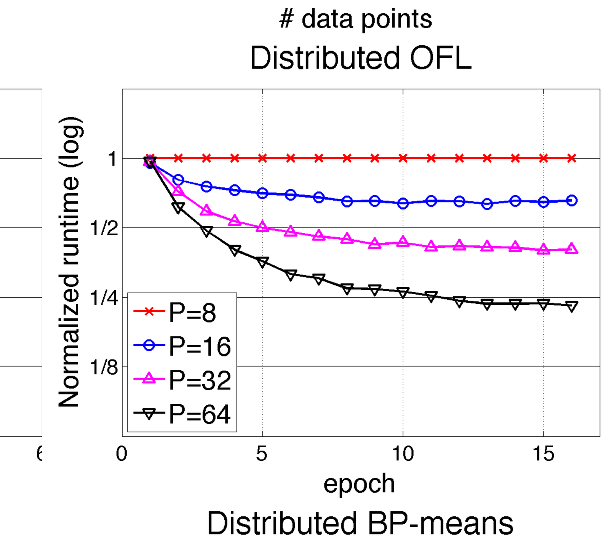
Matlab simulation of distributed DP-means and BP-means algorithm:
data points proposed but rejected as clusters independent of dataset size
→ Constant additional work compared to serial execution!



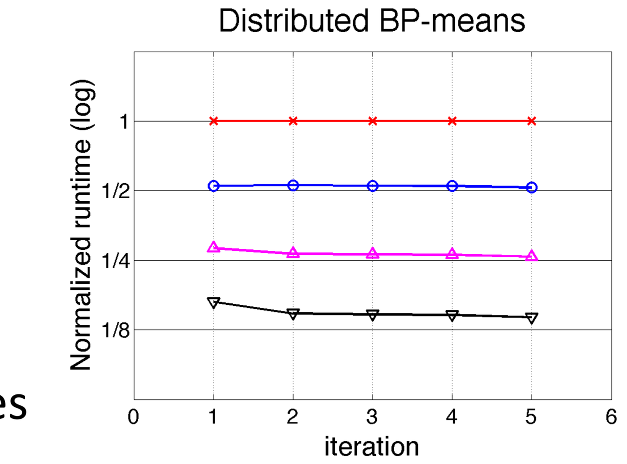








Amazon EC2 experiment with 134 million data points, and 1,2,4,8 machines: 2x # machines decreases the run time by ½
→ Close to perfect scaling up to 8 machines



Discussion & Future Work

Optimistic Concurrency Control can be usefully employed in design of distributed machine learning algorithms
→ Preserves correctness, theoretical properties
→ Provides high concurrency and parallelism

<u>Future work</u>
- Probabilistic acceptance / validation preserving statistical correctness and invariants
- Extension to other distributed ML algorithms
- LDA / HDP Collapsed Gibbs sampling
- Submodular maximization

References

[1] Hsiang-Tsung Kung and John T Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.
[2] Brian Kulis and Michael I. Jordan. Revisiting k-means: New algorithms via Bayesian nonparametrics. In *Proceedings of 23rd International Conference on Machine Learning*, 2012.
[3] Tamara Broderick, Brian Kulis and Michael I. Jordan. MAD-Bayes: MAP-based Asymptotic Derivations from Bayes. *Proceedings of the 30th International Conference on Machine Learning*, 2013.