Strategies & Principles for Distributed Machine Learning

Eric Xing
epxing@cs.cmu.edu
School of Computer Science
Carnegie Mellon University

Acknowledgement:
Wei Dai, Qirong Ho, Jin Kyu Kim, Abhimanu Kumar, Seunghak Lee, Jinliang Wei, Pengtao Xie, Yaoliang Yu, Hao Zhang, Xun Zheng, James Cipar, Henggang Cui,
and, Phil Gibbons, Greg Ganger, Garth Gibson
Machine Learning: 
-- a view from outside
Inside ML …

- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Deep Learning
- Sparse Coding
- Spectral/Matrix Methods
- Sparse Structured I/O Regression
- Large-Margin

Network switches
Infiniband
Network attached storage
Flash storage
Server machines
GPUs
Desktops/Laptops
NUMA machines
Cloud compute
(e.g. Amazon EC2)
Virtual Machines
Hardware and infrastructure
An ML Program

\[
\arg\max_{\mathbf{\theta}} \equiv \mathcal{L}(\{x_i, y_i\}_{i=1}^N ; \mathbf{\theta}) + \Omega(\mathbf{\theta})
\]

Solved by an iterative convergent algorithm

\[
\text{for } (t = 1 \text{ to } T) \{
\text{doThings()}
\hat{\mathbf{\theta}}_{t+1} = g(\hat{\mathbf{\theta}_t}, \Delta f \hat{\mathbf{\theta}}(\mathcal{D}))
\text{doOtherThings()}
\}
\]

This computation needs to be parallelized!
Massive Data

Facebook
1B+ USERS
30+ PETABYTES

Wikipedia
32 million pages

YouTube
100+ hours video uploaded every minute

twitter
645 million users
500 million tweets / day
Growing Model Complexity

Google Brain Deep Learning for images: 1~10 Billion model parameters

Multi-task Regression for simplest whole-genome analysis: 100 million ~ 1 Billion model parameters

Topic Models for news article analysis: Up to 1 Trillion model parameters

Collaborative filtering for Video recommendation: 1~10 Billion model parameters
Challenge #1
– Massive Data Scale

Familiar problem: data from 50B devices, data centers won’t fit into memory of single machine

\[
\Delta f^\theta(D) \\
\text{for } (t = 1 \text{ to } T) \{ \\
\quad \text{doThings()} \\
\quad \text{parallelUpdate(x, \theta)} \\
\quad \text{doOtherThings()} \\
\} 
\]
Challenge #2
– Gigantic Model Size

Big Data needs Big Models to extract understanding

But ML models with >1 trillion params also won’t fit!
Issues with Hadoop and I-C ML Algorithms

Naïve MapReduce not best for ML

- Hadoop can execute iterative-convergent, data-parallel ML...
  - map() to distribute data samples $i$, compute update $\Delta(D_i)$
  - reduce() to combine updates $\Delta(D_i)$
  - Iterative ML algo = repeat map()+reduce() again and again
- But reduce() writes to HDFS before starting next iteration’s map() - very slow iterations!
Spark: Faster MapR on Data-Parallel

- Spark’s solution: **Resilient Distributed Datasets (RDDs)**
  - Input data → load as RDD → apply transforms → output result
  - RDD transforms strict superset of MapR
  - RDDs cached in memory, avoid disk I/O

- **Spark ML library supports data-parallel ML algos, like Hadoop**
  - Spark and Hadoop: comparable first iter timings…
  - But Spark’s later iters are much faster

Source: ebaytechblog.com
Parallelization Strategies

Usually, we worry ...

A sequential program

\[ \beta_1 \rightarrow \beta_2 \]

A parallel program

\[ \beta_1 \rightarrow \beta_2 \]

but assuming an ideal system, e.g.,
- zero-cost sync,
- zero-cost fault recovery
- uniform local progress
- ...

for \( t = 1 \) to \( T \) {
    doThings()
    parallelUpdate(x, \theta)
    doOtherThings()
}

Low bandwidth, High delay

Sync

Unequal performance
Intrinsic Properties of ML Programs

[Xing et al., 2015]

- ML is optimization-centric, and admits an iterative convergent algorithmic solution rather than a one-step closed form solution.
- Error tolerance: often robust against limited errors in intermediate calculations.
- Dynamic structural dependency: changing correlations between model parameters critical to efficient parallelization.
- Non-uniform convergence: parameters can converge in very different number of steps.

Whereas traditional programs are transaction-centric, thus only guaranteed by atomic correctness at every step.

How do existing Big Data platforms fit the above?
4 Principles of ML System Design

How to execute distributed-parallel ML programs?
ML program equations tell us “What to Compute”. But…

1. How to Distribute?
2. How to Bridge Computation and Communication?
3. How to Communicate?
4. What to Communicate?
Principles of ML system Design [Xing et al., to appear 2016]

1. How to Distribute: Scheduling and Balancing workloads
Example: Model Distribution

Lasso via coordinate descent:

\[
\min_{\beta} \| y - X\beta \|^2_2 + \lambda \sum_j |\beta_j|
\]

A huge number of parameters (e.g.) \( M > 100 \) million

- How to correctly divide computational workload across workers?
- What is the best order to update parameters?
Model Dependencies

- Concurrent updates of $\beta$ may induce errors

Sequential updates

\[ \beta_1 \]
\[ \beta_2 \]

Concurrent updates

\[ \beta_1 \]
\[ \beta_2 \]
\[ \beta_1 \]
\[ \beta_2 \]

Sync

Decreases iteration progress

\[ \beta_1^{(t)} \leftarrow S(x_1^T y - x_1^T x_2 \beta_2^{(t-1)}, \lambda) \]

Need to check $x_1^T x_2$ before updating parameters
Avoid Dependency Errors via Structure-Aware Parallelization (SAP)

[Lee et al., 2014] [Kim et al., 2016]

- Smart model-parallel execution:
  - Structure-aware scheduling
  - Variable prioritization
  - Load-balancing

- Simple programming:
  - Schedule()
  - Push()
  - Pull()

Example code:

```cpp
void schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```

```cpp
void push(worker = p, vars = (x[j_1], ..., x[j_U])) {  
  // Compute partial update z for U vars x[j]
  // at worker p
  ...
  return z
}
```

```cpp
void pull(workers = [p], vars = (x[j_1], ..., x[j_U]),
          updates = [z]) {  
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
```
A Structure-aware Dynamic Scheduler (Strads) [Lee et al., 2014] [Kim et al, 2016]

**Strads System**

1. Partition Data + Model into Tasks
2. Schedule & Prioritize Tasks onto Workers
3. Balance Task Load on each Worker

- Priority Scheduling
  \[
  \{\beta_j\} \sim \left(\delta \beta_j^{(t-1)}\right)^2 + \eta
  \]

- Block scheduling

**Worker 1**

**Worker 2**

**Worker 3**

**Worker 4**

Round 1 | Round 2 | Round 3 | Round 4
---|---|---|---

**Sync. barrier**

[SAP]

[Kumar, Beutel, Ho and Xing, Fugue: Slow-worker agnostic distributed learning, AISTATS 2014]
SAP Scheduling: Faster, Better Convergence across algorithms

- SAP on Strads achieves better speed and objective
SAP gives Near-Ideal Convergence Speed [Xing et al., 2015]

**Goal:** solve sparse regression problem
- Via coordinate descent over “SAP blocks” $X^{(1)}, X^{(2)}, \ldots, X^{(B)}$
  - $X^{(b)}$ are data columns (features) in block $(b)$
- $P$ parallel workers, $M$-dimensional data
- $\rho = \text{Spectral Radius}[[\text{BlockDiag}[[X^{(1)}]^T X^{(1)}, \ldots, (X^{(t)})^T X^{(t)}]]];$ this block-diagonal matrix quantifies max level of correlation within all SAP blocks $X^{(1)}, X^{(2)}, \ldots, X^{(t)}$

**SAP converges according to**

\[
\mathbb{E} \left[ f(X^{(t)}) - f(X^*) \right] \leq \frac{\mathcal{O}(M)}{P - \frac{\mathcal{O}(P^2 \rho)}{M}} \frac{1}{t} = \mathcal{O} \left( \frac{1}{Pt} \right)
\]

where $t$ is # of iterations

**Take-away:** SAP minimizes $\rho$ by searching for feature subsets $X^{(1)}, X^{(2)}, \ldots, X^{(B)}$ w/o cross-correlation => as close to $P$-fold speedup as possible
YahooLDA progress per iteration

- YahooLDA attains near-ideal throughput (1→3.8x)...
- … but progress per iteration gets worse with more machines

- YahooLDA only <2x speedup from 25 →100 machines
  - 6.7x slower compared to SAP-LDA

<table>
<thead>
<tr>
<th></th>
<th>YahooLDA data throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 machines</td>
<td>39.7 M/s (1x)</td>
</tr>
<tr>
<td>50 machines</td>
<td>78 M/s (1.96x)</td>
</tr>
<tr>
<td>100 machines</td>
<td>151 M/s (3.8x)</td>
</tr>
</tbody>
</table>

80GB data, 2M words, 1K topics, 100 machines
Correctly Measuring Parallel Performance [blinded, to appear]

SAP-LDA progress per iteration

80GB data, 2M words, 1K topics, 100 machines

<table>
<thead>
<tr>
<th></th>
<th>SAP-LDA data throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 machines</td>
<td>58.3 M/s (1x)</td>
</tr>
<tr>
<td>50 machines</td>
<td>114 M/s (1.96x)</td>
</tr>
<tr>
<td>100 machines</td>
<td>204 M/s (3.5x)</td>
</tr>
</tbody>
</table>

- **Ideal rate**: progress per iter preserved from 25 → 100 machines
  - Thanks to dependency checking

- **Near-ideal throughput**: data rate 1x → 3.5x from 25→100 machines
  - Thanks to load balancing

- **Convergence Speed = rate x throughput**
  - Therefore near-ideal 3.5x speedup from 25→100 machines
Principles of ML system Design [Xing et al., to appear 2016]

2. How to Bridge Computation and Communication: 
Bridging Models and Bounded Asynchrony
The Bulk Synchronous Parallel Bridging Model

- Perform barrier in order to communicate parameters
- Mimics sequential computation – “serializable” property
- Enjoys same theoretical guarantees as sequential execution
The success of the von Neumann model of sequential computation is attributable to the fact it is an efficient bridge between software and hardware... an analogous bridge is required for parallel computation if that is to become as widely used – Leslie G. Valiant

- Numerous implementations since 90s (list by Bill McColl):
  - Oxford BSP Toolset (’98), Paderborn University BSP Library (’01), Bulk Synchronous Parallel ML (’03), BSPonMPI (’06), ScientificPython (’07), Apache Hama (’08), Apache Pregel (’09), MulticoreBSP (’11), BSPedupack (’11), Apache Giraph (’11), GoldenOrb (’11), Stanford GPS Project (’11) …
But There Is No Ideal Distributed System!

- Two distributed challenges:
  - Networks are slow
  - “Identical” machines rarely perform equally

Result: BSP barriers can be slow
Is there a better way to interleave computation and communication?

- Safe/slow (BSP) vs. Fast/risky (Async)?

- Challenge 1: Need “Partial” synchronicity
  - Spread network comms evenly (don’t sync unless needed)
  - Threads usually shouldn’t wait – but mustn’t drift too far apart!

- Challenge 2: Need straggler tolerance
  - Slow threads must somehow catch up

Is persistent memory really necessary for ML?
A Stale Synchronous Parallel Bridging Model [Ho et al., 2013]

Stale Synchronous Parallel (SSP)
- Fastest/slowest workers not allowed to drift $>\delta$ iterations apart

Consequence
- Fast like async, yet correct like BSP
- Why? Workers’ local view of model parameters “not too stale” ($\leq \delta$ iterations old)

Force stop worker 1 until worker 2 catches up
Data-Parallel
Proximal Gradient under SSP

- Model (e.g. SVM, Lasso ...):

\[ \min_{a \in \mathbb{R}^d} \mathcal{L}(a, D), \quad \text{where} \quad \mathcal{L}(a, D) = f(a, D) + g(a) \]

- Algorithm:
  - Update
    \[ a(t) := \text{prox}_g \left( a^p(t) - \eta(t) \sum_{(p', t') \in \text{Recv}_p(t)} \Delta(a^{p'}(t'), D_{p'}) \right) \]
  - sub-update
    \[ \Delta(a^p(t), D_p) := \nabla f(a^p(t), D_p) \]

- Data parallel:
  - Data \( D \) too large to fit in a single worker, divide among \( P \) workers

\[ \quad \text{split} \quad \xrightarrow{\text{Input Data}} \quad \text{Update local copy of ALL params} \quad \xrightarrow{\text{aggregate}} \quad \text{Update ALL params} \]
SSP Data-Parallel
Async Speed, BSP Guarantee

- Massive **Data** Parallelism
- Effective across different algorithms
Theorem: Given L-Lipschitz objective $f_t$ and step size $h_t$,

\[
P \left[ \frac{R[X]}{T} - \frac{\mathcal{O}(F^2 + \mu_\gamma L^2)}{\sqrt{T}} \geq \tau \right] \leq \exp \left\{ \frac{-T \tau^2}{\mathcal{O} (\tilde{\eta} T \sigma_\gamma) + L^2 s P \tau} \right\}
\]

where

\[
R[X] := \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*) \quad \tilde{\eta}_T = \frac{\eta^2 L^4 (\ln T + 1)}{T} = o(T)
\]

Explanation: the distance between true optima and current estimate decreases exponentially with more SSP iterations. Lower staleness mean, variance $\mu_\gamma$, $\sigma_\gamma$ improve the convergence rate.
Model-Parallel Proximal Gradient under SSP

- Model (e.g. SVM, Lasso ...):
  \[
  \min_{a \in \mathbb{R}^d} \mathcal{L}(a, D), \quad \text{where} \quad \mathcal{L}(a, D) = f(a, D) + g(a)
  \]

- Model parallel
  - Model dimension \(d\) too large to fit in a single worker
  - Divide model among \(P\) workers \(a = (a_1, a_2, \ldots, a_P)\)

- Algorithm:
  \[
  \forall p, \quad a_p(t + 1) = a_p(t) + \gamma_p(t) \cdot F_p(a^p(t))
  \]
  \[
  = a_p(0) + \sum_{k=0}^{t} \gamma_p(k) \cdot F_p(a^p(t))
  \]

  (local) \(a^p(t) = (a_1(\tau_1^p(t)), \ldots, a_P(\tau_P^p(t)))\)

  (global) \(a(t) = (a_1(t), \ldots, a_P(t))\).

- worker \(p\) keeps local copy of the full model (can be avoided for linear models)
SSP Model-Parallel
Async Speed, BSP Guarantee

**Lasso: 1M samples, 100M features, 100 machines**

- Massive **Model** Parallelism
- Effective across different algorithms

Curves overlap – no compromise to quality
SSP Model Parallel Convergence Theorem

Theorem: Given that the SSP delay is bounded, with appropriate step size and under mild technical conditions, then

\[ \sum_{t=0}^{\infty} ||a(t+1) - a(t)|| < \infty \quad \sum_{t=0}^{\infty} ||a^p(t+1) - a^p(t)|| < \infty \]

In particular, the global and local sequences converge to the same critical point, with rate \(O(t^{-1})\):

\[ \mathcal{L} \left( \frac{1}{t} \sum_{k=1}^{t} a(k) \right) - \inf \mathcal{L} \leq O \left( t^{-1} \right) \]

Explanation: Finite length guarantees that the algorithm stops (the updates must eventually go to zero). Furthermore, the algorithm converges at rate \(O(t^{-1})\) to the optimal value; same as BSP model parallel.
Principles of ML system Design [Xing et al., to appear 2016]

3. How to Communicate:

*Managed Communication and Topologies*
Managed Communication [Wei et al., 2015]

- **SSP only**
  - Communicates only at iteration boundary
  - Ensures bounded staleness consistency

- **SSP + Managed Communication**
  - Continuous communication/synchronization
  - Update prioritization
  - Same consistency guarantees as SSP
MatrixFact:
Managed Communication Speedup

- Matrix Factorization, Netflix data, rank = 400
- 8 machines * 16 cores, 1GbE ethernet
LDA: Managed Communication Speedup

- Latent Dirichlet Allocation, NYTimes, # topics = 1000,
- 16 machines * 16 cores, 1GbE ethernet

Already enjoying SSP speedup

- 3x additional speed up from comms management
- 25% additional speedup from comms prioritization
Topology: Master-Slave

- Used with **centralized storage** paradigm
- Topology = **bipartite graph**: Servers (masters) to Workers (slaves)
- **Disadvantage**: need to code/manage clients and servers separately
- ** Advantage**: bipartite topology far smaller than full $N^2$ P2P connections
Topology: Peer-to-Peer (P2P)

- Used with decentralized storage paradigm
- Workers update local parameter view by broadcasting/receiving
- **Disadvantage**: expensive unless updates \( \Delta W \) are lightweight; expensive for large # of workers
- **Advantage**: only need worker code (no central server code); if \( \Delta W \) is low rank, comms reduction possible
Halton Sequence Topology  [Li et al., 2015]

- Used with **decentralized storage** paradigm
- Like P2P topology, but route messages through many workers
  - e.g. to send message from 1 to 6, use 1->2->3->6
- **Disadvantage:** incur higher SSP staleness due to routing, e.g. 1->2->3->6 = staleness 3
- **Advantage:** support bigger messages; support more machines than P2P topology
Random Partial Broadcasting and Diverse Mini-Batch Selection

- **Random Partial Broadcasting**
  - Each machine randomly selects $Q \ll P$ machines to send messages (instead of full broadcast)
  - Message cost reduced: from $O(P^2)$ to $O(PQ)$, scales linearly with machine count $P$!

- **Diverse Mini-batch Selection**
  - Choose training data samples that maximize diversity score
    \[ s = \frac{1}{K(K-1)} \sum_{k=1}^{K} \sum_{j \neq i}^{K} (-x_k^T x_j) \]
  - Pick few diverse samples instead of many random samples
  - Using few but diverse samples further reduces comms costs without hurting output quality!
4. What to Communicate:

*Exploiting Structure in ML Updates*
Matrix-Parameterized Models (MPMs)

\[ \min_W \frac{1}{N} \sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W) \]

Matrix parameter \( W \)

Loss function

Regularizer

Distance Metric Learning, Sparse Coding, Distance Metric Learning, Group Lasso, Neural Network, etc.
Big MPMs

Multiclass Logistic Regression on Wikipedia
- Feature dim. = 20K
- #classes=325K
- 6.5B

Distance Metric Learning on ImageNet
- Feature dim. = 172K
- Latent dim. = 50K
- 8.6B

Sparse Coding on ImageNet
- Feature dim. = 172K
- Dic. Size=50K
- 8.6B

Neural Network of Google Brain
- #neurons in layer 0 = 40K
- #neurons in layer 1 = 33K
- 1.3B

Billions of params = 10-100 GBs, costly network synchronization

What do we actually need to communicate?
Full Updates

- Let matrix parameters be $W$. **Need to send parallel worker updates** $\Delta W$ to other machines…
  - Primal stochastic gradient descent (SGD)
    
    $$
    \min_{W} \frac{1}{N} \sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W)
    $$
    
    $$
    \Delta W = \frac{\partial f(Wa_i, b_i)}{\partial W}
    $$
  
  - Stochastic dual coordinate ascent (SDCA)
    
    $$
    \min_{z} \frac{1}{N} \sum_{i=1}^{N} f_i^*(-z_i) + h^*(\frac{1}{N} ZA^T)
    $$
    
    $$
    \Delta W = (\Delta z_i)a_i
    $$
Sufficient Factor (SF) Updates
[Xie et al., 2015]

- **Full parameter matrix update** $\Delta W$ can be computed as outer product of two vectors $uv^T$ (called sufficient factors)
  - Primal stochastic gradient descent (SGD)
    \[
    \min_w \frac{1}{N} \sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W)
    \]
    \[
    \Delta W = uv^T, \quad u = \frac{\partial f(Wa_i, b_i)}{\partial(Wa_i)}, \quad v = a_i
    \]
  - Stochastic dual coordinate ascent (SDCA)
    \[
    \min_z \frac{1}{N} \sum_{i=1}^{N} f^*_i(-z_i) + h^*(\frac{1}{N}ZA^T)
    \]
    \[
    \Delta W = uv^T, \quad u = \Delta z_i, \quad v = a_i
    \]
- Send the lightweight SF updates $(u, v)$, instead of the expensive full-matrix $\Delta W$ updates!
P2P Topology + SF Updates = Sufficient Factor Broadcasting
Theorem 1. Let \( \{W^c_p\}, p = 1, \ldots, P, \) and \( \{W^c\} \) be the local sequences and the auxiliary sequence generated by SFB for problem \( (P) \) (with \( h = 0 \)), respectively. Under Assumption 1 and set the learning rate \( \eta_c^{-1} = \frac{L}{2} + 2sL + \sqrt{c} \), then we have

- \( \lim_{c \to \infty} \inf \mathbb{E}\|\nabla F(W^c)\| = 0 \), hence there exists a subsequence of \( \nabla F(W^c) \) that almost surely vanishes;
- \( \lim_{c \to \infty} \max_p \|W^c - W^c_p\| = 0 \), i.e. the maximal disagreement between all local sequences and the auxiliary sequence converges to 0 (almost surely);
- There exists a common subsequence of \( \{W^c_p\} \) and \( \{W^c\} \) that converges almost surely to a stationary point of \( F \), with the rate \( \min_{c \leq C} \mathbb{E}\| \sum_{p=1}^{P} \nabla F_p(W^c_p)\|_2^2 \leq O \left( \frac{(L + L_F)\sigma^2 s P \log C}{\sqrt{C}} \right) \).

**Explanation:** Parameter copies \( W_p \) on different workers \( p \) converge to the same optima, *i.e. all workers reach the same (correct) answer.*

- Does not need central parameter server or key-value store
- Works with SSP bridging model (staleness = \( s \))
Why is SFB faster?

- Faster than PS and Spark
- Near-linear scalability

Because SFB has faster iterations (less communication) … while keeping the same iteration quality as PS

SFB communication up to 100x smaller than PS and Spark
Summary

1. **How to Distribute?**
   - Structure-Aware Parallelization
   - Work Prioritization

2. **How to Bridge Computation and Communication?**
   - BSP Bridging Model
   - SSP Bridging Model for Data and Model Parallel

3. **How to Communicate?**
   - Managed comms – interleave comms/compute, prioritized comms
   - Parameter Storage: Centralized vs Decentralized
   - Communication Topologies: Master-Slave, P2P, Halton Sequence

4. **What to Communicate?**
   - Full Matrix updates
   - Sufficient Factor updates
   - Hybrid FM+SF updates (as in a DL model)