System Support for Diverse ML Styles

Eric Xing
Carnegie Mellon University

Acknowledgement:
Wei Dai, Qirong Ho, Jin Kyu Kim, Abhimanu Kumar, Seunghak Lee, Jinliang Wei, Pengtao Xie, Xun Zheng
James Cipar, Henggang Cui,
and, Phil Gibbons, Greg Ganger, Garth Gibson

http://www.istc-cc.cmu.edu/
Machine Learning:
A view from outside
An ML Program (from inside)

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

Model \hspace{2cm} Data \hspace{2cm} Parameter

Solved by an iterative convergent algorithm

\[
\text{for (t = 1 to T) } \{ \\
\quad \text{doThings}() \\
\quad \tilde{\theta}^{t+1} = g(\tilde{\theta}^{t}, \Delta f(\theta(\mathcal{D}))) \\
\quad \text{doOtherThings}() \\
\} 
\]

This computation needs to be fast!
Challenge #1
– Massive Data Scale

THE INTERNET OF THINGS
AN EXPLOSION OF CONNECTED POSSIBILITY

Source: The Connectivist

Source: Cisco Global Cloud Index

Familiar problem: data from 50B devices, data centers won’t fit into memory of single machine
Challenge #2
– Gigantic Model Size

Big Data needs Big Models to extract understanding
But ML models with >1 trillion params also won’t fit!
Growing Need for Big and Contemporary ML Programs

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
Why need new Big ML systems?

**MLer’s view**

- Focus on
  - Correctness
  - fewer iteration to converge,
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - uniform local progress

```plaintext
for (t = 1 to T) {
    doThings()
    parallelUpdate(x, θ)
    doOtherThings()
}
```

**Compute vs Network**
LDA 32 machines (256 cores)

- Network waiting time
- Compute time

Parallelize over worker threads
Share global model parameters via RAM
Why need new Big ML systems?

**Systems View:**
- Focus on
  - high iteration throughput (more iter per sec)
  - strong fault-tolerant atomic operations,
- but assume ML algo is a **black box**
  - ML algos “still work” under different execution models
  - “easy to rewrite” in chosen abstraction

Agonistic of ML properties and objectives in system design

- Non-uniform convergence
- Dynamic structures
- Error tolerance

Synchronization model

Programming model
Why need new Big ML systems?

**MLer’s view**
- Focus on
  - Correctness
  - fewer iteration to converge,
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - uniform local progress

```java
for (t = 1 to T) {
  doThings()
  parallelUpdate(x, θ)
  doOtherThings()
}
```

**Oversimplify systems issues**
- need machines to perform consistently
- need lots of synchronization
- or even try not to communicate at all

**Oversimplify ML issues and/or ignore ML opportunities**
- **ML algos “just work” without proof**
- Conversion of ML algos across different program models (graph programs, RDD) is easy

**Systems View:**
- Focus on
  - high iteration throughput (more iter per sec)
  - strong fault-tolerant atomic operations,
- but assume ML algo is a **black box**
  - ML algos “still work” under different execution models
  - “easy to rewrite” in chosen abstraction
Solution:

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

- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- GPUs
- Desktops/Laptops
- NUMA machines
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines

Machine Learning Models/Algorithms
Solution: An Alg/Sys INTERFACE for Big ML

- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Others

Machine Learning Models/Algorithms

- Network switches
- Network attached storage
- Server machines
- GPUs
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines
- Infiniband
- Flash storage
- Desktops/Laptops
- NUMA machines

Solution: An Alg/Sys INTERFACE for Big ML
The Big ML “Stack” - More than just software

**Theory:** Degree of parallelism, convergence analysis, sub-sample complexity …

**System:** Distributed architecture: DFS, parameter server, task scheduler…

**Model:** Generic building blocks: loss functions, structures, constraints, priors …

**Algorithm:** Parallelizable and stochastic MCMC, VI, Opt, Spectrum …

**Representation:** Compact and informative features

**Programming model & Interface:** High: Matlab/R Medium: C/Java Low: MPI

**Hardware:** GPU, flash storage, cloud …
ML algorithms are Iterative-Convergent

Markov Chain Monte Carlo

Optimization
A General Picture of ML
Iterative-Convergent Algorithms

\[ \Delta = \Delta(A^{(t-1)}, D) \]
\[ A^{(t)} = F(A^{(t-1)}, \Delta) \]

Data

Model Parameters at iteration (t-1)
Challenge

- **Optimization programs:**

\[
\Delta \leftarrow \sum_{i=1}^{N} \left[ \frac{d}{d\theta_1}, \ldots, \frac{d}{d\theta_M} \right] f(x_i, y_i; \tilde{\theta})
\]

A huge volume of data (e.g.) \( N = 1B \)

A huge number of parameters (e.g.) \( J = 1B \)
Challenge

- Probabilistic programs

\[ z_{di} \sim p(z_{di} = k | \text{rest}) \propto \left( \frac{n_{kd}^{-di}}{n_{k}^{-di} + \alpha_k} \right) \cdot \frac{n_{kw}^{-di} + \beta_w}{n_{k}^{-di} + \beta V} \]
A Dichotomy of Data and Model in ML Programs

\[ \hat{\theta}^{t+1} = \hat{\theta}^{t} + \Delta f \hat{\theta}(D) \]
A Dichotomy of Data and Model in ML Programs

\[
\theta^{t+1} = \theta^t + \Delta_f \theta(D)
\]

\[
D \equiv \{D_1, D_2, \ldots, D_n\}
\]

Data Parallel

\[
D_i \perp D_j \quad | \quad \theta, \forall i \neq j
\]

Model Parallel

\[
\theta_i \not\equiv \theta_j \quad | \quad D, \exists (i, j)
\]
Good Parallelization Strategy is important

for (t = 1 to T) {
  doThings()
  parallelUpdate(x, θ)
  doOtherThings()
}
Usually, we worry …

A sequential program

\[ \beta_1 \rightarrow \beta_2 \]

A parallel program

\[ \beta_1 \rightarrow \beta_2 \]

\[ \beta_1 \rightarrow \beta_2 \]

- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - zero-cost fault recovery
  - uniform local progress
  - …
ML Computation vs. Classical Computing Programs

ML Program: optimization-centric and iterative convergent

Traditional Program: operation-centric and deterministic
Traditional Data Processing needs operational correctness

Example: Merge sort

Sorting error: 2 after 5

Error persists and not corrected
ML Algorithms can Self-heal
Intrinsic Properties of ML Programs

- 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 platforms fit the above?
Efficient and correct ML programming is nontrivial

- ML is very effective under non-blocking, bounded-asynchronous communication, but how to ensure correct dependencies?
- ML programs are “stateful” --- model state $\theta$ updated every iteration; (fresh) auxiliary local variables, e.g. summary statistics, needed at each parallel worker
- Big ML programs can require explicit partition and scheduling

- An ideal ML programming interface should make it easy to write correct data-parallel, model-parallel ML programs
  - Abstract away inter-worker communication/synchronization
  - Abstract scheduling away from update equations
  - Abstract away worker management
  - Ideally, programmer does just 3 things: declare model, write updates, write schedule
What does an ML programmer need?

First-timer’s ideal view of ML

`global model = (a,b,c,...)`
`global data = load(file)`

`Update(var a):`
  `a = doSomething(data,model)`

`Main:`
  `do Update() on all var in model until converged`

High-performance implementation

Many considerations

- What data batch size?
- How to partition model?
- When to sync up model?
- How to tune step size?
- What order to Update()?

1000s of lines of extra code

Goal: system that can be programmed like “ideal view”, yet yields state-of-the-art performance
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!

Image source: dzone.com

D ≔ \{D_1, D_2, \ldots, D_n\}
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
GraphLab: Model-Parallel via Graphs

- GraphLab **Graph consistency models**
  - Guide search for “ideal” model-parallel execution order
  - ML algo correct if input graph **has all dependencies**

- GraphLab supports asynchronous (no-waiting) execution
  - Correctness enforced by graph consistency model
  - Result: GraphLab graph-parallel ML much faster than Hadoop

Source: Low et al. (2010)
A New Framework for Large Scale Parallel Machine Learning
(Petuum.org)
Petuum Overview

- Key modules
  - KV-store for data-parallel ML algos
  - Scheduler for model-parallel ML algos

- “Think like an ML algo”
  - ML algo = (1) update equations + (2) run those eqns in some order

- HDFS (distributed storage)
- YARN (resource manager, fault tolerance)
- Bounded-Async KV-store (Bösen)
- Dynamic Scheduler (Strads)
- Data-Parallel API
- Model-Parallel API
- ML application library
High Efficiency

• Petuum makes ML apps more efficient

• Versus Spark MLlib v1.3, Petuum is faster by
  • 8x on Logistic Regression for CTR and Event Prediction
  • 100x on Topic Modeling for User Profiling
  • 20x on Lasso Regression for Genetic Assay Analysis
  • Scale: 10-100 machines, GBs-TBs of data

• Versus specialized implementations

• Distributed Convolutional Neural Network built on Caffe: Train Alexnet (60m parameters) in under 24 hours, on 8 GPU machines
Lots of Advanced Apps

- **DNN**: Petuum Brain for mining images, videos, speech, text, biology
- **(Med)LDA**: Web-scale analysis of docs, blogs, tweets
- **Regression**: Linear and Logistic for intent prediction, stock/future hedging
- **(N)MF**: Collaborative Filtering for recommending movies, products
- **MMTM**: Societal/web-scale network analysis, community detection
- **SVM**: General-purpose Classification
- **Ising**: Model power and sensor grids
- **SIOR**: Genome-wide association, stock/future hedging
- **ADMM**: Constrained optimization for operations research, logistics management
- **Kalman**: Kalman Filters for aviation control, dynamic system prediction
- **SC**: Sparse Coding for web-scale, million-class classification
- **Metric**: Distance Metric Learning to boost large-scale classification
principles, design, and theory

- Key insight: ML algos have special properties
  - Error-tolerance, dependency structures, uneven convergence
  - How to harness for faster data/model-parallelism?
Data-Parallel Stochastic Gradient Descent

- Consider:
  \[
  \min_x \mathbb{E}\{f(x, d)\}
  \]

- SPG:
  \[
  x^{(t+1)} \leftarrow x^{(t)} - \gamma \nabla_x f(x^{(t)}, d_i)
  \]

- Parallel SGD [Zinkevich et al., 2010]: Partition data to different workers; all workers update full parameter vector

- PSGD runs SGD on local copy of params in each machine
There Is No Ideal Distributed System!

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

[Diagram showing unequal performance with low bandwidth and high delay]

[Bar chart comparing compute time and network waiting time for LDA 32 machines (256 cores)]
How to speed up Data-Parallelism?

- Existing ways are either safe but slow, or fast but risky

- 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?
High-Performance Consistency Models for Fast Data-Parallelism


Stale Synchronous Parallel (SSP)
- Allow threads to run at their own pace, without synchronization
- Fastest/slowest threads not allowed to drift >S iterations apart
- Threads cache local (stale) versions of the parameters, to reduce network syncing

Consequence:
- Asynchronous-like speed, BSP-like ML correctness guarantees
- Guaranteed age bound (staleness) on reads
- Contrast: no-age-guarantee Eventual Consistency seen in Cassandra, Memcached
Bösen:

a bounded async key-value store

- Put global parameters in BA-KVS. Examples:
  - **Topic Modeling (MCMC)**
    - Topic-word table
  - **Matrix Factorization (SGD)**
    - Factor matrices $L$, $R$
  - **Lasso Regression (CD)**
    - Coefficients $\beta$
  - **A DSM UI:**
    ```python
    UpdateVar(i) {
        old = PS.read(y,i)
        delta = f(old)
        PS.inc(y,i,delta)
    }
    ```
  - Supports **many classes of algorithms**
  - Above are just a few examples

Convergence Theorem


- **Goal**: minimize convex \( f(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x) \) (Example: Stochastic Gradient)
  - \( L \)-Lipschitz, problem diameter bounded by \( F^2 \)
  - Staleness \( s \), using \( P \) threads across all machines
  - Use step size \( \eta_t = \frac{\sigma}{\sqrt{t}} \) with \( \sigma = \frac{F}{L\sqrt{2(s+1)P}} \)

- **SSP converges according to**
  - Where \( T \) is the number of iterations

\[
R[X] := \frac{1}{T} \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*) \leq 4FL \sqrt{\frac{2(s+1)P}{T}}
\]

- Note the RHS interrelation between \((L, F)\) and \((s, P)\)
  - An interaction between *theory* and *systems* parameters
  - Stronger guarantees on means and variances can also be proven
Enjoys Async Speed, But BSP Guarantee across algorithms

- Massive Data Parallelism
- Effective across different algorithms
BAP Data Parallel: (E)SSP Probability Bound [Dai et al., 2015]

Let real staleness observed by system be $\gamma_t$
Let its mean, variance be $\mu_\gamma = \mathbb{E}[\gamma_t], \quad \sigma_\gamma = \text{var}(\gamma_t)$

**Theorem:** Given L-Lipschitz objective $f_t$ and stepsize $h_t$,

$$P \left[ \frac{R[X]}{T} - \frac{1}{\sqrt{T}} \left( \eta L^2 + \frac{F^2}{\eta} + 2\eta L^2 \mu_\gamma \right) \geq \tau \right] \leq \text{exp} \left\{ \frac{-T \tau^2}{2\bar{\eta}T \sigma_\gamma + \frac{2}{3} \eta L^2 (2s + 1) P \tau} \right\}$$

- Gap between current estimate and optimum
- Penalty due to high avg. staleness $\mu_{\text{stale}}$
- Penalty due to high staleness var. $\sigma_{\text{stale}}$

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

**Explanation:** the (E)SSP distance between true optima and current estimate decreases exponentially with more iterations. Lower staleness mean, variance $\mu_\gamma, \sigma_\gamma$ improve the convergence rate.

**Take-away:** controlling staleness mean $\mu_\gamma$, variance $\sigma_\gamma$ (on top of max staleness s) is needed for faster ML convergence, which ESSP does.
Steadier convergence

**Theorem:** the variance in the (E)SSP estimate is

\[
\text{Var}_{t+1} = \text{Var}_t - 2\eta_t \text{cov}(x_t, \mathbb{E}^{\Delta_t}[g_t]) + \mathcal{O}(\eta_t \xi_t) + \mathcal{O}(\eta_t^2 \rho_t^2) + \mathcal{O}^*_{\gamma_t}
\]

where

\[
\text{cov}(a, b) := \mathbb{E}[a^T b] - \mathbb{E}[a^T] \mathbb{E}[b]
\]

and \(\mathcal{O}^*_{\gamma_t}\) represents 5th order or higher terms in \(\gamma_t\).

**Explanation:** The variance in the (E)SSP parameter estimate monotonically decreases when close to an optimum.

**Lower (E)SSP staleness** \(\gamma_t\) \(\Rightarrow\) **Lower variance in parameter** \(\Rightarrow\) **Less oscillation in parameter** \(\Rightarrow\) **More confidence in estimate quality and stopping criterion.**
Challenges in Model Parallelism

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

A huge number of parameters (e.g.) \( J = 100M \)

- Within group – synchronous (i.e., sequential) update
- Inter group – asynchronous update
Model Dependencies in Lasso

• Concurrent updates of $\beta$ may induce errors

Sequential updates

Concurrent updates

Induces parallelization error

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

Need to check $x_1^T x_2$ before updating parameters
How to speed up Data-Parallelism?

- Existing ways are either safe but slow, or fast but risky
- Challenge 1: need approximate but fast model partition
  - Full representation of data/model, and explicitly compute all dependencies via graph cut is not feasible
- Challenge 2: need dynamic load balancing
  - Capture and explore transient model dependencies
  - Explore uneven parameter convergence

Is full consistency really necessary for ML?
Structure-Aware Parallelization (SAP)

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

- Simple programming:
  - Schedule()
  - Push()
  - Pull()
Structure-aware Dynamic Scheduler (STRADS)  


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

- Block scheduling

[Kumar, Beutel, Ho and Xing, Fugue: Slow-worker agnostic distributed learning, AISTATS 2014]
Comparison: p-scheduling vs. u-scheduling

- Priority-based scheduling converged faster than the baseline with random scheduling.

Priority-based scheduling + dep. checker

\[ p(j) \sim (\delta \beta_j^{(t-1)})^2 + r \]

Shotgun scheduling [Bradley et al. 2011]

Initialization phase to make a sampling distribution

100M features 9 machines

Seconds

Objective

Out of memory

Single Lasso
Dynamic Scheduling Leads to Faster Convergence

For \( P \) parallel workers, \( M \)-dimensional data

Let \( \rho \) be the spectral radius of \( X \)

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

**Theorem:** the difference between the STRARD estimate and the true optima is

\[
\text{Gap between current parameter estimate and optimum}
\]

SAP explicitly minimizes \( \rho \), ensuring as close to \( 1/P \) convergence as possible

**Explanation:** Dynamic scheduling ensures the gap between the objective at the \( t \)-th iteration and the optimal objective is bounded by \( \mathcal{O} \left( \frac{1}{P \cdot t} \right) \), which decreases as \( t \to \infty \). Therefore dynamic scheduling ensures convergence.
Dynamic scheduling is close to ideal

Let $S_{\text{ideal}}(\cdot)$ be an ideal model-parallel schedule
Let $\beta_{\text{ideal}}^{(t)}$ be the parameter trajectory by ideal schedule
Let $\beta_{\text{dyn}}^{(t)}$ be the parameter trajectory by dynamic schedule
Let $C \propto P L^2$

Theorem: After $t$ iterations, we have

$$E[|\beta_{\text{ideal}}^{(t)} - \beta_{\text{dyn}}^{(t)}|] \leq C \frac{2M}{(t + 1)^2} X^\top X$$

Explanation: Under dynamic scheduling, algorithmic progress is nearly as good as ideal model-parallelism. Intuitively, it is because both ideal and dynamic model-parallelism seek to minimize the parameter dependencies crossing between workers.
Faster, Better Convergence across algorithms

- STRADS+SAP achieves better speed and objective
Summary: ML Computing is not Traditional Computing

1. Self-healing
2. Dynamic-rewiring
3. Uneven pace

A new architecture adapts to the new needs for ML computing is needed to turbocharge ML performance.
Acknowledgements

Garth Gibson
Greg Ganger
Phillip Gibbons
James Cipar