Petuum: A New Platform for Distributed Machine Learning on Big Data

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

Acknowledgement:
Wei Dai, Qirong Ho, Jin Kyu Kim, Abhimanyu Kumar, Seunghak Lee, Jinliang Wei, Pengtao Xie, Xun Zheng
Zhiting Hu, Hao Zhang, James Cipar, Henggang Cui,
and, Phil Gibbons, Greg Ganger, Garth Gibson
Trees Falling in the Forest

"If a tree falls in a forest and no one is around to hear it, does it make a sound?" --- George Berkeley

Data ≠ Knowledge

- Nobody knows what’s in data unless it has been processed and analyzed
  - Need a scalable way to automatically search, digest, index, and understand contents
Massive Data

1B+ USERS
30+ PETABYTES

32 million pages

100+ hours video uploaded every minute

645 million users
500 million tweets / day
Growing Model Complexity

Google Brain
Deep Learning for images:
10 Billion model parameters

Simplest whole-genome analysis:
100 million model parameters

News article analysis:
1 Trillion model parameters

Video recommendation:
10 Billion model parameters
Solution:
Solution: An Alg/Sys INTERFACE for Big ML

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

Hardware and infrastructure

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

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

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
ML Computing is not Traditional Computing

ML Computing:
Self-healing, iterative, convergent

Traditional Computing:
Error-sensitive, one-pass

Example: Merge Sort

Traditional platforms over-engineered for ML:
too expensive and slow

Correct answer in spite of small error!
A New Framework for Large Scale Parallel Machine Learning
(Petuum.org)
Petuum Architecture & Ecosystem

ML application library

Data-Parallel API

Bösen Data-Parallel Engine

Parameter Tuning

Model-Parallel API

Strads Model-Parallel Engine

Data Transform

YARN (resource manager, fault tolerance)

HDFS (distributed storage)

Stand-alone cluster operation
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
Data-Parallel Engine

- Bösen: a bounded-asynchronous distributed key-value store
  - Data-parallel programming via distributed shared memory (DSM) abstraction
  - Managed communication for better parallel efficiency & guaranteed convergence

```
UpdateVar(i) {
  old = y[i]
  delta = f(old)
  y[i] += delta
}
```

```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```
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
How to speed up Data-Parallelism?

- Existing ways are either safe/slow (BSP), or fast/risky (Async)

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

- 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

Convergence Theorem

W. Dai, A. Kumar, J. Wei. Q. Ho, G. Gibson and E. P. Xing, High-Performance Distributed ML at Scale through Parameter Server Consistency Models. AAAI 2015.

- **Goal:** minimize convex \( f(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x) \)
  - (Ex: Stochastic Gradient)
  - Max allowed staleness \( s \), over \( P \) parallel workers
  - For accurate analysis, we must consider real staleness observed by system
    - average \( \mu_{\text{stale}} \), variance \( \sigma_{\text{stale}} \)

- **SSP converges according to**
  - Where \( T \) is # of iterations, \( \tau \) is desired answer quality

\[
\Pr \left[ \frac{R[X]}{T} \geq \tau + \frac{O(\mu_{\text{stale}})}{\sqrt{T}} \right] \leq \exp \left\{ -\frac{T \tau^2}{o(T)\sigma_{\text{stale}} + O(s\tau)} \right\}
\]

- Take-away: faster and more efficient ML requires SSP, which limits the staleness max \( s \), and minimizes staleness mean \( \mu_{\text{stale}} \), variance \( \sigma_{\text{stale}} \)
Bösen advantages:

- **Enjoys async speed**, but **BSP guarantee** across algorithms

- **Light network traffic**: Low-rank (sufficient factor) pre-updates
- **Auto-tuning**: for both parameter & step-size
- **Unified data interface**: support many **feature transforms**
Model-Parallel Engine

- Strads: a structure-aware load-balancer and task prioritizer
  - Model-parallel programming via a scheduler interface
  - Explore structural dependencies and non-uniform convergence within ML models for best execution order

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

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

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.
    ...
}
```
Challenges in Model Parallelism

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

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

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

- Within group – synchronous (i.e., sequential) update
- Inter group – asynchronous update
How to Model-Parallel?

- Again, existing ways are either safe but slow, or fast but risky
- Need to avoid processing whole-data just for optimal distribution
  - i.e., build expensive data representation on the whole data
  - Compute all variable dependencies
- Dynamic load balance

Is correct global dependency really necessary for ML?
Structure-Aware Parallelization (SAP)

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

- **Block scheduling**

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

**Simple programming:**
- Schedule()
- Push()
- Pull()
Convergence Theorem


- **Goal:** solve sparse regression problem
  \[
  \min_{\beta} \| y - X\beta \|_2^2 + \lambda \sum_j |\beta_j| 
  \]
  - Via coordinate descent over \(X^{(1)}, X^{(2)}, ..., X^{(T)}\)
    - where \(X^{(t)}\) is the data columns (features) chosen for updating at iteration \(t\)
  - \(P\) parallel workers, \(M\)-dimensional data
  - \(\rho = \max_t (X^{(t)})^T X^{(t)}\), i.e. the maximum “data difficulty” (technically, spectral radius) across all data subsets \(X^{(1)}, X^{(2)}, ..., X^{(t)}\)

- **SAP converges according to**
  - Where \(t\) is # of iterations
  - SAP scheduling minimizes \(\rho\), ensuring close to \(1/P\) convergence, i.e. near-perfect scaling with \(P\) workers

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

- Take-away: faster and more efficient ML requires SAP, which minimizes the difficulty \(\rho\) of the problem by searching for uncorrelated feature subsets \(X^{(1)}, X^{(2)}, ..., X^{(t)}\)
Strads advantages:

- Faster, **near-ideal** convergence speed across algorithms

- **Structure-aware parallelization**: auto-find best update order
- **Pipelining**: overlap schedule() with push/pull() for speed
- Parameter & step-size **auto-tuning**
- Library of **feature transforms**
Efficiency

- Petuum automatically 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
Efficiency Demo - MatrixFact
Efficiency Demo - CNN
High-Speed Model Building and Prediction

**High-volume model building for real-time data streams**

- CNN Deep Learning: *200 images / second / GPU machine*
- Topic Model: *5 million words / second / machine*
- Multiclass LR: *1 million events / second / machine*
- Random Forest: *500k events / second / machine*
- Metric Learning: *24 million ops / second / machine*

**High-speed analytics predictions**

- CNN Deep Learning: *350 images / second / machine*
- Matrix Factorization: *1.2m recommendations / second / machine*
- Random Forest: *0.5m classification / second / machine*
Petuum Verticals

Video Summarization with deep learning & parallel GPU system
1000s of cameras, millions of frames: find traffic accidents, suspicious activity

User Profiling with supervised topic models
Turn blogs, tweets, videos, pics, shopping => customer profile

Mobile Call Data Record (CDR) Analysis with graph miners
Predict device adoption, churners, influence spread in a telco setting

Click-through-rate and Event Prediction with sparse regression
~1B events/hour for mobile apps, e.g. streaming video, taxi services, mobile advertising

Recommender Systems with collaborative filtering
~100M users, ~10M products, even with ever-changing ratings, growing userbase/product catalogue
Acknowledgements

Garth Gibson
Greg Ganger
Phillip Gibbons
James Cipar