High-Performance Distributed Machine Learning in Heterogeneous Compute Environments

Celestine Dünner

Martin Jaggi (EPFL)
Thomas Parnell, Kubilay Atasu, Manolis Sifalakis, Dimitrios Sarigiannis, Haris Pozidis (IBM Research)
Motivation of Our Work

- fast training
- interpretable models
- training on large-scale datasets
Motivation of Our Work

Distributed Training of Large-Scale Linear Models

Choose an Algorithm
Choose an Implementation
Choose an Infrastructure

? How does the infrastructure and the implementation impact the performance of the algorithm?
Motivation of Our Work

Distributed Training of Large-Scale Linear Models

Choose an Algorithm

Choose an Implementation

Choose an Infrastructure

How can algorithms be optimized and implemented to achieve optimal performance on a given system?
Algorithmic Challenge of Distributed Learning

\[ \min_w \ f( A^T w) + g(w) \]
Algorithmic Challenge of Distributed Learning

\[
\min_w \left( f(A^T w) + g(w) \right)
\]

- The more frequently you exchange information the faster your model converges
- Communication over the network can be very expensive

Trade-off

aggregate local models

Tunable hyper-parameter $H$

$H^*$ depends on:
- system
- Implementation/framework

\[
\min_w f(A^T w) + g(w)
\]
Implementation: Frameworks for Distributed Computing

Open Source Cloud Computing Framework

- Easy-to-use
- Powerful APIs: Python, Scala, Java, R
- Poorly understood overheads

* http://spark.apache.org/

High-Performance Computing Framework

- Requires advanced system knowledge
- C++
- Good performance

Designed for different purposes → different characteristics

Apache Spark, Spark and the Spark logo are trademarks of the Apache Software foundation (ASF). Other product or service names may be trademarks or service of IBM or other companies.
Different implementations of CoCoA

- (A) Spark Reference Implementation*
- (B) pySpark Implementation
- (C) MPI Implementation

Offload local solver to C++

- (A*) Spark+C
- (B*) pySpark+C

(A*),(B*),(C) execute identical C++ code

*https://github.com/gingsmith/cocoa

100 iterations of CoCoA for $H$ fixed
Understanding characteristics of the framework and correctly adapting the algorithm can decide upon orders of magnitude in performance!
strive to design flexible algorithms that can be adapted to system characteristics

be aware that machine learning algorithms need to be tuned to achieve good performance

Which local solver should we use?
Stochastic Primal-Dual Coordinate Descent Methods

Algorithm 1 Primal-Dual Coordinate Descent

1: Initialize $w^{(0)} = 0$
2: Initialize $v^{(0)} = 0$ ($v := A^T w$)
3: for $t = 0, 1, 2, \ldots$ do
4: \hspace{1em} randomly select a coordinate $j \in [n]$
5: \hspace{1em} find $\Delta w_j$ to minimize $f(w^{(t)} + \Delta w_j x_j) + g_j(w_j + \Delta w_j)$
6: \hspace{1em} $w^{(t+1)} = w^{(t)} + \Delta w_j e_j$
7: \hspace{1em} $v^{(t+1)} = v^{(t)} + \Delta w_j x_j$
8: end for

- $f$: smooth
- $f^*$: strongly-convex

$\min_w f(A^T w) + \sum_i g_i(w_i)$

$\min_w \frac{1}{2n} ||A^T w - y||_2^2 + \frac{\lambda}{2} ||w||_2^2$

$\min_w \frac{1}{2n} ||A^T w - y||_2^2 + \lambda ||w||_1$

Problem: cannot leverage full power of modern CPUs or GPUs

Sequential

Asynchronous implementations

$\min_{\alpha} f^*(\alpha) + \sum_i g_i^*(-A_{i}^T w)$

$L_2$-regularized SVM, Ridge Regression, $L_2$-regularized Logistic Regression....

Ridge Regression, Lasso, Logistic Regression....

Good convergence properties
Asynchronous Stochastic Algorithms

$$\min_w f(A^T w) + \sum_i g_i(w_i)$$

**Parallelized over cores:**
Every core updates a dedicated subset of coordinates

---

**Algorithm 1 Primal-Dual Coordinate Descent**

1. Initialize $w^{(0)} = 0$
2. Initialize $v^{(0)} = 0$ ($v := A^T w$)
3. for $t = 0, 1, 2, \ldots$ do
4. randomly select a coordinate $j \in [n]$
5. find $\Delta w_j$ to minimize $f(v^{(t)} + \Delta w_j x_j^T) + g_j(w_j + \Delta w_j)$
6. $w^{(t+1)} = w^{(t)} + \Delta w_j e_j$
7. $v^{(t+1)} = v^{(t)} + \Delta w_j x_j$
8. end for

---

Write collision on shared vector

- Recompute shared vector Liu et al., "AsySCD" (JMLR’15)
- Memory-locking K. Tran et al., “Scaling up SDCA, (SIGKDD’15)
- Live with undefined behavior C.-J. Hsieh et al. “PASSCoDe:“ (ICML’15)
Asynchronous Stochastic Algorithms

**Algorithm 1 Primal-Dual Coordinate Descent**

1. Initialize $w^{(0)} = 0$
2. Initialize $v^{(0)} = 0$ ($v := A^T w$)
3. for $t = 0, 1, 2, \ldots$ do
   4. randomly select a coordinate $j \in [n]$
   5. find $\Delta w_j$ to minimize $f(v^{(t)} + \Delta w_j x_j^T) + g_j(w_j + \Delta w_j)$
   6. $w^{(t+1)} = w^{(t)} + \Delta w_j e_j$
   7. $v^{(t+1)} = v^{(t)} + \Delta w_j x_j$
8. end for

Write collision on shared vector

- Recompute shared vector Liu et al., "AsySCD" (JMLR’15)
- Memory-locking K. Tran et al., “Scaling up SDCA, (SIGKDD’15)
- Live with undefined behavior C.-J. Hsieh et al. “PASSCoDe:“ (ICML’15)

Ridge Regression on webspam
2-level Parallelism of GPUs

1st level of parallelism:
- A GPU consists of streaming multiprocessors
- Thread blocks get assigned to multi-processors and are executed asynchronously

2nd level of parallelism
- Each thread block consists of up to 1024 threads
- Threads are grouped into warps (32 threads) which are executed as SMDI operations
GPU Acceleration

A Twice Parallel Asynchronous Stochastic Coordinate Descent (TPA-SCD) Algorithm

1. thread-blocks are executed in parallel, asynchronously updating one coordinate each.

2. Update computation within a thread block is interleaved to ensure memory locality within warp and local memory is used to accumulate partial sums

3. Atomic add functionality of modern GPUs is used to update shared vector

---

T. Parnell, C. Dünner, K. Atasu, M. Sifalakis, H. Pozidis, „Large-Scale Stochastic Learning Using GPUs“, IEEE International Parallel and Distributed Processing Symposium (IPDPS), Lake Buena Vista, FL, 2017

---

webspam dataset
GPU: GeForce GTX 1080Ti
CPU: 8-core Intel Xeon E5
Which local solver should we use?

It depends on the available hardware.
Heterogeneous System
Heterogeneous System
Dual Heterogeneous Learning [NIPS’17]

A scheme to efficiently use Limited-Memory Accelerators for Linear Learning

Idea: The GPU should work on the part of the data it can learn most from.

Contribution of individual data columns to the duality gap is indicative for their potential to improving the model

Algorithm 1 Primal-Dual Coordinate Descent

1: Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2: Initialize $\mathbf{v}^{(0)} = \mathbf{0}$ \hspace{1cm} ($\mathbf{v} := A^T \mathbf{w}$)
3: for $t = 0, 1, 2, \ldots$ do
4: \hspace{1cm} randomly select a coordinate $j \in [n]$
5: \hspace{1cm} find $\Delta w_j$ to minimize $f(\mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^T) + g_j(w_j + \Delta w_j)$
6: \hspace{1cm} $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta w_j \mathbf{e}_j$
7: \hspace{1cm} $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^T$
8: end for
**Dual Heterogeneous Learning [NIPS’17]**

A scheme to efficiently use Limited-Memory Accelerators for Linear Learning

**Idea:** The GPU should work on the part of the data it can learn most from.

- Contribution of individual data columns to the duality gap is indicative for their potential to improving the model

Duality Gap Computation is expensive!

\[ \text{Gap}(\mathbf{w}) = \sum_j w_j \langle x_j, \mathbf{v} \rangle + g_j(w_j) + g_j^*(x_j^\top \mathbf{v}) \]

- we introduce a *gap-memory*

- Parallelization of workload between CPU and GPU
  - **GPU** runs algorithm on subset of the data
  - **CPU** computes importance values
DuHL Algorithm

<table>
<thead>
<tr>
<th>Algorithm 3 DuHL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Initialize $w^{(0)} = 0$, $z = 0$</td>
</tr>
<tr>
<td>2: for $t = 0, 1, 2, \ldots$</td>
</tr>
<tr>
<td>3: determine $\mathcal{P} := \arg \max_{\mathcal{P} \subseteq [n]:</td>
</tr>
<tr>
<td>4: refresh GPU memory to contain $A[\mathcal{P}]$.</td>
</tr>
<tr>
<td>5: on GPU do:</td>
</tr>
<tr>
<td>6: find $\Delta w[\mathcal{P}] \leftarrow$ approx. solution to local subproblem</td>
</tr>
<tr>
<td>7: in parallel on CPU do:</td>
</tr>
<tr>
<td>8: while GPU not finished</td>
</tr>
<tr>
<td>9: sample $j \in [n]$</td>
</tr>
<tr>
<td>10: update $z_j = \text{Gap}_j(w_j^{(t)})$</td>
</tr>
<tr>
<td>11: $w^{(t+1)} = w^{(t)} + \Delta w[\mathcal{P}]$</td>
</tr>
</tbody>
</table>

C. Dünner, T. Parnell, M. Jaggi, „Efficient Use of Limited-Memory Accelerators for Linear Learning on Heterogeneous Systems“, NIPS, Long Beach, CA, 2017
DuHL: Performance Results

**Fig 1:** Superior convergence properties of DuHL over existing schemes

**Fig 2:** I/O efficiency of DuHL

- Reduced I/O cost and faster convergence accumulate to **10x speedup**

ImageNet dataset 30GB

**GPU:** NVIDIA Quadra M4000 (8GB)

**CPU:** 8-core Intel Xeon X86 (64GB)
**DuHL: Performance Results**

Reduced I/O cost and faster convergence accumulate to **10x speedup**

---

**ImageNet dataset 30GB**

**GPU:** NVIDIA Quadra M4000 (8GB)

**CPU:** 8-core Intel Xeon X86 (64GB)
Combining it all

A library for ultra-fast machine learning

**Goal:** remove training as a bottleneck

- Enable seamless retraining of models
- Enable agile development
- Enable training on large-scale datasets
- Enable high quality insights

✅ Exploit Primal-Dual Structure of ML problems to minimize communication

✅ Offer GPU acceleration

✅ Implement DuHL for efficient utilization of limited-memory accelerators

✅ Improved memory management of Spark
Combining it all

A library for ultra-fast machine learning

**Goal:** remove training as a bottleneck

- Enable seamless retraining of models
- Enable agile development
- Enable training on large-scale datasets
- Enable high quality insights

✔ Exploit Primal-Dual Structure of ML problems to minimize communication
✔ Offer GPU acceleration
✔ Implement DuHL for efficient utilization of limited-memory accelerators
✔ Improved memory management of Spark

Tera-Scale Advertising Application

*Predict whether a user will click on a given advert based on an anonymized set of features.*

*training:* 1 billion example  
*testing:* 100 million unseen examples

<table>
<thead>
<tr>
<th>Linear Regression (8 executors)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Library</strong></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>spark.ml</td>
</tr>
<tr>
<td>our library</td>
</tr>
</tbody>
</table>

**Power8 (Minsky) infrastructure**  
8 x P100 GPU
Summary

- Distributed Algorithm that offer tunable hyper-parameters are of particular practical interest.

- A user may expect orders of magnitude improvement by optimizing such parameters.

- GPUs can accelerate machine learning workloads by an order of magnitude if algorithms are carefully designed.

- DuHL enables GPU acceleration even if the data exceeds the capacity of the GPU memory.

- Combining all this knowledge we can remove training time as a bottleneck for ML applications.
Questions?