Scalable Asynchronous Gradient Descent Optimization for Out-of-Core Models

Chengjie Qin\textsuperscript{1}, Martin Torres\textsuperscript{2}, and Florin Rusu\textsuperscript{2}

\textsuperscript{1}GraphSQL, Inc.
\textsuperscript{2}University of California Merced

August 31, 2017
Machine Learning (ML) Is Booming

- General frameworks with ML libraries: Hadoop’s Mahout, Spark’s MLLib, GraphLab
- Specialized ML systems: Vowpal Wabbit, SystemML, SimSQL, TensorFlow
- **In-Database ML: MADlib, Bismarck, GLADE**
ML for Generalized Linear Models

• Model is $d$-dimensional vector $\vec{w}$, $d \geq 1$

• Training data $\bar{X}$ of $N$ $d$-dimensional feature vectors $\bar{x}_i$ and their corresponding label $y_i$, $1 \leq i \leq N$

• Objective function (or loss): $\Lambda(\vec{w}) = \min_{\vec{w} \in \mathbb{R}^d} \sum_{i=1}^{N} f(\vec{w}, \bar{x}_i; y_i)$

• Find model $\vec{w}$ that minimizes objective function based on training data

Logistic Regression (LR)

• $\Lambda_{LR}(\vec{w}) = \sum_{i=1}^{N} \log \left(1 + e^{-y_i \vec{w} \cdot \bar{x}_i}\right)$

Low-Rank Matrix Factorization (LMF)

• $\Lambda_{LMF}(L, R) = \frac{1}{2} \sum_{(i,j) \in M} \left(\vec{L}_i^T \cdot \vec{R}_j - M_{ij}\right)^2$
Agenda

• Big Model Analytics
• Gradient Descent Optimization
• Scalable HOGWILD! for Big Models
• Experimental Results
• Conclusions
Big Model Example 1

Recommender Systems

http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify

- Spotify applies low-rank matrix factorization (LMF) to 24 million users and 20 million songs which is 4.4 billion features at a relatively small rank of 100
### Text Analytics

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full sentence</strong></td>
<td>It does not, however, control whether an exaction is within Congress’s power to tax.</td>
</tr>
<tr>
<td><strong>Unigrams</strong></td>
<td>“It”; “does”; “not,”; “however,”; “control”; “whether”; “an”; “exaction”; “is”; “within”; “Congress’s”; “power”; “to”; “tax.”</td>
</tr>
<tr>
<td><strong>Bigrams</strong></td>
<td>“It does”; “does not”; “not, however,”; “however, control”; “control whether”; “whether an”; “an exaction”; “exaction is”; “is within”; “within Congress’s”; “Congress’s power”; “power to”; “to tax.”</td>
</tr>
<tr>
<td><strong>Trigrams</strong></td>
<td>“It does not”; “does not, however”; “not, however, control”; “however, control whether”; “control whether an”; “whether an exaction”; “an exaction is”; “exaction is within”; “is within Congress’s”; “within Congress’s power”; “Congress’s power to”; “power to tax.”</td>
</tr>
</tbody>
</table>

N-gram features of a sentence

- For the English Wikipedia corpus, a feature vector with **25 billion unigrams and 218 billion bigrams** can be constructed [S. Lee, J. K. Kim et al., “On Model Parallelization and Scheduling Strategies for Distributed Machine Learning”, NIPS 2015]
**Big Model Motivation**

**In the Cloud …**
- There is always enough memory
- You can always add more servers

**ML in IoT, Edge, and Fog Environments**
- Push processing to the devices acquiring the data which have rather scarce resources
- Data transfer is not a viable alternative for bandwidth and privacy reasons
- Secondary storage (disk, SSD, or flash) is plentiful
Support for Big Models

Existing ML Systems Do Not Support Big Models

- Model is an in-memory container data structure, e.g., vector or map
- Model is array attribute in a single-column table (at most 1 GB in PostgreSQL) and in-memory state of a UDA (User-Defined Aggregate)

Parameter Server [M. Li et al., OSDI 2014]

- Partition the model across the distributed shared memory of multiple servers, with each server storing a sufficiently small model partition that fits in its local memory
- Complex partitioning, replication, and synchronization

Dot-Product Join [C. Qin and F. Rusu, SSDBM 2017]

- Serial dot-product computation between sparse matrix and massive dense vector
- Range-based model partitioning independent of training data
- Training data reordering at chunk-level
In-Database Big Model ML

Problem

- Provide scalable in-database support for Big Model ML in a single multi-core server with attached storage, i.e., disk or SSD

Challenge

- Access to model is unpredictable and there are many accesses for each training example
- Highly-concurrent READ/WRITE model accesses across partitions (worker threads)
Agenda

- Big Model Analytics
- **Gradient Descent Optimization**
- Scalable HOGWILD! for Big Models
- Experimental Results
- Conclusions
Gradient Descent Optimization

\[
\min_{\vec{w} \in \mathbb{R}^d} \left\{ \Lambda(\vec{w}) = \sum_{i=1}^{N} f(\vec{w}, \vec{x}_i; y_i) \right\}
\]

\[
\vec{w}^{(k+1)} = \vec{w}^{(k)} - \alpha^{(k)} \nabla \Lambda(\vec{w}^{(k)})
\]

\(\alpha^{(k)}\) is step size or learning rate

\(\vec{w}^{(0)}\) is the starting point (random)

\[
\nabla \Lambda(\vec{w}) = \left[ \frac{\partial \Lambda(\vec{w})}{\partial w_1}, \ldots, \frac{\partial \Lambda(\vec{w})}{\partial w_d} \right]
\]

is the gradient

\[
\frac{\partial \Lambda_{LR}(\vec{w})}{\partial w_i} = \sum_{i=1}^{N} \left( -y_i e^{-y_i \vec{w} \cdot \vec{x}_i} + \frac{e^{-y_i \vec{w} \cdot \vec{x}_i}}{1 + e^{-y_i \vec{w} \cdot \vec{x}_i}} \right) \vec{x}_i
\]

\[
\frac{\partial \Lambda_{LMF}(L, R)}{\partial \vec{L}_{i'}} = \sum_{(i',j) \in M} \left( \vec{L}_{i'}^T \cdot \vec{R}_j - M_{i'j} \right) \vec{R}_j^T
\]

- Convergence to minimum guaranteed for convex objective function

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Stochastic Gradient Descent (SGD)

Input: \(\{(\vec{x}_i, y_i)\}_{1 \leq i \leq N}, f, \nabla f, \vec{w}^{(0)}, \alpha^{(0)}\)

Output: \(\vec{w}^{(k-1)}\)

1: Let \(k = 1\)
2: while (true) do
3: if convergence(\(\{\Lambda(\vec{w}^{(l)})\}_{0 \leq l < k}\)) then break
4: for each example \((\vec{x}_{\eta}^{(k)}, y_{\eta}^{(k)})\) do
5: Approximate gradient: \(\nabla f \left( \vec{w}^{(k)}, \vec{x}_{\eta}^{(k)}; y_{\eta}^{(k)} \right)\)
6: \(\vec{w}^{(k)} = \vec{w}^{(k)} - \alpha^{(k)} \nabla f \left( \vec{w}^{(k)}, \vec{x}_{\eta}^{(k)}; y_{\eta}^{(k)} \right)\)
7: end for
8: Update step size \(\alpha^{(k)}\)
9: Let \(k = k + 1\)
10: end while
11: return \(\vec{w}^{(k-1)}\)

http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
HOGWILD!: Parallel Asynchronous SGD

HOGWILD! Main Loop
1: for \( i = 1 \) to \( N \) do in parallel
2: \( \vec{w} \leftarrow \vec{w} - \alpha^{(k)} \nabla f (\vec{w}, \vec{x}_i; y_i) \)

- Parallelize inner loop while ignoring the sequential nature of SGD [F. Niu et al., NIPS 2011]
- Single model shared across threads
- Lock-free access to model: data races
  - 2 READ: gradient and model update
  - 1 WRITE: model update
- Convergence is preserved for sparse data
- Hardware cache coherence limits speedup [S. Sallinen et al., IPDPS 2016]

Naive Extension to Big Models
1: for \( i = 1 \) to \( N \) do in parallel
2: for each non-zero feature \( j \in \{1, \ldots, d\} \) in \( \vec{x}_i \) do
3: get \( \vec{w}[j] \)
4: compute \( \nabla f (\vec{w}[j], \vec{x}_i; y_i) \)
5: end for
6: \( \vec{w} \leftarrow \vec{w} - \alpha^{(k)} \nabla f (\vec{w}, \vec{x}_i; y_i) \)
7: for each non-zero feature \( j \in \{1, \ldots, d\} \) in \( \vec{x}_i \) do
8: put \( \vec{w}[j] \)
9: end for

- Model is stored in disk key-value store with get/put interface, e.g., (Hyper)LevelDB

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Agenda

• Big Model Analytics
• Gradient Descent Optimization
• **Scalable HOGWILD! for Big Models**
• Experimental Results
• Conclusions
Scalable HOGWILD! for Big Models

- **Offline** model vertical partitioning
  - Correlate indices to reduce number of get calls inside an example

- **Online** asynchronous model access sharing
  - Vertical partition traversal to reduce number of get calls inside a partition
  - Push-based model sharing to reduce number of get calls across partitions
  - Partition-level model update (mini-batch) to reduce number of put calls

---

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Offline Model Partitioning

- Composite key-value storage scheme
  - Two-level values in (key, value) intermediate representation

- Model vertical partitioning algorithm
  - Vertical partitioning in physical design: examples are workload; model is relation
  - Bottom-up greedy strategy with precomputed affinity matrix
  - Cost model combines seek time and payload access time
  - Sampling & frequency-based pruning

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Online Asynchronous Model Access Sharing

• On-the-fly example mapping into composite key-value scheme
• Vertical partition traversal
  – Incremental dot-products in gradient computation
• Push-based sharing of model indices across partitions
  – Key-to-vector inverted index (cuckoo hash table): single request across partition
• Partition-level (mini-batch) model updates preceded by get call

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Experimental Results (1)

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Experimental Results (2)

The diagrams illustrate the reduction in requests and time per iteration for different datasets and models. The x-axis represents the value of K, and the y-axis shows the number of requests reduced. The bars indicate the time per iteration for different datasets and models.

C. Qin, M. Torres, and F. Rusu – Scalable HOGWILD! for Big Models
Conclusions

• Design a scalable model and data-parallel framework for parallelizing stochastic optimization algorithms over big models: offline model partitioning and asynchronous online training

• Formalize model partitioning as vertical partitioning and design a scalable frequency-based model vertical partitioning algorithm

• Devise an asynchronous method to traverse vertically the training examples in all the data partitions

• Design a push-based model sharing mechanism for incremental gradient computation based on partial dot-products

• Implement the entire framework using User-Defined Aggregates (UDA) which provides generality across databases

• Evaluate the framework for three analytics tasks over synthetic and real datasets

• Results prove the scalability and reduced overhead incurred by model partitioning and key-value store
Thank you.
Questions ???