Parameter Server

Marco Serafini

COMPSCI 532
Lecture 19
Machine Learning

• Wide array of problems and algorithms
  • Classification
    • Given labeled data points, predict label of new data point
  • Regression
    • Learn a function from some (x, y) pairs
  • Clustering
    • Group data points into “similar” clusters
  • Segmentation
    • Partition image into meaningful segments
  • Outlier detection
More Dimensions

• Supervision:
  • Supervised ML: labeled ground truth is available
  • Unsupervised ML: no ground truth
• Training vs. Inference
  • Training: obtain model from training data
  • Inference: actually run the prediction
• Today we focus on the training problem
Example: Ad Click Predictor

• Ad prediction problem
  • A user is browsing the web
  • Choose ad that maximizes the likelihood of a click

• Training data
  • Trillions of ad-click log entries
  • Trillions of features per ad and user

• Important to reduce running time of training
  • Want to retrain frequently
  • Reduce energy and resource utilization costs
Abstracting ML Algorithms

• Can we find commonalities among ML algorithms?
• This would allow finding
  • Common abstractions
  • Systems solutions to efficiently implement these abstractions

• Some common aspects
  • We have a prediction model $A$
  • $A$ should optimize some complex *objective function* $L$
    • E.g.: Likelihood of correctly labeling a new ad as “click” or “no-click”
  • ML algorithm does this by iteratively refining $A$
High-Level View

• Notation
  • $D$: data
  • $A$: model parameters
  • $L$: function to optimize (e.g., minimize loss)
• Goal: Update $A$ based on $D$ to optimize $L$
• Typical approach: iterative convergence

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

merge updates to parameters

iteration $t$
compute updates that minimize $L$
How to Parallelize?

• How to execute the algorithm over a set of workers?
  • *Data-parallel* approach
    • Partition data $D$
    • All workers share the model parameters $A$
  • *Model-parallel* approach
    • Partition model parameters $A$
    • All workers process the same data $D$
How to Parallelize?

• How to execute the algorithm over a set of workers?
  • *Data-parallel approach*
    • Partition data $D$
    • All workers share the model parameters $A$
  • *Model-parallel approach*
    • Partition model parameters $A$
    • All workers process the same data $D$
Scaling Distributed Machine Learning with the Parameter Server

Mu Li*, David G. Andersen*, Jun Woo Park*, Alexander J. Smola†, Amr Ahmed†, Vanja Josifovski†, James Long†, Eugene J. Shekita†, Bor-Yiing Su†

* Carnegie Mellon University † Baidu ‡ Google

{muli, dga, junwoo}@cs.cmu.edu, alex@smola.org, {amra, vanjaj, jumlons, shekita, boryingsu}@google.com

Abstract

We propose a parameter server framework for distributed machine learning problems. Both data and workloads are distributed over worker nodes, while the server nodes maintain globally shared parameters, represented as dense or sparse vectors and matrices. The framework manages asynchronous data communication between nodes, and supports flexible consistency models, elastic scalability, and continuous fault tolerance.

To demonstrate the scalability of the proposed framework, we show experimental results on petabytes of real data with billions of examples and parameters on problems ranging from Sparse Logistic Regression to Latent Dirichlet Allocation and Distributed Sketching.

<table>
<thead>
<tr>
<th>≈ #machine × time</th>
<th># of jobs</th>
<th>failure rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 hours</td>
<td>13,187</td>
<td>7.8%</td>
</tr>
<tr>
<td>1,000 hours</td>
<td>1,366</td>
<td>13.7%</td>
</tr>
<tr>
<td>10,000 hours</td>
<td>77</td>
<td>24.7%</td>
</tr>
</tbody>
</table>

Table 1: Statistics of machine learning jobs for a three month period in a data center.

- At scale, fault tolerance is critical. Learning tasks are often performed in a cloud environment where machines can be unreliable and jobs can be preempted.

To illustrate the last point, we collected all job logs for a three month period from one cluster at a large internet company. We show statistics of batch machine learning jobs.
Data-Parallel Approach

\[ A^t = A^{(t-1)} + \sum_{p=1}^{P} \Delta(A^{(t-1)}, D_p) \]

- Process for each worker
  - Update parameters based on data
  - Push updates to **parameter servers**
  - Servers aggregate & apply updates
  - Pull parameters

- Requirements
  - Updates associative and commutative!
  - Example: Stochastic Gradient Descent
Example

• Each worker
  • Loads a partition of data
  • At every iteration, compute gradients

• Server
  • Aggregate gradients
  • Update parameters

---

**Algorithm 1 Distributed Subgradient Descent**

**Task Scheduler:**
1. issue `LoadData()` to all workers
2. for iteration $t = 0, \ldots, T$ do
3.   issue `WORKER_ITERATE(t)` to all workers.
4. end for

**Worker $r = 1, \ldots, m$:**
1. function `LOADDATA()`
2.   load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
3.   pull the working set $w_r^{(0)}$ from servers
4. end function
5. function `WORKER_ITERATE(t)`
6.   gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
7.   push $g_r^{(t)}$ to servers
8.   pull $w_r^{(t+1)}$ from servers
9. end function

**Servers:**
1. function `SERVER_ITERATE(t)`
2.   aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3.   $w^{(t+1)} \leftarrow w^{(t)} - \eta \left( g^{(t)} + \partial \Omega(w^{(t)}) \right)$
4. end function
Parameter Server

- Stores model parameters
- Advantages
  - No need for message passing
  - Distributed shared memory abstraction
- Very first implementation: key-value store
- Improvements by the work we read
  - Server-side UDFs
  - Worker scheduling
  - Bandwidth optimizations
Architecture

- Different namespaces
- Single parameters as <key, value> pairs
- Server-side linear algebra operations
  - Sum
  - Multiplication
  - 2-norm
Does This Scale?

• We said that a model can have trillion parameters
• Q: Does this scale?
• A: Yes
  • Each data point (worker) only updates few parameters
  • Example: Sparse Logistic Regression
Optimizing communication

- Machine learning is communication-heavy
- Ranges
  - Workers do not update single keys
  - Instead they batch updates per range
- Message compression
  - Worker-side caching of lists + send hash of lists
  - Don’t send zeroes
  - Snappy compression
  - Filtering: small updates are omitted (application-specific)
Tasks

• Activated by RPC: push or pull operations
• Executed asynchronously
• Users can specify the dependency of tasks
Flexible Consistency

- Typical semantics
  - Sequential
  - Eventual
  - Bounded delay

(a) Sequential  (b) Eventual  (c) 1 Bounded delay
Dependencies

• Vector clocks to express dependencies
  • Size: one entry per parameter per node is too large
  • Use instead one entry per range per node
  • Ranges are few and not split frequently
Consistent Hashing

- Server manager maintains the ring
- Other servers receive ranges
Replication

• Synchronous replication
  • Master pushes aggregated updates
  • When all replicas receive update, ack

• Replication after aggregation
  • Master waits until multiple updates are ready
Results: Sparse Logistic Regression

• Convergence and CPU utilization
Effect of Network Compression
Effect of Asynchrony

• Note: More asynchrony not always better
How to Parallelize?

• How to execute the algorithm over a set of workers?
  • *Data-parallel approach*
    • Partition data $D$
    • All workers share the model parameters $A$
  • *Model-parallel approach*
    • Partition model parameters $A$
    • All workers process the same data $D$
Petuum: A New Platform for Distributed Machine Learning on Big Data

Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanyu Kumar, and Yaoliang Yu

Abstract—What is a systematic way to efficiently apply a wide spectrum of advanced ML programs to industrial scale problems, using Big Models (up to 100 s of billions of parameters) on Big Data (up to terabytes or petabytes)? Modern parallelization strategies employ fine-grained operations and scheduling beyond the classic bulk-synchronous processing paradigm popularized by MapReduce, or even specialized graph-based execution that relies on graph representations of ML programs. The variety of approaches tends to pull systems and algorithms design in different directions, and it remains difficult to find a universal platform applicable to a wide range of ML programs at scale. We propose a general-purpose framework, Petuum, that systematically addresses data- and model-parallel challenges in large-scale ML, by observing that many ML programs are fundamentally optimization-centric and admit error-tolerant, iterative-convergent algorithmic solutions. This presents unique opportunities for an integrative system design, such as bounded-error network synchronization and dynamic scheduling based on ML program structure. We demonstrate the efficacy of these system designs versus well-known implementations of modern ML algorithms, showing that Petuum allows ML programs to run in much less time and at considerably larger model sizes, even on modestly-sized compute clusters.

Index Terms—Machine learning, big data, big model, distributed systems, theory, data-parallelism, model-parallelism

1 INTRODUCTION

MACHINE Learning (ML) is becoming a primary mechanism for extracting information from data. However, the surging volume of Big Data from Internet activities and sensory advancements, and the increasing needs for Big Models for ultra high-dimensional problems have put tremendous pressure on ML methods to scale beyond a single such big models with a single machine can be prohibitively slow, if not impossible. While careful model design and feature engineering can certainly reduce the size of the model, they require domain-specific expertise and are fairly labor-intensive, hence the recent appeal (as seen in the above papers) of building high-capacity Big Models in order to
Model-Parallel Approach

\[ A^t = A^{(t-1)} + Con \left( \{ \Delta_p (A^{(t-1)}, S_p^{(t-1)}(A^{(t-1)}), D) \}_{p=1}^P \right) \]

- Process for each worker
  - Receive ids of parameters \( S_p^{(t-1)} \) to update (from scheduler)
    - This is a partition of the entire space of parameters
  - Compute update on those parameters
  - Send updates to \textit{parameter server} that
    - \textit{Concatenates} updates (which are \textit{disjoint})
    - Applies updates to parameters

- Requirements
  - There should be no/weak correlation among parameters
  - Example: matrix factorization

- Q: Advantage?
Model-Parallel Scheduler

• Some systems (e.g. Petuum) support global scheduler
• Scheduler runs application-specific logic
• Two main goals
  • Partition parameters
  • Prioritized scheduling: give precedence to parameters that converge slower
Horovod: fast and easy distributed deep learning in TensorFlow

Alexander Sergeev  
Uber Technologies, Inc.  
asergeev@uber.com

Mike Del Balso  
Uber Technologies, Inc.  
mdb@uber.com

Abstract

Training modern deep learning models requires large amounts of computation, often provided by GPUs. Scaling computation from one GPU to many can enable much faster training and research progress but entails two complications. First, the training library must support inter-GPU communication. Depending on the particular methods employed, this communication may entail anywhere from negligible to significant overhead. Second, the user must modify his or her training
Horovod

• Use a ring topology among workers for aggregation
• Linear instead of quadratic number of messages
• Schedule non-overlapping updates
Scheduling Updates