15-712:
Advanced Operating Systems & Distributed Systems

MapReduce & TensorFlow

Prof. Phillip Gibbons

Spring 2020, Lecture 23
“MapReduce: Simplified Data Processing on Large Clusters”
Jeffrey Dean, Sanjay Ghemawat 2004

• Jeff Dean (Head, Google Brain)
  – ACM Prize in Computing, Mark Weiser Award
  – NAE, AAAS Fellow
  – ACM Fellow

• Sanjay Ghemawat (Google)
  – ACM Prize in Computing, Mark Weiser Award
  – NAE, AAAS Fellow
  – But not an ACM Fellow (?!)

2
The paper proposed a simple yet highly effective approach for processing large data sets in a scalable and fault-tolerant manner. An impressive aspect of the design is its simplicity: it elegantly captures a common pattern that solves two critical problems faced by many developers today (scalability and fault tolerance), while still retaining a clean, easy-to-use interface that supports a wide range of applications. The impact of MapReduce has been huge. It is widely used in industry, with virtually every large company running MapReduce. As a sign of great system design, developers have adopted MapReduce in many use cases beyond its original goals and inspired many follow-on systems.
Major Contributions

• A simple & powerful interface that enables automatic parallelization & distribution of large-scale computations
  – Uses Map & Reduce concepts from functional languages

• An implementation of this interface that achieves high performance on large clusters of commodity PCs

“Programmers without any experience with parallel & distributed systems can easily [in 30 mins] utilize the resources of a large distributed system.”
Programming Model

- Map: \((k1,v1) \rightarrow \text{list}(k2,v2)\)
- [Shuffle: group-by \(k2\)]
- Reduce: \((k2,\text{list}(v2)) \rightarrow \text{list}(v2)\)
o Input split pieces typically 16-64MBs
o Buffer then write to local disk, partitioned into regions
o Reduce-workers use RPC to remotely read from disks then sort
Fault Tolerance

- **When map-worker fails?**
  - Map-tasks re-assigned
  - Completed map-tasks re-executed (local disk lost)

- **When reduce-worker fails?**
  - Reduce-tasks re-assigned
  - Completed reduce-tasks are already in global file system

- **When master fails?**
  - Currently: abort MapReduce computation

- **Semantics on failure?**
  - When map & reduce are deterministic, semantically equivalent to a sequential execution
Locality & Stragglers

- **Locality**: Schedule map task (near) where data reside

- **Stragglers** (late finishing tasks):
  - Causes: Error correction on bad disk, multi-tenancy, bug in initialization code that disabled processor caches, etc
  - Solution: Fire off back-up tasks for remaining in-progress tasks
  - Why is duplicating work NOT a problem?
Experiences with MapReduce

• Big success: Widely used within Google
  – Large-scale machine learning, clustering for Google News & Froogle, popular queries reports, large-scale graph computations,…

• Complete rewrite of production indexing system for Google web search (20 TBs of crawled webpages)
  – Indexing code is simpler, smaller, easier to understand
  – Keep conceptually unrelated computations separate—makes easier to change indexing process
  – Ease of elasticity
Hadoop MapReduce overtaken by Spark (in-memory) in mid-2010s
“TensorFlow: A System for Large-Scale Machine Learning”

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng 2016

- **Martin Abadi** (Google) – NAE, ACM Fellow

......et al. 😊

a.k.a. “BigArray” in its anonymized OSDI’16 submission

Not an award winner...so why are we discussing it??? 😊
Primer: Machine Learning Training

• **Training formulated as an optimization problem**
  – Use training data to learn model parameters that minimize/maximize an objective function

• **No closed-form solution, instead algorithms iterate until convergence**
  – E.g., using Stochastic Gradient Descent

Image from charlesfranzen.com
Deep Neural Networks (DNNs)

**Goal:** Learn model parameters (weights on nodes/edges)

For each iteration (mini-batch):

- **Forward pass:**
  - Read batch of data
  - Compute class probabilities & measure loss

- **Backward pass:**
  - Based on loss, adjust parameters using gradients (back-propagation)
Parameter Servers for Distributed ML

- Provides all workers with convenient access to global model parameters
  - “Distributed shared memory” programming style

```
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)
}
```

[Power & Li, OSDI’10], [Ahmed et al, WSDM’12], [Ho et al, NIPS’13], [Li et al, OSDI’14], Petuum, MXNet, TensorFlow, etc
TensorFlow

• **Unified dataflow graph representing computation & state**
  – Nodes: individual mathematical operations
    (unit of local computation: matrix multiplication, convolution, etc)
  – Nodes own or update state
  – Edges: tensors (multi-dimensional arrays) passed between nodes

• **TensorFlow maps nodes across many machines & across many CPUs/GPUs/TPUs within a machine**

• **Goal: Power users can try out new models, training algorithms, optimizations**
High-Level Schematic of a TensorFlow Dataflow Graph
Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.
Layered Architecture

Training libraries | Inference libraries
Python client | C++ client ...

C API

Distributed master | Dataflow executor

Kernel implementations: Const, Var, MatMul, Conv2D, ReLU, Queue ...

Networking layer: RPC, RDMA ...

Device layer: CPU, GPU ...

18
TensorFlow Features

- Stateful operators: variables, queues
- Partial & concurrent execution
- Distributed execution
- Dynamic control flow (for RNNs)
- Fault tolerance
Synchronous Training w/Backup Worker
Training Very Large Models
Discussion: Summary Question #2

• Describe the paper's single most glaring deficiency. Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.
Performance on 1 Machine with 1 GPU

<table>
<thead>
<tr>
<th>Library</th>
<th>AlexNet</th>
<th>Overfeat</th>
<th>OxfordNet</th>
<th>GoogleNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe [38]</td>
<td>324</td>
<td>823</td>
<td>1068</td>
<td>1935</td>
</tr>
<tr>
<td>Neon [58]</td>
<td>87</td>
<td>211</td>
<td>320</td>
<td>270</td>
</tr>
<tr>
<td>Torch [17]</td>
<td>81</td>
<td>268</td>
<td>529</td>
<td>470</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>81</td>
<td>279</td>
<td>540</td>
<td>445</td>
</tr>
</tbody>
</table>
Coordination Overheads

Throughput for Synchronous training with “null training” steps
Figure 8: Results of the performance evaluation for Inception-v3 training (§6.3). (a) TensorFlow achieves slightly better throughput than MXNet for asynchronous training. (b) Asynchronous and synchronous training throughput increases with up to 200 workers. (c) Adding backup workers to a 50-worker training job can reduce the overall step time, and improve performance even when normalized for resource consumption.
Next word prediction. 40K word vocabulary. Sampled softmax uses 78x less data transfer & computation.
Limitations

• Non-power users suffer
  – Need better automatic tools / default policies

• Inadequate support for strong consistency

• Dataflow graphs are static
“Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data”

Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, Emmett Witchel 2016