Living with Big Data: Challenges and Opportunities

Jeff Dean, Sanjay Ghemawat
Google

Joint work with many collaborators
Computational Environment

- Many datacenters around the world
Zooming In...
Decomposition into Services

query

Frontend Web Server

Ad System

Super root

Spelling correction

Images

Local

News

Web

Video

Blogs

Books

Storage

Scheduling

Naming

...
Communication Protocols

• Example:
  – Request: query: “ethiopiaan restaurnnts”
  – Response: list of (corrected query, score) results
    correction { query: “ethiopian restaurants” score: 0.97 }
    correction { query: “ethiopia restaurants” score: 0.02 }
    ...

• Benefits of structure:
  – easy to examine and evolve (add user_language to request)
  – language independent
  – teams can operate independently

• We use Protocol Buffers for RPCs, storage, etc.
  – http://code.google.com/p/protobuf/
The Horrible Truth...

Typical first year for a new cluster:

~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

Friday, September 14, 2012
The Horrible Truth...

Typical first year for a new cluster:

~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

• Reliability/availability must come from software!
Replication

• Data loss
  – replicate the data on multiple disks/machines (GFS/Colossus)

• Slow machines
  – replicate the computation (MapReduce)

• Too much load
  – replicate for better throughput (nearly all of our services)

• Bad latency
  – utilize replicas to improve latency
  – improved worldwide placement of data and services
Shared Environment

Linux
Shared Environment

file system
chunkserver

Linux
Shared Environment

- file system
- chunkserver
- scheduling system
- Linux
Shared Environment

- various other system services
- file system
- chunkserver
- scheduling system
- Linux
Shared Environment

- Bigtable tablet server
- Various other system services
- File system chunkserver
- Scheduling system
- Linux
Shared Environment

- CPU intensive job
- Random MapReduce #1
- Bigtable tablet server
- Various other system services
- File system chunkserver
- Scheduling system
- Linux
Shared Environment

- random app #2
- cpu intensive job
- random MapReduce #1
- Bigtable tablet server
- various other system services
- file system chunkserver
- scheduling system
- Linux
Shared Environment

• **Huge benefit: greatly increased utilization**

• ... but hard to predict effects increase variability
  – network congestion
  – background activities
  – bursts of foreground activity
  – not just your jobs, but everyone else’s jobs, too
  – not static: change happening constantly

• **Exacerbated by large fanout systems**
The Problem with Shared Environments
The Problem with Shared Environments
• Server with 10 ms avg. but 1 sec 99%ile latency
  – touch 1 of these: 1% of requests take $\geq 1$ sec
  – touch 100 of these: 63% of requests take $\geq 1$ sec
Tolerating Faults vs. Tolerating Variability

• Tolerating faults:
  – rely on extra resources
    • RAIDed disks, ECC memory, dist. system components, etc.
    – make a reliable whole out of unreliable parts

• Tolerating variability:
  – use these same extra resources
    – make a predictable whole out of unpredictable parts

• Times scales are very different:
  – variability: 1000s of disruptions/sec, scale of milliseconds
  – faults: 10s of failures per day, scale of tens of seconds

Friday, September 14, 2012
Latency Tolerating Techniques

• Cross request adaptation
  – examine recent behavior
  – take action to improve latency of future requests
  – typically relate to balancing load across set of servers
  – time scale: 10s of seconds to minutes

• Within request adaptation
  – cope with slow subsystems in context of higher level request
  – time scale: right now, while user is waiting

• Many such techniques
  [The Tail at Scale, Dean & Barroso, to appear in CACM late 2012/early 2013]
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.
Tied Requests

Each request identifies other server(s) to which request might be sent

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.
Tied Requests

Similar to Michael Mitzenmacher's work on "The Power of Two Choices", except send to both, rather than just picking "best" one.

Each request identifies other server(s) to which request might be sent.
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.

Each request identifies other server(s) to which request might be sent.
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.

Each request identifies other server(s) to which request might be sent.
Tied Requests

---

Each request identifies other server(s) to which request might be sent.

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one.

Each request identifies other server(s) to which request might be sent.
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one

Each request identifies other server(s) to which request might be sent
Tied Requests

Similar to Michael Mitzenmacher’s work on “The Power of Two Choices”, except send to both, rather than just picking “best” one

Each request identifies other server(s) to which request might be sent
Tied Requests: Bad Case

Server 1

req 3

Server 2

req 5

Client
Tied Requests: Bad Case

Server 1

req 3

req 9

Server 2

req 5

Friday, September 14, 2012
Tied Requests: Bad Case

Server 1

- req 3
- req 9
  also: server 2

Server 2

- req 5

req 9
Tied Requests: Bad Case

Server 1
- req 3
- req 9
  also: server 2

Server 2
- req 5
- req 9
  also: server 1

Client
Tied Requests: Bad Case

Server 1
- req 9
  - also: server 2

Server 2
- req 9
  - also: server 1

Client
Tied Requests: Bad Case

Server 1

```
req 9
also: server 2
```

Server 2

```
req 9
also: server 1
```

Client

“Server 1: Starting req 9”

“Server 2: Starting req 9”

Friday, September 14, 2012
Tied Requests: Bad Case

Server 1

```
req 9
also: server 2
```

“Server 2: Starting req 9”

Client

Server 2

```
req 9
also: server 1
```

“Server 1: Starting req 9”
Tied Requests: Bad Case

Server 1

req 9
also: server 2

Client

Server 2

reply

req 9
also: server 1
Tied Requests: Bad Case

Server 1
- req 9
  also: server 2

Server 2
- req 9
  also: server 1

reply
Tied Requests: Bad Case

Server 1

req 9
also: server 2

Server 2

req 9
also: server 1

reply

Likelihood of this bad case is reduced with lower latency networks
Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk
Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td>Backup after 2 ms</td>
<td></td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
</tbody>
</table>
Tied Requests: Performance Benefits

• Read operations in distributed file system client
  – send tied request to first replica
  – wait 2 ms, and send tied request to second replica
  – servers cancel tied request on other replica when starting read
• Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
</tbody>
</table>
Tied Requests: Performance Benefits

• Read operations in distributed file system client
  – send tied request to first replica
  – wait 2 ms, and send tied request to second replica
  – servers cancel tied request on other replica when starting read

• Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
<tr>
<td>+Terasort</td>
<td>No backups</td>
<td>24 ms</td>
<td>56 ms</td>
<td>108 ms</td>
<td>159 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>19 ms</td>
<td>35 ms</td>
<td>67 ms</td>
<td>108 ms</td>
</tr>
</tbody>
</table>
Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
<tr>
<td>+Terasort</td>
<td>No backups</td>
<td>24 ms</td>
<td>56 ms</td>
<td>108 ms</td>
<td>159 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>19 ms</td>
<td>35 ms</td>
<td>67 ms</td>
<td>108 ms</td>
</tr>
</tbody>
</table>
Tied Requests: Performance Benefits

- Read operations in distributed file system client
  - send tied request to first replica
  - wait 2 ms, and send tied request to second replica
  - servers cancel tied request on other replica when starting read
- Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
<tr>
<td>+Terasort</td>
<td>No backups</td>
<td>24 ms</td>
<td>56 ms</td>
<td>108 ms</td>
<td>159 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>19 ms</td>
<td>35 ms</td>
<td>67 ms</td>
<td>108 ms</td>
</tr>
</tbody>
</table>

Backups cause about ~1% extra disk reads
Read operations in distributed file system client
  – send tied request to first replica
  – wait 2 ms, and send tied request to second replica
  – servers cancel tied request on other replica when starting read

Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
<tr>
<td>+Terasort</td>
<td>No backups</td>
<td>24 ms</td>
<td>56 ms</td>
<td>108 ms</td>
<td>159 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>19 ms</td>
<td>35 ms</td>
<td>67 ms</td>
<td>108 ms</td>
</tr>
</tbody>
</table>
Tied Requests: Performance Benefits

• Read operations in distributed file system client
  – send tied request to first replica
  – wait 2 ms, and send tied request to second replica
  – servers cancel tied request on other replica when starting read
• Measure higher-level monitoring ops that touch disk

<table>
<thead>
<tr>
<th>Cluster state</th>
<th>Policy</th>
<th>50%ile</th>
<th>90%ile</th>
<th>99%ile</th>
<th>99.9%ile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly idle</td>
<td>No backups</td>
<td>19 ms</td>
<td>38 ms</td>
<td>67 ms</td>
<td>98 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>16 ms</td>
<td>28 ms</td>
<td>38 ms</td>
<td>51 ms</td>
</tr>
<tr>
<td>+Terasort</td>
<td>No backups</td>
<td>24 ms</td>
<td>56 ms</td>
<td>108 ms</td>
<td>159 ms</td>
</tr>
<tr>
<td></td>
<td>Backup after 2 ms</td>
<td>19 ms</td>
<td>35 ms</td>
<td>67 ms</td>
<td>108 ms</td>
</tr>
</tbody>
</table>

Backups w/big sort job gives same read latencies as no backups w/ idle cluster!
Cluster-Level Services

- Our earliest systems made things easier within a cluster:
  - GFS/Colossus: reliable cluster-level file system
  - MapReduce: reliable large-scale computations
  - Cluster scheduling system: abstracted individual machines
  - BigTable: automatic scaling of higher-level structured storage
Cluster-Level Services

• Our earliest systems made things easier within a cluster:
  – GFS/Colossus: reliable cluster-level file system
  – MapReduce: reliable large-scale computations
  – Cluster scheduling system: abstracted individual machines
  – BigTable: automatic scaling of higher-level structured storage

• Solve many problems, but leave many cross-cluster issues to human-level operators
  – different copies of same dataset have different names
  – moving or deploying new service replicas is labor intensive
Spanner: Worldwide Storage

• Single global namespace for data
• Consistent replication across datacenters
• Automatic migration to meet various constraints
  – resource constraints
    “The file system in this Belgian datacenter is getting full...”
  – application-level hints
    “Place this data in Europe and the U.S.”
    “Place this data in flash, and place this other data on disk”
Spanner: Worldwide Storage

• Single global namespace for data
• Consistent replication across datacenters
• Automatic migration to meet various constraints
  – resource constraints
    “The file system in this Belgian datacenter is getting full...”
  – application-level hints
    “Place this data in Europe and the U.S.”
    “Place this data in flash, and place this other data on disk”

• System underlies Google’s production advertising system, among other uses

• [Spanner: Google’s Globally-Distributed Database, Corbett, Dean, ..., Ghemawat, ... et al., to appear in OSDI 2012]
Questions you might want to ask:
– did this change I rolled out last week affect # of errors / request?
– why are my tasks using so much memory?
– where is CPU time being spent in my application?
– what kinds of requests are being handled by my service?
– why are some requests very slow?

Important to have enough visibility into systems to answer these kinds of questions
## Exported Variables

- **Special URL on every Google server**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rpc-server-count-minute</code></td>
<td>11412</td>
</tr>
<tr>
<td><code>rpc-server-count</code></td>
<td>502450983</td>
</tr>
<tr>
<td><code>rpc-server-arg-bytes-minute</code></td>
<td>8039419</td>
</tr>
<tr>
<td><code>rpc-server-arg-bytes</code></td>
<td>372908296166</td>
</tr>
<tr>
<td><code>rpc-server-rpc-errors-minute</code></td>
<td>0</td>
</tr>
<tr>
<td><code>rpc-server-rpc-errors</code></td>
<td>0</td>
</tr>
<tr>
<td><code>rpc-server-app-errors-minute</code></td>
<td>8</td>
</tr>
<tr>
<td><code>rpc-server-app-errors</code></td>
<td>2357783</td>
</tr>
<tr>
<td><code>uptime-in-ms</code></td>
<td>679532636</td>
</tr>
<tr>
<td><code>build-timestamp-as-int</code></td>
<td>1343415737</td>
</tr>
<tr>
<td><code>build-timestamp</code></td>
<td>&quot;Built on Jul 27 2012 12:02:17 (1343415737)&quot;</td>
</tr>
</tbody>
</table>

- **On top of this, we have systems that gather all of this data**
  - can aggregate across servers & services, compute derived values, graph data, examine historical changes, etc.
Every server supports sampling-based hierarchical profiling
  – CPU
  – memory usage
  – lock contention time

Example: memory sampling
  – every Nth byte allocated, record stack trace of where allocation occurred
  – when sampled allocation is freed, drop stack trace
  – (N is large enough that overhead is small)
Memory Profile
Request Tracing

• Every client and server gathers sample of requests
  – different sampling buckets, based on request latency

2012/09/09-11:39:21.029630  0.018978 Read (trace_id: c6143c073204f13f ...)
11:39:21.029611  -0.000019 ... RPC: 07eb70184bfff86f ... deadline:0.8526s
11:39:21.029611  -0.000019 ... header:<path:"..." length:33082 offset:3037807
11:39:21.029729        .    99 ... StartRead(..., 3037807, 33082)
11:39:21.029730        .     1 ... ContentLock
11:39:21.029732        .     2 ... GotContentLock
...
11:39:21.029916        .     2 ... IssueRead
11:39:21.048196        . 18280 ... HandleRead: OK
11:39:21.048666        .  431 ... RPC: OK [33082 bytes]
Request Tracing

• Every client and server gathers sample of requests
  – different sampling buckets, based on request latency

```
2012/09/09-11:39:21.029630       0.018978 Read (trace_id: c6143c073204f13f ...)
11:39:21.029611      -0.000019 ... RPC: 07eb70184bfff86f ... deadline:0.8526s
11:39:21.029611      -0.000019 ... header:<path:"..." length:33082 offset:3037807
11:39:21.029729        .    99 ... StartRead(..., 3037807, 33082)
11:39:21.029730        .     1 ... ContentLock
11:39:21.029732        .     2 ... GotContentLock
...
11:39:21.029916        .     2 ... IssueRead
11:39:21.048196        . 18280 ... HandleRead: OK
11:39:21.048666        .  431 ... RPC: OK [33082 bytes]
```

• Dapper: cross-machine view of preceding information
  – can understand complex behavior across many services
  – [Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Sigelman et al., 2010]
Higher Level Systems

• Systems that provide high level of abstraction that “just works” are incredibly valuable:
  • GFS, MapReduce, BigTable, Spanner, transparent latency reduction techniques, etc.

• Can we build high-level systems that just work in other domains like machine learning?
Scaling Deep Learning

• Much of Google is working on approximating AI. AI is hard
  • Many people at Google spend countless person-years hand-engineering complex features to feed as input to machine learning algorithms

• Is there a better way?

• Deep Learning: Use very large scale brain simulations
  • improve many Google applications
  • make significant advances towards perceptual AI
Deep Learning

• Algorithmic approach
  • automatically learn high-level representations from raw data
  • can learn from both labeled and unlabeled data

• Recent academic deep learning results improve on state-of-the-art in many areas:
  • images, video, speech, NLP, ...
  • ... using modest model sizes (<= ~50M parameters)

• We want to scale this approach up to much bigger models
  • currently: ~2B parameters, want ~10B-100B parameters
  • general approach: parallelize at many levels
Deep Networks

Input Image
(or video)
Deep Networks
Deep Networks

Some scalar, nonlinear function of local image patch

Input Image  
(or video)
Deep Networks

Some scalar, nonlinear function of local image patch

Input Image
(or video)
Deep Networks

Some scalar, nonlinear function of local image patch

Input Image
(or video)
Deep Networks

Many responses at a single location. In many models these are independent, but some allow strong nonlinear interactions.

Some scalar, nonlinear function of local image patch.
Deep Networks

Input Image
(or video)
Deep Networks

Input Image
(or video)
Deep Networks

Multiple “maps”

Input Image
(or video)
Deep Networks
Unsupervised Training

Core idea: try to reconstruct input from just the learned representation

Due to Geoff Hinton, Yoshua Bengio, Andrew Ng, and others
Layer 1

Input Image
(or video)
Traditional ML tools

Layer 1

Layer 2

Input Image (or video)

Output feature vector
Partition model across machines

Partition assignment in vertical silos.

Layer 0

Layer 1

Layer 2

Layer 3
Partition model across machines

Minimal network traffic: The most densely connected areas are on the same partition

Partition assignment in vertical silos.
Partition model across machines

Partition assignment in vertical silos.

Minimal network traffic: The most densely connected areas are on the same partition

One replica of our biggest models: 144 machines, ~2300 cores
Basic Model Training

- Unsupervised or Supervised Objective
- Minibatch Stochastic Gradient Descent (SGD)
- Model parameters sharded by partition
- 10s, 100s, or 1000s of cores per model
Basic Model Training

Making a single model bigger and faster is the right first step.

But training still slow with large data sets/model with a single model replica.

How can we add another dimension of parallelism, and have multiple model instances train on data in parallel?
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

Model

Data

Friday, September 14, 2012
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

Model

Data
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

Model

Data

$\Delta \phi$
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server \[ p' = p + \Delta p \]
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

Model

Data

$p'$
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server

\[ \Delta \rho' \]

Model

Data
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server \( p'' = p' + \Delta p' \)
Asynchronous Distributed Stochastic Gradient Descent

Parameter Server  \( p' = p + \Delta p \)

Model Workers

Data Shards

Friday, September 14, 2012
Training System

• Some aspects of asynchrony and distribution similar to some recent work:

  Slow Learners are Fast
  John Langford, Alexander J. Smola, Martin Zinkevich, NIPS 2009

  Distributed Delayed Stochastic Optimization
  Alekh Agarwal, John Duchi, NIPS 2011

  Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent
  Feng Niu, Benjamin Recht, Christopher Re, Stephen J. Wright, NIPS 2011

• Details of our system to appear:

  [Large Scale Distributed Deep Networks, Dean et al., to appear in NIPS 2012]
Deep Learning Systems Tradeoffs

• Lots of tradeoffs can be made to improve performance. Which ones are possible without hurting learning performance too much?

• For example:
  • Use lower precision arithmetic
  • Send 1 or 2 bits instead of 32 bits across network
  • Drop results from slow partitions

• What’s the right hardware for training and deploying these sorts of systems?
  • GPUs? FPGAs? Lossy computational devices?
Applications

• Acoustic Models for Speech
• Unsupervised Feature Learning for Still Images
• Neural Language Models
Acoustic Modeling for Speech Recognition

8000-label Softmax

One or more hidden layers of a few thousand nodes each.

Frames of 40-value Log Energy Power Spectra and the label for central frame

Close collaboration with Google Speech team

Trained in <5 days on cluster of 800 machines
Acoustic Modeling for Speech Recognition

8000-label Softmax

One or more hidden layers of a few thousand nodes each.

11 Frames of 40-value Log Energy Power Spectra and the label for central frame

Close collaboration with Google Speech team

Trained in <5 days on cluster of 800 machines

Major reduction in Word Error Rate ("equivalent to 20 years of speech research")
Acoustic Modeling for Speech Recognition

8000-label Softmax

One or more hidden layers of a few thousand nodes each.

Frames of 40-value Log Energy Power Spectra and the label for central frame

Close collaboration with Google Speech team

Trained in <5 days on cluster of 800 machines

Major reduction in Word Error Rate (“equivalent to 20 years of speech research”)

Deployed in Jellybean release of Android

Friday, September 14, 2012
Applications

• Acoustic Models for Speech

• Unsupervised Feature Learning for Still Images

• Neural Language Models
Purely Unsupervised Feature Learning in Images

60,000 neurons at top level

- 1.15 billion parameters (50x larger than largest deep network in the literature)
- Trained on 16k cores for 1 week using Async-SGD
- Do unsupervised training on one frame from each of 10 million YouTube videos (200x200 pixels)
- No labels!

Details in our ICML paper [Le et al. 2012]
Top level neurons seem to discover high-level concepts. For example, one neuron is a decent face detector:
Purely Unsupervised Feature Learning in Images

Most face-selective neuron

Top 48 stimuli from the test set
Purely Unsupervised Feature Learning in Images

Most face-selective neuron

Top 48 stimuli from the test set

Optimal stimulus by numerical optimization
Purely Unsupervised Feature Learning in Images

It is YouTube... We also have a cat neuron!

Top stimuli from the test set
Purely Unsupervised Feature Learning in Images

It is YouTube... We also have a cat neuron!

Top stimuli from the test set

Optimal stimulus
We made a cat detector!

It uses a few CPUs!
Semi-supervised Feature Learning in Images

Are the higher-level representations learned by unsupervised training a useful starting point for supervised training?

We do have some labeled data, so let’s fine tune this same network for a challenging image classification task.
Semi-supervised Feature Learning in Images

Are the higher-level representations learned by unsupervised training a useful starting point for supervised training?

We do have some labeled data, so let’s fine tune this same network for a challenging image classification task.

**ImageNet:**
- 16 million images
- ~21,000 categories
- Recurring academic competitions
Aside: 20,000 is a lot of categories....
Aside: 20,000 is a lot of categories....

- roughtail stingray
- manta ray

Aside: 20,000 is a lot of categories....
Semi-supervised Feature Learning in Images

ImageNet Classification Results:
- ImageNet 2011 (20k categories)
  - Chance: 0.005%
  - Best reported: 9.5%
  - Our network: 16% (+70% relative)
Semi-supervised Feature Learning in Images

Example top stimuli after fine tuning on ImageNet:
Semi-supervised Feature Learning in Images

Example top stimuli after fine tuning on ImageNet:
Semi-supervised Feature Learning in Images

Example top stimuli after fine tuning on ImageNet:

Neuron 10

Neuron 11

Neuron 12

Neuron 13
Applications

• Acoustic Models for Speech
• Unsupervised Feature Learning for Still Images
• Neural Language Models
Embeddings

~100-D joint embedding space

porpoise    dolphin
Embeddings

~100-D joint embedding space

porpoise

dolphin
Embeddings

~100-D joint embedding space

porpoise

dolphin

SeaWorld
Embeddings

~100-D joint embedding space

Obama

porpoise

dolphin

SeaWorld
Embeddings

~100-D joint embedding space

Obama

porpoise

dolphin

SeaWorld

Paris
Neural Language Models

Hinge Loss // Softmax

Hidden Layers?

Word Embedding Matrix

\[ E \]

is a matrix of dimension \(|Vocab| \times d\)

Top prediction layer has \(|Vocab| \times h\) parameters.

Most ideas from Bengio et al 2003, Collobert & Weston 2008
Neural Language Models

Hinge Loss // Softmax

Hidden Layers?

Word Embedding Matrix

\[ E \]

is a matrix of dimension \( ||\text{Vocab}|| \times d \)

Top prediction layer has \( ||\text{Vocab}|| \times h \) parameters.

100s of millions of parameters, but gradients very sparse

Most ideas from Bengio et al 2003, Collobert & Weston 2008
Embedding sparse tokens in an N-dimensional space

Example: 50-D embedding trained for semantic similarity

Cluster 1: apple

<table>
<thead>
<tr>
<th>Id</th>
<th>Distance</th>
<th>Adjust</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>11114</td>
<td>0.000000</td>
<td>Remove</td>
<td>apple</td>
</tr>
<tr>
<td>5026</td>
<td>0.652580</td>
<td>Add</td>
<td>fruit</td>
</tr>
<tr>
<td>14080</td>
<td>0.699192</td>
<td>Add</td>
<td>apples</td>
</tr>
<tr>
<td>48657</td>
<td>0.717818</td>
<td>Add</td>
<td>melon</td>
</tr>
<tr>
<td>28498</td>
<td>0.722390</td>
<td>Add</td>
<td>peach</td>
</tr>
<tr>
<td>39795</td>
<td>0.729893</td>
<td>Add</td>
<td>blueberry</td>
</tr>
<tr>
<td>35570</td>
<td>0.730500</td>
<td>Add</td>
<td>berry</td>
</tr>
<tr>
<td>25974</td>
<td>0.739561</td>
<td>Add</td>
<td>strawberry</td>
</tr>
<tr>
<td>46156</td>
<td>0.745343</td>
<td>Add</td>
<td>pecan</td>
</tr>
<tr>
<td>11907</td>
<td>0.756422</td>
<td>Add</td>
<td>potato</td>
</tr>
<tr>
<td>33847</td>
<td>0.759111</td>
<td>Add</td>
<td>pear</td>
</tr>
<tr>
<td>30895</td>
<td>0.763317</td>
<td>Add</td>
<td>mango</td>
</tr>
<tr>
<td>17848</td>
<td>0.768230</td>
<td>Add</td>
<td>pumpkin</td>
</tr>
<tr>
<td>39133</td>
<td>0.770143</td>
<td>Add</td>
<td>almond</td>
</tr>
<tr>
<td>14395</td>
<td>0.773105</td>
<td>Add</td>
<td>tomato</td>
</tr>
<tr>
<td>18163</td>
<td>0.782610</td>
<td>Add</td>
<td>onion</td>
</tr>
<tr>
<td>10470</td>
<td>0.782994</td>
<td>Add</td>
<td>pie</td>
</tr>
<tr>
<td>3023</td>
<td>0.787229</td>
<td>Add</td>
<td>tree</td>
</tr>
<tr>
<td>20340</td>
<td>0.793602</td>
<td>Add</td>
<td>bean</td>
</tr>
<tr>
<td>34968</td>
<td>0.794979</td>
<td>Add</td>
<td>watermelon</td>
</tr>
</tbody>
</table>
Embedding sparse tokens in an N-dimensional space

Example: 50-D embedding trained for semantic similarity

apple  
stab
Embedding sparse tokens in an N-dimensional space

Example: 50-D embedding trained for semantic similarity

<table>
<thead>
<tr>
<th>Cluster 1</th>
<th>Cluster 1</th>
<th>Cluster 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>apple</strong></td>
<td><strong>stab</strong></td>
<td><strong>iPhone</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Distance</th>
<th>Adjust</th>
<th>Word</th>
<th>Id</th>
<th>Distance</th>
<th>Adjust</th>
<th>Word</th>
<th>Id</th>
<th>Distance</th>
<th>Adjust</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>11114</td>
<td>0.000000</td>
<td>Remove</td>
<td>apple</td>
<td>14979</td>
<td>0.000000</td>
<td>Remove</td>
<td>stab</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5026</td>
<td>0.652580</td>
<td>Add</td>
<td>fruit</td>
<td>7728</td>
<td>0.868853</td>
<td>Add</td>
<td>punch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14080</td>
<td>0.699192</td>
<td>Add</td>
<td>apples</td>
<td>469</td>
<td>0.909304</td>
<td>Add</td>
<td>shot</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48657</td>
<td>0.717818</td>
<td>Add</td>
<td>melon</td>
<td>12820</td>
<td>0.907950</td>
<td>Add</td>
<td>thrust</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28498</td>
<td>0.722390</td>
<td>Add</td>
<td>peach</td>
<td>8934</td>
<td>0.939908</td>
<td>Add</td>
<td>shell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39795</td>
<td>0.729893</td>
<td>Add</td>
<td>blueberry</td>
<td>10880</td>
<td>0.951466</td>
<td>Add</td>
<td>hammer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35570</td>
<td>0.730500</td>
<td>Add</td>
<td>berry</td>
<td>6975</td>
<td>0.951679</td>
<td>Add</td>
<td>bullet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25974</td>
<td>0.739561</td>
<td>Add</td>
<td>strawberry</td>
<td>1848</td>
<td>0.962053</td>
<td>Add</td>
<td>push</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46156</td>
<td>0.745343</td>
<td>Add</td>
<td>pecan</td>
<td>10888</td>
<td>0.962319</td>
<td>Add</td>
<td>eyed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11907</td>
<td>0.756422</td>
<td>Add</td>
<td>potato</td>
<td>718</td>
<td>0.965448</td>
<td>Add</td>
<td>hand</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33847</td>
<td>0.759111</td>
<td>Add</td>
<td>pear</td>
<td>5865</td>
<td>0.966663</td>
<td>Add</td>
<td>grab</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30895</td>
<td>0.763317</td>
<td>Add</td>
<td>mango</td>
<td>4611</td>
<td>0.967574</td>
<td>Add</td>
<td>swing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17848</td>
<td>0.768230</td>
<td>Add</td>
<td>pumpkin</td>
<td>302</td>
<td>0.975696</td>
<td>Add</td>
<td>hit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39133</td>
<td>0.770143</td>
<td>Add</td>
<td>almond</td>
<td>869</td>
<td>0.976967</td>
<td>Add</td>
<td>force</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14395</td>
<td>0.773105</td>
<td>Add</td>
<td>tomato</td>
<td>1597</td>
<td>0.977625</td>
<td>Add</td>
<td>attempt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18163</td>
<td>0.782610</td>
<td>Add</td>
<td>onion</td>
<td>5977</td>
<td>0.978384</td>
<td>Add</td>
<td>finger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10470</td>
<td>0.782994</td>
<td>Add</td>
<td>pie</td>
<td>6162</td>
<td>0.978776</td>
<td>Add</td>
<td>knife</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3023</td>
<td>0.787229</td>
<td>Add</td>
<td>tree</td>
<td>3434</td>
<td>0.980028</td>
<td>Add</td>
<td>sharp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20340</td>
<td>0.793602</td>
<td>Add</td>
<td>bean</td>
<td>1504</td>
<td>0.980160</td>
<td>Add</td>
<td>struck</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34968</td>
<td>0.794979</td>
<td>Add</td>
<td>watermelon</td>
<td>39157</td>
<td>0.980219</td>
<td>Add</td>
<td>slug</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Friday, September 14, 2012
Neural Language Models

- 7 Billion word Google News training set
- 1 Million word vocabulary
- 8 word history, 50 dimensional embedding
- Three hidden layers each w/200 nodes
- 50-100 asynchronous model workers
Neural Language Models

- 7 Billion word Google News training set
- 1 Million word vocabulary
- 8 word history, 50 dimensional embedding
- Three hidden layers each w/200 nodes
- 50-100 asynchronous model workers

Perplexity Scores

<table>
<thead>
<tr>
<th></th>
<th>Traditional 5-gram</th>
<th>NLM</th>
<th>5-gram + NLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity Scores</td>
<td>XXX</td>
<td>+15%</td>
<td>-33%</td>
</tr>
</tbody>
</table>

Friday, September 14, 2012
Deep Learning Applications

Many other applications not discussed today:

• Clickthrough prediction for advertising
• Video understanding
• User action prediction

...
Thanks! Questions...?

Further reading:


• Le, Ranzato, Monga, Devin, Chen, Corrado, Dean, & Ng. Building High-Level Features Using Large Scale Unsupervised Learning, ICML 2012.

• Dean et al., Large Scale Distributed Deep Networks, to appear NIPS 2012.


• Dean & Barroso, The Tail at Scale, to appear in CACM 2012/2013.

• Snappy. http://code.google.com/p/snappy/
• LevelDB. http://code.google.com/p/leveldb/

These and many more available at: http://labs.google.com/papers.html