Operating Systems and The Cloud, Part II: Search => Cluster Apps => Scalable Machine Learning

David E. Culler
CS162 – Operating Systems and Systems Programming
Lecture 40
December 3, 2014
The Data Center as a System

- Clusters became THE architecture for large scale internet services
  - Distribute disks, files, I/O, net, and compute over everything
  - Massive AND Incremental scalability
- Search Engines the initial “Killer App”
- Multiple components as Cluster Apps
  - Web crawl, Index, Search & Rank, Network, ...
- Global Layer as a Master/Worker pattern
  - GFS, HDFS
- Map Reduce framework address core of search on massive scale – and much more
  - Indexing, log analysis, data querying
  - Collating, inverted indexes: map(k,v) => f(k,v),(k,v)
  - Filtering, Parsing, Validation
  - Sorting

Lessons from Giant-Scale Services, Eric Brewer, IEEE Computer, Jul 2001
Research ...

MapReduce:
Simplified Data Processing on Large Clusters

Jeff Dean, Sanjay Ghemawat
Google, Inc.

Related Work

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort [’97]
- Re-execution for fault tolerance
  - BAD-FS [’04] and TACC [’97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks [’01], Diamond [’04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte [’96]
- Dynamic load balancing solves similar problem as River’s distributed queues
  - River [’99]
The Data Center as a System

• Clusters became THE architecture for large scale internet services
  - Distribute disks, files, I/O, net, and compute over everything
  - Massive AND Incremental scalability

• Search Engines the initial “Killer App”

• Multiple components as Cluster Apps
  - Web crawl, Index, Search & Rank, Network, …

• Global Layer as a Master/Worker pattern
  - GFS, HDFS

• Map Reduce framework address core of search on massive scale – and much more
  - Indexing, log analysis, data querying
  - Collating, inverted indexes: map(k,v) => f(k,v),(k,v)
  - Filtering, Parsing, Validation
  - Sorting,
  - Graph Processing (???) – page rank,
  - Cross-correlation (???)
  - Machine Learning (???)
Time Travel

- It’s not just storing it, it’s what you do with the data

AMPLab Unification Philosophy

Don’t specialize MapReduce – Generalize it!

Two additions to Hadoop MR can enable all the models shown earlier!

1. General Task DAGs
2. Data Sharing

For Users:
- Fewer Systems to Use
- Less Data Movement
Berkeley Data Analytics Stack

Cancer Genomics, Energy Debugging, Smart Buildings

BlinkDB
Sample Clean
MLBase
SparkR

Spark Streaming
SparkSQL
GraphX
MLlib

Apache Spark
Velox Model Serving

Tachyon
HDFS, S3,

Apache Mesos
Yarn

In-house Apps
Access and Interfaces
Processing Engine
Storage
Resource Virtualization

12/1/14
The Data Deluge

- Billions of users connected through the net
  - WWW, Facebook, twitter, cell phones, …
  - 80% of the data on FB was produced last year
- Clock Rates stalled
- Storage getting cheaper
  - Store more data!

![Graph showing data growth over time](image)
Data Grows Faster than Moore’s Law

Projected Growth

- Moore's Law
- Particle Accel.
- DNA Sequencers

Increase over 2010

2010 2011 2012 2013 2014 2015
Complex Questions

• Hard questions
  – What is the impact on traffic and home prices of building a new ramp?

• Detect real-time events
  – Is there a cyber attack going on?

• Open-ended questions
  – How many supernovae happened last year?
MapReduce Pros

• Distribution is completely transparent
  – Not a single line of distributed programming (ease, correctness)

• Automatic fault-tolerance
  – Determinism enables running failed tasks somewhere else again
  – Saved intermediate data enables just re-running failed reducers

• Automatic scaling
  – As operations as side-effect free, they can be distributed to any number of machines dynamically

• Automatic load-balancing
  – Move tasks and speculatively execute duplicate copies of slow tasks (stragglers)
HDFS – distributed file system

- Blocks are distributed, with replicas, across nodes
- Name-node provides the index structure
- Client locates blocks via RPC to metadata
- Data nodes inform Namenode of failures through heartbeats
- Block locations made visible to MapReduce Framework
MapReduce

• both a *programming model* and a *clustered computing system*

  – A specific pattern of computing over large distributed data sets
  
  – A system which takes a MapReduce-formulated problem and executes it on a large cluster
  
  – Hides implementation details, such as hardware failures, grouping and sorting, scheduling …
Word-Count using MapReduce

Problem: determine the frequency of each word in a large document collection

\[
(d_k, \{'w_1 \ldots w_n'\}) \rightarrow [(w_i, c_i)]
\]

\[
(w_i, [c_i]) \rightarrow (w_i, \sum_i c_i)
\]
General MapReduce Formulation

Map:
– Preprocesses a set of files to generate intermediate key-value pairs, in parallel

\[
\text{Map: } (k_1, v_1) \rightarrow [(k_2, v_2)]
\]

Group:
– Partitions intermediate key-value pairs by unique key, generating a list of all associated values
  » Shuffle so each key list is all on a node

Reduce:
– For each key, iterate over value list
– Performs computation that requires context between iterations
– Parallelizable amongst different keys, but not within one key

\[
\text{Reduce: } (k_2, [v_2]) \rightarrow (k_2, f([v_2]))
\]
MapReduce Logical Execution

Input

Intermediate

Group by Key

Grouped

Output
MapReduce Parallel Execution

Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
MapReduce Parallelization: Pipelining

- Fine grain tasks: many more map tasks than machines
  - Better dynamic load balancing
  - Minimizes time for fault recovery
  - Can pipeline the shuffling/grouping while maps are still running

- Example: 2000 machines -> 200,000 map + 5000 reduce tasks

Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
MR Runtime Execution Example

• The following slides illustrate an example run of MapReduce on a Google cluster

• A sample job from the indexing pipeline, processes ~900 GB of crawled pages
Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>1857</td>
<td>1707</td>
<td>878934.6</td>
<td>191995.8</td>
<td>113936.6</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>113936.6</td>
<td>57113.7</td>
<td>57113.7</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>57113.7</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>699.1</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>349.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5004411944</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17290135</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17331371</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17290135</td>
</tr>
</tbody>
</table>

Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>195</td>
<td>305</td>
<td>523499.2</td>
<td>523389.6</td>
<td>523389.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>195</td>
<td>523389.6</td>
<td>2685.2</td>
<td>2742.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.3</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>45.7</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>2313178</td>
</tr>
<tr>
<td>doc-indexed</td>
<td>7936</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>1954105</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>1954105</td>
</tr>
</tbody>
</table>

Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>133837.8</td>
<td>136929.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.1</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1238.8</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0.1</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0.0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0.0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51738599</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51738599</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>263283.3</td>
<td>269351.2</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1225.1</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0.10</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dupsin-indexmerge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51842100</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51842100</td>
</tr>
</tbody>
</table>

Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
Shamelessly stolen from Jeff Dean’s OSDI ‘04 presentation
Fault Tolerance vis re-execution

• **On Worker Failure:**
  – Detect via periodic heartbeats
  – Re-execute completed and in-progress *map* tasks
  – Re-execute in progress *reduce* tasks
  – Task completion committed through master

• **Master ???**
Admin

• Project 3
• Reviews during R&R
• Mid 3 in Final Exam Group 1 (12/15 8-11)
  – 10 Evans (currently)
MapReduce Cons

• **Restricted programming model**
  – Not always natural to express problems in this model
  – Low-level coding necessary
  – Little support for iterative jobs (lots of disk access)
  – High-latency (batch processing)

• **Addressed by follow-up research and Apache projects**
  – Pig and Hive for high-level coding
  – Spark for iterative and low-latency jobs
Big Data Ecosystem Evolution

MapReduce

General batch processing

Specialized systems
(iterative, interactive and streaming apps)

Pregel
Giraph
Dremel
Drill
Tez
Impala
GraphLab
Storm
S4
...
Don’t specialize MapReduce – Generalize it!
Two additions to Hadoop MR can enable all the models shown earlier!
1. General Task DAGs
2. Data Sharing
For Users:
- Fewer Systems to Use
- Less Data Movement
Complex jobs, interactive queries and online processing all need one thing that MR lacks:

Efficient primitives for data sharing

Iterative job

Interactive mining

Stream processing
Spark Motivation

Complex jobs, interactive queries and online processing all need one thing that MR lacks:

Efficient primitives for data sharing

Problem: in MR, the only way to share data across jobs is using stable storage (e.g. file system) \( \rightarrow \) slow!
Examples

Opportunity: DRAM is getting cheaper $\rightarrow$ use main memory for intermediate results instead of disks
“It’s only September but it’s already clear that 2014 will be the year of Apache Spark”

-- Datanami, 9/15/14

Iteration in Map-Reduce

Initial Model $w^{(0)}$

Training Data

Map

Reduce

Learned Model

$w^{(1)}$

$w^{(2)}$

$w^{(3)}$
Cost of Iteration in Map-Reduce

Initial Model

Training Data

Map

Reduce

Learned Model

$w^{(0)}$

$w^{(1)}$

$w^{(2)}$

$w^{(3)}$

Repeatedly load same data

Read 1

Read 2

Read 3

Initial Model

Training Data

Map

Reduce

Learned Model

$w^{(0)}$

$w^{(1)}$

$w^{(2)}$

$w^{(3)}$

Repeatedly load same data

Initial Model

Training Data

Map

Reduce

Learned Model

$w^{(0)}$

$w^{(1)}$

$w^{(2)}$

$w^{(3)}$

Repeatedly load same data
Cost of Iteration in Map-Reduce

- Initial Model: $w^{(0)}$
- Map
- Reduce
- Learned Model:
  - $w^{(1)}$
  - $w^{(2)}$
  - $w^{(3)}$

*Redundantly* save output between stages.
Memory Opt. Dataflow

Cached Load

Training Data (HDFS)

Map → Reduce

Map → Reduce

Map → Reduce
Memory Opt. Dataflow View

Training Data (HDFS)

Map → Reduce

Map → Reduce

Map → Reduce

Efficiently move data between stages

Spark: 10-100× faster than Hadoop MapReduce
Resilient Distributed Datasets (RDDs)

- **API:** coarse-grained *transformations* (map, group-by, join, sort, filter, sample, …) on immutable collections

- Resilient Distributed Datasets (RDDs)
  - Collections of objects that can be stored in memory or disk across a cluster
  - Built via parallel transformations (map, filter, …)
  - Automatically rebuilt on failure

- **Rich enough to capture many models:**
  - Data flow models: MapReduce, Dryad, SQL, …
  - Specialized models: Pregel, Hama, …

---

Fault Tolerance with RDDs

RDDs track the series of transformations used to build them (their *lineage*)

- Log one operation to apply to many elements
- No cost if nothing fails

Enables per-node recomputation of lost data

```
messages = textFile(...).filter(_.contains("error"))
  .map(_.split(\t')(2))
```
Systems Research as Time Travel ...

Cancer Genomics, Energy Debugging, Smart Buildings

BlinkDB
Sample Clean
MLBase
SparkR
Spark Streaming
SparkSQL
GraphX
MLlib

Apache Spark
Velox Model Serving

Tachyon
HDFS, S3,

Apache Mesos
Yarn

In-house Apps
Access and Interfaces
Processing Engine
Storage
Resource Virtualization

Apache Mesos
UCB CS162 Fa14 L39
12/1/14