Introduction to Hadoop

ID2210
Jim Dowling
Large Scale Distributed Computing

- **In #Nodes**
  - BitTorrent (millions)
  - Peer-to-Peer

- **In #Instructions/sec**
  - Teraflops, Petaflops, Exascale
  - Super-Computing

- **In #Bytes stored**
  - Facebook: 300+ Petabytes (April 2014)*
  - Hadoop

- **In #Bytes processed/time**
  - Google processed 24 petabytes of data per day in 2013
  - Colossus, Spanner, BigQuery, BigTable, Borg, Omega, ..

*http://www.adweek.com/socialtimes/orcfile/434041
Where does Big Data Come From?

- On-line services: PBs per day
- Scientific instruments: PBs per minute
- Whole genome sequencing: 250 GB per person
- Internet-of-Things: Will be lots!
What is Big Data?

Small Data

Big Data

Congratulations, it only took you 65,298 seconds

Found It!!!
Why is Big Data “hot”?

- Companies like Google and Facebook have shown how to extract value from Big Data

Orbitz looks for higher prices from Safari users [WSJ’12]
Why is Big Data “hot”?  

- Big Data helped Obama win the 2012 election through **data-driven decision making***

  Data said: middle-aged females like contests, dinners and celebrity

Why is Big Data Important in Science?

- In a wide array of academic fields, the ability to effectively process data is superseding other more classical modes of research.

  “More data trumps better algorithms”*

*“The Unreasonable Effectiveness of Data” [Halevey et al 09]
4 Vs of Big Data

- Volume
- Velocity
- Variety
- Veracity/Variability/Value
A quick historical tour of data systems
In the Beginning
Batch Sequential Processing

IBM 082 Punch Card Sorter

Scan → Sort

No Fault Tolerance 😊
1960s
First Database Management Systems

**COBOL**

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PAYROLL.
000300 AUTHOR. JOHN DOE.
000400 DATE. APRIL 5TH 1960.
000500 REMARKS.
000600 INPUT FROM RUN 4 AND OUTPUT TO RUN 25.
000700 THIS PROGRAM PROCESSES SALARIED
000800 EMPLOYEES ONLY.

000900 ENVIRONMENT DIVISION.
001000 CONFIGURATION SECTION.
001100 SOURCE COMPUTER. COMPUTER NAME.
001200 OBJECT COMPUTER. COMPUTER NAME.
001300 SPECIAL NAMES. HARDWARE NAME.

001400 INPUT-OUTPUT SECTION.
001500 FILE CONTROL. SELECT FILE-NAME 1
001600 SELECT FILE-NAME 2 SELECT ........
001700 1-O CONTROL. APPLY ****

001800 DATA DIVISION.
001900 RD MASTER-PAYROLL, LABEL RECORDS ARE
002000 STANDARD, DATA RECORDS ARE MASTER-
002100 PAY, SEQUENCED ON BADGE-NUMBER.
002200 01 MASTER-PAY SIZE IS 180 CHAR-
002300 ACTERS, CLASS IS ALPHAMERIC.
002400 02 BADGE-NUMBER SIZE IS 12
002500 CHARACTERS, PICTURE IS
002600 AAAAAA9999999.
```

**DBMS**
Hierarchical and Network Database Management Systems
You had to know what data you want, and how to find it
Early DBMS’ were Disk-Aware
Codd's Relational Model

Just tell me the data you want, the system will find it.
CREATE TABLE Students (id INT PRIMARY_KEY, firstname VARCHAR(96), lastname VARCHAR(96));

SELECT * FROM Students WHERE id > 10;
Each color represents a program in this plan diagram.

- Each program produces the same result for the Query.
- Each program has different performance characteristics depending on changes in the data characteristics.
What if I have lots of Concurrent Queries?

- Data Integrity using Transactions*

ACID

Atomicity Consistency Isolation Durability

*Jim Gray, ”The Transaction Concept: Virtues and Limitation”
In the 1990s
Data Read Rates Increased Dramatically
Distribute within a Data Center

Master-Slave Replication

Data-location awareness is back:
Clients read from slaves, write to master.
Possibility of reading stale data.
In the 2000s
Data Write Rates Increased Dramatically
Unstructured Data explodes

Source: IDC whitepaper. As the Economy contracts, the Digital Universe Explodes. 2009
Key-Value stores don’t do Big Data yet. Existing Big Data systems currently only work for a single Data Centre.*

*The usual Google Exception applies
Storage and Processing of Big Data
What is Apache Hadoop?

- Huge data sets and large files
  - Gigabytes files, petabyte data sets
  - Scales to thousands of nodes on commodity hardware

- No Schema Required
  - Data can be just copied in, extract required columns later

- Fault tolerant

- Network topology-aware, Data Location-Aware

- Optimized for analytics: high-throughput file access
Hadoop (version 1)

Application

MapReduce

Hadoop Filesystem
write "/crawler/bot/jd.io/1"

HDFS: Hadoop Filesystem

Under-replicated blocks

Heartbeats

Rebalance
Re-replicate blocks
HDFS v2 Architecture

Active-Standby Replication of NN Log
Agreement on the Active NameNode
Faster Recovery - Cut the NN Log

Journal Nodes

Zookeeper Nodes

NameNode

Standby NameNode

Snapshot Node

DataNodes

HDFS Client
Processing Big Data
Big Data Processing with No Data Locality

Job("/crawler/bot/jd.io/1")

Workflow Manager

Compute Grid Node

This doesn’t scale. Bandwidth is the bottleneck

submit
MapReduce – Data Locality

Job("/crawler/bot/jd.io/1")

submit

Job Tracker

Task Tracker

Job

Task Tracker

Job

Task Tracker

Job

Task Tracker

Job

Task Tracker

Job

Task Tracker

Job

R = resultFile(s)
1. Programming Paradigm

2. Processing Pipeline (moving computation to data)

*Dean et al, OSDI’04
MapReduce Programming Paradigm

\[
\text{map}(\text{record}) \rightarrow \{(\text{key}_i, \text{value}_i), \ldots, (\text{key}_l, \text{value}_l)\}
\]

\[
\text{reduce}((\text{key}_i, \{\text{value}_k, \ldots, \text{value}_y\}) \rightarrow \text{output}
\]
MapReduce Programming Paradigm

• Also found in:
  
  Functional programming languages

  MongoDB

  Cassandra
Example: Building a Web Search Index

\[
\text{map}(\text{url}, \text{doc}) \rightarrow \\
\{(\text{term}_i, \text{url}), (\text{term}_m, \text{url})\}
\]

\[
\text{reduce}((\text{term}, \{\text{url}_k, \ldots, \text{url}_y\}) \rightarrow \\
(\text{term}, (\text{posting list of } \text{url}, \text{count}))
\]
map( ("jd.io", "A hipster website with news"))

->

{
    emit("a", "jd.io"),
    emit("hipster", "jd.io"),
    emit("website", "jd.io"),
    emit("with", "jd.io"),
    emit("news", "jd.io")
}
Example: Building a Web Search Index

```python
map( ("hn.io", "Hacker hipster news"))
->
{
    emit("hacker", "hn.io"),
    emit("hipster", "hn.io"),
    emit("news", "hn.io")
}
```
Example: Building a Web Search Index

reduce( "hipster", { "jd.io", "hn.io" } ) ->
( "hipster", (["jd.io", "hn.io"], 2))
Example: Building a Web Search Index

```
reduce("website", {"jd.io"}) ->
  ("website", (["jd.io"], 1))
```
Example: Building a Web Search Index

\[
\text{reduce}( \text{"news"}, \{ \text{"jd.io"}, \text{"hn.io"} \}) \rightarrow \\
( \text{"news"}, ([\text{"jd.io"}, \text{"hn.io"}], 2))
\]
MapReduce

\[
\text{map}(\text{url, doc}) \rightarrow \{(\text{term}_1, \text{url}), (\text{term}_1, \text{url})\}
\]
Shuffle Phase

MapReduce

\texttt{group by term}

Shuffle over the Network using a Partitioner
Reduce Phase

**MapReduce**

\[
\text{reduce}((\text{term,}\{\text{url}_k,\text{url}_y}\}) \rightarrow \\
(\text{term, (posting list of url, count)})
\]
Hadoop 2.x

Single Processing Framework
Batch Apps

Hadoop 1.x

MapReduce
(resource mgmt, job scheduler, data processing)

HDFS
(distributed storage)

Multiple Processing Frameworks
Batch, Interactive, Streaming ...

Hadoop 2.x

MapReduce
(data processing)

Others
(spark, mpi, giraph, etc)

YARN
(resource mgmt, job scheduler)

HDFS
(distributed storage)
MapReduce and MPI as YARN Applications

[Murthy et. al, Apache Hadoop YARN: Yet Another Resource Negotiator”, SOCC’13]
Limitations of MapReduce [Zaharia’11]

- MapReduce is based on an *acyclic data flow* from stable storage to stable storage.
  - Slow writes data to HDFS at every stage in the pipeline
- Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:
  - **Iterative** algorithms (machine learning, graphs)
  - **Interactive** data mining tools (R, Excel, Python)
```scala
val input = TextFile(textInput)

val words = input
  .flatMap{
    line => line.split(" ")
  }

val counts = words
  .groupBy{
    word => word
  }
  .count()

val output = counts
  .write(wordsOutput,
    RecordDataSinkFormat() )

val plan = new ScalaPlan(Seq(output))
```
Spark – Resilient Distributed Datasets

- Allow apps to keep working sets in memory for efficient reuse
- Retain the attractive properties of MapReduce
  - Fault tolerance, data locality, scalability

Resilient distributed datasets (RDDs)
- Immutable, partitioned collections of objects
- Created through parallel *transformations* (map, filter, groupBy, join, ...) on data in stable storage
- Can be *cached* for efficient reuse

*Actions* on RDDs
- Count, reduce, collect, save, ...
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
```

**Result:** scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)
Apache Flink – DataFlow Operators

<table>
<thead>
<tr>
<th>Map</th>
<th>Iterate</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduce</td>
<td>Delta Iterate</td>
<td>Aggregate</td>
</tr>
<tr>
<td>Join</td>
<td>Filter</td>
<td>Distinct</td>
</tr>
<tr>
<td>CoGroup</td>
<td>FlatMap</td>
<td>Vertex Update</td>
</tr>
<tr>
<td>Union</td>
<td>GroupReduce</td>
<td>Accumulators</td>
</tr>
</tbody>
</table>

Built-in vs. driver-based looping

- Loop outside the system, in driver program
- Iterative program looks like many independent jobs
- Dataflows with feedback edges
- System is iteration-aware, can optimize the job
Cloud Computing traditionally separates storage and computation.

- **OpenStack**
  - Nova (Compute)
  - Glance (VM Images)
  - Swift (Object Storage)

- **Amazon Web Services**
  - EC2
  - Elastic Block Storage
  - S3
Data Locality for Hadoop on the Cloud

- Cloud hardware configurations should support data locality

- Hadoop’s original topology awareness breaks
  - Placement of >1 VM containing block replicas for the same file on the same physical host increases correlated failures

- VMWare introduced a NodeGroup aware topology
  - HADOOP-8468
Conclusions

• Hadoop is the open-source enabling technology for Big Data

• YARN is rapidly becoming the operating system for the Data Center

• Apache Spark and Flink are in-memory processing frameworks for Hadoop
References

• Dean et. Al, “MapReduce: Simplified Data Processing on Large Clusters”, OSDI’04.


• “Processing a Trillion Cells per Mouse Click”, VLDB’12