Introduction to Apache Hadoop & Pig

Milind Bhandarkar
(milindb@yahoo-inc.com)
Y!IM: gridsolutions
Agenda

- Apache Hadoop
- Map-Reduce
- Distributed File System
- Writing Scalable Applications
- Apache Pig
- Q & A
Hadoop: Behind Every Click At Yahoo!
Hadoop At Yahoo! (Some Statistics)

- 40,000 + machines in 20+ clusters
- Largest cluster is 4,000 machines
- 6 Petabytes of data (compressed, unreplicated)
- 1000+ users
- 200,000+ jobs/day
Sample Applications

- Data analysis is the inner loop at Yahoo!
- Data $\Rightarrow$ Information $\Rightarrow$ Value
- Log processing: Analytics, reporting, buzz
- Search index
- Content Optimization, Spam filters
- Computational Advertising
BEHIND EVERY CLICK
Who Uses Hadoop?
Why Clusters?
<table>
<thead>
<tr>
<th>Dataset Description</th>
<th>Size (PB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Physics Large Hadron Collider</td>
<td>15PB</td>
</tr>
<tr>
<td>Annual Email Traffic, no spam</td>
<td>300PB+</td>
</tr>
<tr>
<td>Internet Archive</td>
<td>1PB+</td>
</tr>
<tr>
<td>2004 Walmart Transaction DB</td>
<td>500TB</td>
</tr>
<tr>
<td>Typical Oil Company</td>
<td>350TB+</td>
</tr>
<tr>
<td>200 of London’s Traffic Cams</td>
<td>8TB/day</td>
</tr>
<tr>
<td>Merck Blo Research DB</td>
<td>1.5TB/qtr</td>
</tr>
<tr>
<td>UPMC Hospitals Imaging Data</td>
<td>500TB/yr</td>
</tr>
<tr>
<td>MIT Babytalk Speech Experiment</td>
<td>1.4PB</td>
</tr>
<tr>
<td>Terashake Earthquake Model of LA Basin</td>
<td>1PB</td>
</tr>
<tr>
<td>One Day of Instant Messaging in 2002</td>
<td>750GB</td>
</tr>
</tbody>
</table>

Total digital data to be created this year **270,000PB** (IDC)

---

**Big Datasets**

Cost Per Gigabyte
(http://www.mkomo.com/cost-per-gigabyte)
Storage Trends
(Graph by Adam Leventhal, ACM Queue, Dec 2009)
Yahoo! Search Assist
Search Assist

• Insight: Related concepts appear close together in text corpus

• Input: Web pages
  • 1 Billion Pages, 10K bytes each
  • 10 TB of input data

• Output: List(word, List(related words))
Search Assist

// Input: List(URL, Text)
foreach URL in Input :
    Words = Tokenize(Text(URL));
    foreach word in Tokens :
        Insert (word, Next(word, Tokens)) in Pairs;
        Insert (word, Previous(word, Tokens)) in Pairs;
// Result: Pairs = List (word, RelatedWord)
Group Pairs by word;
// Result: List (word, List(RelatedWords)
foreach word in Pairs :
    Count RelatedWords in GroupedPairs;
// Result: List (word, List(RelatedWords, count))
foreach word in CountedPairs :
    Sort Pairs(word, *) descending by count;
    choose Top 5 Pairs;
// Result: List (word, Top5(RelatedWords))
You Might Also Know

Yong Gao
Senior Engineering Manager at Yahoo Inc.

Priyank Garg
Director Product Management, Yahoo! Search; Jt Managing Director at Advance Valves

Andrew Yu
Yahoo!, Co-creator of PostgreSQL
You Might Also Know

• Insight: You might also know Joe Smith if a lot of folks you know, know Joe Smith
  • if you don’t know Joe Smith already

• Numbers:
  • 300 MM users
  • Average connections per user is 100
// Input: List(UserName, List(Connections))

foreach u in UserList : // 300 MM
    foreach x in Connections(u) : // 100
        foreach y in Connections(x) : // 100
            if (y not in Connections(u)) :
                Count(u, y)++; // 3 Trillion Iterations

Sort (u, y) in descending order of Count(u, y);
Choose Top 3 y;
Store (u, {y0, y1, y2}) for serving;
Performance

- 101 Random accesses for each user
  - Assume 1 ms per random access
  - 100 ms per user
- 300 MM users
  - 300 days on a single machine
MapReduce Paradigm
Map & Reduce

• Primitives in Lisp (& Other functional languages) 1970s

• Google Paper 2004

• http://labs.google.com/papers/mapreduce.html
Output_List = Map (Input_List)

Square (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) =
(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
Output Element = Reduce (Input List)

Sum (1, 4, 9, 16, 25, 36, 49, 64, 81, 100) = 385
Parallelism

- Map is inherently parallel
- Each list element processed independently
- Reduce is inherently sequential
- Unless processing multiple lists
- Grouping to produce multiple lists
// Input:  http://hadoop.apache.org

Pairs = Tokenize_And_Pair ( Text ( Input ) )

Output = {
    (apache, hadoop) (hadoop, mapreduce) (hadoop, streaming) 
    (hadoop, pig) (apache, pig) (hadoop, DFS) (streaming, 
    commandline) (hadoop, java) (DFS, namenode) (datanode, 
    block) (replication, default)... 
}
Search Assist Reduce

// Input: GroupedList (word, GroupedList(words))

CountedPairs = CountOccurrences (word, RelatedWords)

Output = {
  (hadoop, apache, 7) (hadoop, DFS, 3) (hadoop, streaming, 4) (hadoop, mapreduce, 9) ...
}

Wednesday, 22 September 2010
Issues with Large Data

• Map Parallelism: Splitting input data
  • Shipping input data

• Reduce Parallelism:
  • Grouping related data

• Dealing with failures
  • Load imbalance
Wednesday, 22 September 2010
Apache Hadoop

- January 2006: Subproject of Lucene
- January 2008: Top-level Apache project
- Stable Version: 0.20 S (Strong Authentication)
- Latest Version: 0.21
- Major contributors: Yahoo!, Facebook, Powerset, Cloudera
Apache Hadoop

• Reliable, Performant Distributed file system
• MapReduce Programming framework
• Related Projects: HBase, Hive, Pig, Howl, Oozie, Zookeeper, Chukwa, Mahout, Cascading, Scribe, Cassandra, Hypertable, Voldemort, Azkaban, Sqoop, Flume, Avro ...
Problem: Bandwidth to Data

• Scan 100TB Datasets on 1000 node cluster
  • Remote storage @ 10MB/s = 165 mins
  • Local storage @ 50-200MB/s = 33-8 mins
• Moving computation is more efficient than moving data
• Need visibility into data placement
Problem: Scaling Reliably

- Failure is not an option, it’s a rule!
- 1000 nodes, MTBF < 1 day
- 4000 disks, 8000 cores, 25 switches, 1000 NICs, 2000 DIMMS (16TB RAM)
- Need fault tolerant store with reasonable availability guarantees
- Handle hardware faults transparently
Hadoop Goals

- Scalable: Petabytes ($10^{15}$ Bytes) of data on thousands of nodes
- Economical: Commodity components only
- Reliable
  - Engineering reliability into every application is expensive
Hadoop MapReduce
Think MR

- Record = (Key, Value)
- Key : Comparable, Serializable
- Value: Serializable
- Input, Map, Shuffle, Reduce, Output
cat /var/log/auth.log* | \
grep “session opened” | cut -d’ ‘ -f10 | \nsort | \nuniq -c > \n~/userlist
Map

- Input: \((\text{Key}_1, \text{Value}_1)\)
- Output: List(\(\text{Key}_2, \text{Value}_2\))
- Projections, Filtering, Transformation
Shuffle

- Input: List(Key₂, Value₂)
- Output
  - Sort(Partition(List(Key₂, List(Value₂))))
- Provided by Hadoop
Reduce

- Input: List(Key₂, List(Value₂))
- Output: List(Key₃, Value₃)
- Aggregation
Example: Unigrams

• Input: Huge text corpus
  • Wikipedia Articles (40GB uncompressed)
  • Output: List of words sorted in descending order of frequency
Unigrams

$ cat ~/wikipedia.txt | \n  sed -e 's/ \n/g' | grep . | \n  sort | \n  uniq -c > \n  ~/frequencies.txt

$ cat ~/frequencies.txt | \n  # cat | \n  sort -n -k1,1 -r | \n  # cat > \n  ~/unigrams.txt
mapper (filename, file-contents):
    for each word in file-contents:
        emit (word, 1)

reducer (word, values):
    sum = 0
    for each value in values:
        sum = sum + value
    emit (word, sum)
MR for Unigrams

mapper (word, frequency):
    emit (frequency, word)

reducer (frequency, words):
    for each word in words:
        emit (word, frequency)
Hadoop Streaming

- Hadoop is written in Java
- Java MapReduce code is “native”
- What about Non-Java Programmers?
  - Perl, Python, Shell, R
  - grep, sed, awk, uniq as Mappers/Reducers
- Text Input and Output
Hadoop Streaming

- Thin Java wrapper for Map & Reduce Tasks
- Forks actual Mapper & Reducer
- IPC via stdin, stdout, stderr
- Key.toString() \t Value.toString() \n
- Slower than Java programs
- Allows for quick prototyping / debugging
Hadoop Streaming

$ bin/hadoop jar hadoop-streaming.jar \
   -input in-files -output out-dir \
   -mapper mapper.sh -reducer reducer.sh

# mapper.sh

sed -e 's/ /\n/g' | grep .

# reducer.sh

uniq -c | awk '{print $2 "\t" $1}''
Running Hadoop Jobs
Running a Job

[milindb@gateway ~]$ hadoop jar \
$HADOOP_HOME/hadoop-examples.jar wordcount \
data/newsarchive/20080923 /tmp/newsout
input.FileInputFormat: Total input paths to process : 4
mapred.JobClient: Running job: job_200904270516_5709
mapred.JobClient:  map 0% reduce 0%
mapred.JobClient:  map 3% reduce 0%
mapred.JobClient:  map 7% reduce 0%
....
mapred.JobClient:  map 100% reduce 21%
mapred.JobClient:  map 100% reduce 31%
mapred.JobClient:  map 100% reduce 33%
mapred.JobClient:  map 100% reduce 66%
mapred.JobClient:  map 100% reduce 100%
mapred.JobClient: Job complete: job_200904270516_5709
Running a Job

mapred.JobClient: Counters: 18
mapred.JobClient:   Job Counters
mapred.JobClient:     Launched reduce tasks=1
mapred.JobClient:     Rack-local map tasks=10
mapred.JobClient:     Launched map tasks=25
mapred.JobClient:     Data-local map tasks=1
mapred.JobClient:   FileSystemCounters
mapred.JobClient:     FILE_BYTES_READ=491145085
mapred.JobClient:     HDFS_BYTES_READ=3068106537
mapred.JobClient:     FILE_BYTES_WRITTEN=724733409
mapred.JobClient:     HDFS_BYTES_WRITTEN=377464307
Running a Job

mapred.JobClient: Map-Reduce Framework
mapred.JobClient: Combine output records=73828180
mapred.JobClient: Map input records=36079096
mapred.JobClient: Reduce shuffle bytes=233587524
mapred.JobClient: Spilled Records=78177976
mapred.JobClient: Map output bytes=4278663275
mapred.JobClient: Combine input records=371084796
mapred.JobClient: Map output records=313041519
mapred.JobClient: Reduce input records=15784903
kry-jt1 Hadoop Map/Reduce Administration

State: RUNNING
Started: Mon Apr 27 05:16:52 UTC 2009
Version: 0.20.0-2701599, r
Compiled: Tue Apr 21 18:09:33 UTC 2009 by hadoopqa
Identifier: 200904270516
Cluster Summary (Heap Size is 16.46 GB/23.14 GB)

<table>
<thead>
<tr>
<th>Maps</th>
<th>Reduces</th>
<th>Total Submissions</th>
<th>Nodes</th>
<th>Map Task Capacity</th>
<th>Reduce Task Capacity</th>
<th>Avg. Tasks/Node</th>
<th>Blacklisted Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>21</td>
<td>5568</td>
<td>1497</td>
<td>2994</td>
<td>2994</td>
<td>4.00</td>
<td>3</td>
</tr>
</tbody>
</table>
### Running Jobs

<table>
<thead>
<tr>
<th>Jobid</th>
<th>Priority</th>
<th>User</th>
<th>Name</th>
<th>Map % Complete</th>
<th>Map Total</th>
<th>Maps Completed</th>
<th>Reduce % Complete</th>
<th>Reduce Total</th>
<th>Reduces Completed</th>
<th>Job Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_200904270516_5719</td>
<td>NORMAL</td>
<td>milindb</td>
<td>word count</td>
<td>0.00%</td>
<td>25</td>
<td>0</td>
<td>0.00%</td>
<td>1</td>
<td>0</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Completed Jobs

<table>
<thead>
<tr>
<th>Jobid</th>
<th>Priority</th>
<th>User</th>
<th>Name</th>
<th>Map % Complete</th>
<th>Map Total</th>
<th>Maps Completed</th>
<th>Reduce % Complete</th>
<th>Reduce Total</th>
<th>Reduces Completed</th>
<th>Job Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_200904270516_5709</td>
<td>NORMAL</td>
<td>milindb</td>
<td>word count</td>
<td>100.00%</td>
<td>25</td>
<td>25</td>
<td>100.00%</td>
<td>1</td>
<td>1</td>
<td>NA</td>
</tr>
</tbody>
</table>

Jobs Status
### Hadoop job_200904270516_5709 on kry-jt1

**User:** milindb  
**Job Name:** word count  
**Job File:** hdfs://kry-nn1.inktomisearch.com/mapredsystem/hadoop/mapredsystem/job_200904270516_5709/job.xml  
**Job Setup:** Successful  
**Status:** Succeeded  
**Started at:** Sun May 03 21:38:54 UTC 2009  
**Finished at:** Sun May 03 21:42:52 UTC 2009  
**Finished in:** 3mins, 57sec  
**Job Cleanup:** Successful

<table>
<thead>
<tr>
<th>Kind</th>
<th>% Complete</th>
<th>Num Tasks</th>
<th>Pending</th>
<th>Running</th>
<th>Complete</th>
<th>Killed</th>
<th>Failed/Killed Task Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>100.00%</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>reduce</td>
<td>100.00%</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>
## Job Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launched reduce tasks</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rack-local map tasks</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>FileStreamCounters</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE_BYTES_READ</td>
<td>342,134,961</td>
<td>149,010,124</td>
<td>491,145,085</td>
</tr>
<tr>
<td>HDFS_BYTES_READ</td>
<td>3,068,106,537</td>
<td>0</td>
<td>3,068,106,537</td>
</tr>
<tr>
<td>FILE_BYTES_WRITTEN</td>
<td>575,723,285</td>
<td>149,010,124</td>
<td>724,733,409</td>
</tr>
<tr>
<td>HDFS_BYTES_WRITTEN</td>
<td>0</td>
<td>377,464,307</td>
<td>377,464,307</td>
</tr>
<tr>
<td><strong>Map-Reduce Framework</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduce input groups</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Combine output records</td>
<td>62,393,073</td>
<td>11,435,107</td>
<td>73,828,180</td>
</tr>
<tr>
<td>Map input records</td>
<td>36,079,096</td>
<td>0</td>
<td>36,079,096</td>
</tr>
<tr>
<td>Reduce shuffle bytes</td>
<td>0</td>
<td>233,587,524</td>
<td>233,587,524</td>
</tr>
<tr>
<td>Reduce output records</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Spilled Records</td>
<td>62,393,073</td>
<td>15,784,903</td>
<td>78,177,976</td>
</tr>
<tr>
<td>Map output bytes</td>
<td>4,278,663,275</td>
<td>0</td>
<td>4,278,663,275</td>
</tr>
<tr>
<td>Map output records</td>
<td>313,041,519</td>
<td>0</td>
<td>313,041,519</td>
</tr>
<tr>
<td>Combine input records</td>
<td>350,530,796</td>
<td>20,554,000</td>
<td>371,084,796</td>
</tr>
<tr>
<td>Reduce input records</td>
<td>0</td>
<td>15,784,903</td>
<td>15,784,903</td>
</tr>
</tbody>
</table>

Wednesday, 22 September 2010
Job Progress
<table>
<thead>
<tr>
<th>Task</th>
<th>Complete</th>
<th>Status</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Errors</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_200904270516_5709_m_000000</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:41:06 (1mins, 45sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000001</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:52 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000002</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000003</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000004</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000005</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:20</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000006</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000007</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:20</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000008</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:41:07 (1mins, 46sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000009</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200904270516_5709_m_000010</td>
<td>100.00%</td>
<td>Complete</td>
<td>3-May-2009 21:39:21</td>
<td>3-May-2009 21:40:51 (1mins, 30sec)</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**All Tasks**

Wednesday, 22 September 2010
### All Task Attempts

<table>
<thead>
<tr>
<th>Task Attempts</th>
<th>Machine</th>
<th>Status</th>
<th>Progress</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Errors</th>
<th>Task Logs</th>
<th>Counters</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>attempt_200904270516_5709_m_000000_0</td>
<td>/72.30.127.192/kry1540.inktomisearch.com</td>
<td>SUCCEEDED</td>
<td>100.00%</td>
<td>3-May-2009 21:39:20</td>
<td>3-May-2009 21:40:54 (1mins, 34sec)</td>
<td></td>
<td>Last 4KB All</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Input Split Locations

- /72.30.127.192/kry1540.inktomisearch.com
- /72.30.62.192/kry50414.inktomisearch.com
- /72.30.62.192/kry50413.inktomisearch.com

---

**Task Details**
# Task Counters

## Counters for attempt_200904270516_5709_m_000000_0

### FileSystemCounters
- FILE_BYTES_READ: 15,201,458
- HDFS_BYTES_READ: 143,611,125
- FILE_BYTES_WRITTEN: 25,452,741

### Map-Reduce Framework
- Combine output records: 2,755,686
- Map input records: 1,717,449
- Spilled Records: 2,755,686
- Map output bytes: 200,041,337
- Combine input records: 16,226,718
- Map output records: 14,560,207
Task Logs: 'attempt_200904270516_5709_m_000000_0'

**stdout logs**

**stderr logs**

**syslog logs**

ed-local/taskTracker/jobcache/job_200904270516_5709/attempt_200904270516_5709_m_000000_0/job.xml:a attempt to override final parameter

**Task Logs**

Wednesday, 22 September 2010
Hadoop Distributed File System
HDFS

• Data is organized into files and directories
• Files are divided into uniform sized blocks (default 64MB) and distributed across cluster nodes
• HDFS exposes block placement so that computation can be migrated to data
HDFS

- Blocks are replicated (default 3) to handle hardware failure
- Replication for performance and fault tolerance (Rack-Aware placement)
- HDFS keeps checksums of data for corruption detection and recovery
HDFS

• Master-Worker Architecture
• Single NameNode
• Many (Thousands) DataNodes
HDFS Master (NameNode)

• Manages filesystem namespace
• File metadata (i.e. “inode”)
• Mapping inode to list of blocks + locations
• Authorization & Authentication
• Checkpoint & journal namespace changes
Namenode

- Mapping of datanode to list of blocks
- Monitor datanode health
- Replicate missing blocks
- Keeps ALL namespace in memory
- 60M objects (File/Block) in 16GB
Datanodes

- Handle block storage on multiple volumes & block integrity
- Clients access the blocks directly from data nodes
- Periodically send heartbeats and block reports to Namenode
- Blocks are stored as underlying OS’s files
HDFS Architecture
Replication

• A file’s replication factor can be changed dynamically (default 3)
• Block placement is rack aware
• Block under-replication & over-replication is detected by Namenode
• Balancer application rebalances blocks to balance datanode utilization
hadoop fs [-fs <local | file system URI>] [-conf <configuration file>]
[-D <property=value>] [-ls <path>] [-lsr <path>] [-du <path>]
[-dus <path>] [-mv <src> <dst>] [-cp <src> <dst>] [-rm <src>]
[-rmdir <src>] [-put <localsrc> ... <dst>] [-copyToLocal <localsrc> ... <dst>]
[-moveFromLocal <localsrc> ... <dst>] [-get [-ignoreCrc] [-crc] <src> <localdst>
[-getmerge <src> <localdst> [addnl]] [-cat <src>]
[-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>] [-moveToLocal <src> <localdst>]
[-touchz <path>] [-test [-ezd] <path>] [-stat [format] <path>]
[-tail [-f] <path>] [-text <path>]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...] 
[-chown [-R] [OWNER][:[GROUP]] PATH...] 
[-chgrp [-R] GROUP PATH...] 
[-count[-q] <path>]
[-help [cmd]]
HDFS Java API

// Get default file system instance
fs = Filesystem.get(new Configuration());

// Or Get file system instance from URI
fs = Filesystem.get(URI.create(uri),
    new Configuration());

// Create, open, list, ...
OutputStream out = fs.create(path, ...);

InputStream in = fs.open(path, ...);

boolean isDone = fs.delete(path, recursive);

FileStatus[] fstat = fs.listStatus(path);
libHDFS

#include "hdfs.h"

hdfsFS fs = hdfsConnectNewInstance("default", 0);

hdfsFile writeFile = hdfsOpenFile(fs, "/tmp/test.txt", 
   O_WRONLY|O_CREAT, 0, 0, 0);

int num_written = hdfsWrite(fs, writeFile, 
   (void*)buffer, sizeof(buffer));

hdfsCloseFile(fs, writeFile);

hdfsFile readFile = hdfsOpenFile(fs, "/tmp/test.txt", 
   O_RDONLY, 0, 0, 0);

int num_read = hdfsRead(fs, readFile, (void*)buffer, 
   sizeof(buffer));

hdfsCloseFile(fs, readFile);

hdfsDisconnect(fs);
Hadoop Scalability
Scalability of Parallel Programs

- If one node processes $k$ MB/s, then $N$ nodes should process $(k*N)$ MB/s
- If some fixed amount of data is processed in $T$ minutes on one node, the $N$ nodes should process same data in $(T/N)$ minutes
- Linear Scalability
Latency

Goal: Minimize program execution time
Throughput

Goal: Maximize Data processed per unit time
Amdahl’s Law

• If computation $C$ is split into $N$ different parts, $C_1..C_N$

• If partial computation $C_i$ can be speeded up by a factor of $S_i$

• Then overall computation speedup is limited by $\sum C_i / \sum (C_i / S_i)$
Amdahl’s Law

• Suppose Job has 5 phases: $P_0$ is 10 seconds, $P_1$, $P_2$, $P_3$ are 200 seconds each, and $P_4$ is 10 seconds

• Sequential runtime = 620 seconds

• $P_1$, $P_2$, $P_3$ parallelized on 100 machines with speedup of 80 (Each executes in 2.5 seconds)

• After parallelization, runtime = 27.5 seconds

• Effective Speedup: 22.5
Efficiency

• Suppose, speedup on N processors is S, then Efficiency = S/N

• In previous example, Efficiency = 0.225

• Goal: Maximize efficiency, while meeting required SLA
Amdahl’s Law
Map Reduce Data Flow
MapReduce Dataflow
MapReduce
Job Submission
Initialization
Scheduling
Execution
Map Task

Every map task does a small sort
Reduce Task
Hadoop Performance Tuning
Example

• “Bob” wants to count records in event logs (several hundred GB)
• Used Identity Mapper (
/bin/cat) & Single counting reducer (
/bin/wc -l)
• What is he doing wrong?
• This happened, really!
MapReduce Performance

- Minimize Overheads
- Task Scheduling & Startup
- Network Connection Setup
- Reduce intermediate data size
- Map Outputs + Reduce Inputs
- Opportunity to load balance
Number of Maps

• Number of Input Splits, computed by `InputFormat`

• Default `FileInputFormat`:
  • Number of HDFS Blocks
    • Good locality
  • Does not cross file boundary
Changing Number of Maps

- `mapred.map.tasks`
- `mapred.min.split.size`
- Concatenate small files into big files
- Change `dfs.block.size`
- Implement `InputFormat`
- `CombineFileInputFormat`
Number of Reduces

- `mapred.reduce.tasks`
- Shuffle Overheads (M*R)
  - 1-2 GB of data per reduce task
- Consider time needed for re-execution
- Memory consumed per group/bag
Shuffle

- Often the most expensive component
- $M \times R$ Transfers over the network
- Sort map outputs (intermediate data)
- Merge reduce inputs
Typical Hadoop Network Architecture
Improving Shuffle

- Avoid shuffling/sorting if possible
- Minimize redundant transfers
- Compress intermediate data
Avoid Shuffle

- Set `mapred.reduce.tasks` to zero
- Known as map-only computations
- Filters, Projections, Transformations
- Number of output files = number of input splits = number of input blocks
- May overwhelm HDFS
Minimize Redundant Transfers

• Combiners

• Intermediate data compression
Combiners

• When Maps produce many repeated keys
• Combiner: Local aggregation after Map & before Reduce
• Side-effect free
• Same interface as Reducers, and often the same class
Compression

- Often yields huge performance gains
- Set `mapred.output.compress=true` to compress job output
- Set `mapred.compress.map.output=true` to compress map outputs
- Codecs: Java zlib (default), LZO, bzip2, native gzip
Load Imbalance

- Inherent in application
- Imbalance in input splits
- Imbalance in computations
- Imbalance in partitions
- Heterogenous hardware
- Degradation over time
Speculative Execution

- Runs multiple instances of slow tasks
- Instance that finishes first, succeeds
- `mapred.map.speculative.execution=true`
- `mapred.reduce.speculative.execution=true`
- Can dramatically bring in long tails on jobs
Best Practices - 1

• Use higher-level languages (e.g. Pig)
• Coalesce small files into larger ones & use bigger HDFS block size
• Tune buffer sizes for tasks to avoid spill to disk, consume one slot per task
• Use combiner if local aggregation reduces size
Best Practices - 2

- Use compression everywhere
- CPU-efficient for intermediate data
- Disk-efficient for output data
- Use Distributed Cache to distribute small side-files (Less than DFS block size)
- Minimize number of NameNode/JobTracker RPCs from tasks
Apache Pig
What is Pig?

- System for processing large semi-structured data sets using Hadoop MapReduce platform
- Pig Latin: High-level procedural language
- Pig Engine: Parser, Optimizer and distributed query execution
Pig vs SQL

- Pig is procedural (How)
- Nested relational data model
- Schema is optional
- Scan-centric analytic workloads
- Limited query optimization

- SQL is declarative
- Flat relational data model
- Schema is required
- OLTP + OLAP workloads
- Significant opportunity for query optimization
Pig vs Hadoop

• Increase programmer productivity
• Decrease duplication of effort
• Insulates against Hadoop complexity
• Version Upgrades
• JobConf configuration tuning
• Job Chains
Example

• Input: User profiles, Page visits

• Find the top 5 most visited pages by users aged 18-25
In Native Hadoop
Users = load ‘users’ as (name, age);
Filtered = filter Users by age >= 18 and age <= 25;
Pages = load ‘pages’ as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
       COUNT(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into ‘top5sites’;
Natural Fit
Comparison
Flexibility & Control

• Easy to plug-in user code
• Metadata is not mandatory
• Pig does not impose a data model on you
• Fine grained control
• Complex data types
Pig Data Types

- Tuple: Ordered set of fields
  - Field can be simple or complex type
  - Nested relational model
- Bag: Collection of tuples
  - Can contain duplicates
- Map: Set of (key, value) pairs
Simple data types

- int : 42
- long : 42L
- float : 3.1415f
- double : 2.7182818
- chararray : UTF-8 String
- bytearray : blob
Expressions

A = LOAD 'data.txt AS
(f1:int, f2:{t:(n1:int, n2:int)}, f3: map[])

A =
{
  (1, -- A.f1 or A.$0
  { (2, 3), (4, 6) }, -- A.f2 or A.$1
  [ 'yahoo'#'mail' ] -- A.f3 or A.$2
}

Wednesday, 22 September 2010
Counting Word Frequencies

• Input: Large text document

• Process:
  • Load the file
  • For each line, generate word tokens
  • Group by word
  • Count words in each group
myinput = load '/user/milindb/text.txt'
    USING TextLoader() as (myword:chararray);

(program program)
(pig pig)
(program pig)
(hadoop pig)
(latin latin)
(pig latin)
words = FOREACH myinput GENERATE FLATTEN(TOKENIZE(*));

(program) (program) (pig) (pig) (program) (pig) (hadoop) (pig) (latin) (latin) (pig) (latin)
grouped = GROUP words BY $0;

(pig, {(pig), (pig), (pig), (pig), (pig)})
(latin, {(latin), (latin), (latin)})
(hadoop, {(hadoop)})
(program, {(program), (program), (program)})
counts = FOREACH grouped GENERATE group, COUNT(words);

(pig, 5L)
(latin, 3L)
(hadoop, 1L)
(program, 3L)
store counts into ‘/user/milindb/output’
using PigStorage();

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pig</td>
<td>5</td>
</tr>
<tr>
<td>latin</td>
<td>3</td>
</tr>
<tr>
<td>hadoop</td>
<td>1</td>
</tr>
<tr>
<td>program</td>
<td>3</td>
</tr>
</tbody>
</table>
Example: Log Processing

```
-- use a custom loader
Logs = load 'apachelogfile' using
    CommonLogLoader() as (addr, logname,
    user, time, method, uri, p, bytes);
-- apply your own function
Cleaned = foreach Logs generate addr,
    canonicalize(url) as url;
Grouped = group Cleaned by url;
-- run the result through a binary
Analyzed = stream Grouped through
    'urlanalyzer.py';
store Analyzed into 'analyzedurls';
```
-- declare your types
Grades = load ‘studentgrades’ as
    (name: chararray, age: int,
     gpa: double);
Good = filter Grades by age > 18
    and gpa > 3.0;
-- ordering will be by type
Sorted = order Good by gpa;
store Sorted into ‘smartgrownups’;
Logs = load 'weblogs' as (url, userid);
Grouped = group Logs by url;
-- Code inside {} will be applied to each
-- value in turn.
DistinctCount = foreach Grouped {
    Userid = Logs.userid;
    DistinctUsers = distinct Userid;
    generate group, COUNT(DistinctUsers);
}
store DistinctCount into 'distinctcount';
Pig Architecture

Wednesday, 22 September 2010
Pig Latin Programs

Query Parser

Semantic Checking

Logical Optimizer

Logical to Physical Translator

Physical To M/R Translator

Map Reduce Launcher

Logical Plan

Logical Plan

Optimized Logical Plan

Physical Plan

MapReduce Plan

Create a job jar to be submitted to hadoop cluster

Pig Frontend
Logical Plan

• Directed Acyclic Graph
• Logical Operator as Node
• Data flow as edges
• Logical Operators
• One per Pig statement
• Type checking with Schema
## Pig Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Read data from the file system</td>
</tr>
<tr>
<td>Store</td>
<td>Write data to the file system</td>
</tr>
<tr>
<td>Dump</td>
<td>Write data to stdout</td>
</tr>
</tbody>
</table>
## Pig Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreach..Generate</td>
<td>Apply expression to each record and generate one or more records</td>
</tr>
<tr>
<td>Filter</td>
<td>Apply predicate to each record and remove records where false</td>
</tr>
<tr>
<td>Stream..through</td>
<td>Stream records through user-provided binary</td>
</tr>
</tbody>
</table>
# Pig Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group/CoGroup</td>
<td>Collect records with the same key from one or more inputs</td>
</tr>
<tr>
<td>Join</td>
<td>Join two or more inputs based on a key</td>
</tr>
<tr>
<td>Order..by</td>
<td>Sort records based on a key</td>
</tr>
</tbody>
</table>
Physical Plan

- Pig supports two back-ends
  - Local
  - Hadoop MapReduce
- 1:1 correspondence with most logical operators
- Except Distinct, Group, Cogroup, Join etc
MapReduce Plan

- Detect Map-Reduce boundaries
- Group, Cogroup, Order, Distinct
- Coalesce operators into Map and Reduce stages
- *Job.jar* is created and submitted to Hadoop *JobControl*
Lazy Execution

- Nothing really executes until you request output
- Store, Dump, Explain, Describe, Illustrate

Advantages
- In-memory pipelining
- Filter re-ordering across multiple commands
Parallelism

- Split-wise parallelism on Map-side operators
- By default, 1 reducer
- PARALLEL keyword
  - group, cogroup, cross, join, distinct, order
$ pig
grunt > A = load ‘students’ as (name, age, gpa);
grunt > B = filter A by gpa > ‘3.5’;
grunt > store B into ‘good_students’;
grunt > dump A;
(jessica thompson, 73, 1.63)
(victor zipper, 23, 2.43)
(rachel hernandez, 40, 3.60)
grunt > describe A;
A: (name, age, gpa )
Running Pig

• Batch mode
  • $ pig myscript.pig

• Local mode
  • $ pig –x local

• Java mode (embed pig statements in java)
  • Keep pig.jar in the class path
Pig for SQL Programmers
### SQL to Pig

<table>
<thead>
<tr>
<th>SQL</th>
<th>Pig</th>
</tr>
</thead>
<tbody>
<tr>
<td>...FROM MyTable...</td>
<td>A = LOAD ‘MyTable’ USING PigStorage(‘\t’) AS (col1:int, col2:int, col3:int);</td>
</tr>
<tr>
<td>SELECT col1 + col2, col3 ...</td>
<td>B = FOREACH A GENERATE col1 + col2, col3;</td>
</tr>
<tr>
<td>...WHERE col2 &gt; 2</td>
<td>C = FILTER B by col2 &gt; 2;</td>
</tr>
<tr>
<td>SQL</td>
<td>Pig</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>SELECT col1, col2, sum(col3) FROM X GROUP BY col1, col2</td>
<td>D = GROUP A BY (col1, col2)</td>
</tr>
<tr>
<td></td>
<td>E = FOREACH D GENERATE</td>
</tr>
<tr>
<td></td>
<td>FLATTEN(group), SUM(A.col3);</td>
</tr>
<tr>
<td>...HAVING sum(col3) &gt; 5</td>
<td>F = FILTER E BY $2 &gt; 5;</td>
</tr>
<tr>
<td>...ORDER BY col1</td>
<td>G = ORDER F BY $0;</td>
</tr>
</tbody>
</table>
## SQL to Pig

<table>
<thead>
<tr>
<th>SQL</th>
<th>Pig</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT DISTINCT col1 from X</td>
<td>I = FOREACH A GENERATE col1;</td>
</tr>
<tr>
<td></td>
<td>J = DISTINCT I;</td>
</tr>
<tr>
<td>SELECT col1, count (DISTINCT col2) FROM X</td>
<td>K = GROUP A BY col1;</td>
</tr>
<tr>
<td>GROUP BY col1</td>
<td>L = FOREACH K {</td>
</tr>
<tr>
<td></td>
<td>M = DISTINCT A.col2;</td>
</tr>
<tr>
<td></td>
<td>GENERATE FLATTEN(group), count(M);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
# SQL to Pig

<table>
<thead>
<tr>
<th>SQL</th>
<th>Pig</th>
</tr>
</thead>
</table>
| `SELECT A.col1, B.col3 FROM A JOIN B USING (col1)` | `N = JOIN A by col1 INNER, B by col1 INNER;`  
  `O = FOREACH N GENERATE A.col1, B.col3;`  
  `-- Or`  
  `N = COGROUP A by col1 INNER, B by col1 INNER;`  
  `O = FOREACH N GENERATE flatten(A), flatten(B);`  
  `P = FOREACH O GENERATE A.col1, B.col3` |