Hadoop Operations
Managing Petabytes with Open Source

Jeff Hammerbacher
Chief Scientist and Vice President of Products, Cloudera
June 22, 2009
My Background
Thanks for Asking

- hammer@cloudera.com
- Studied Mathematics at Harvard
- Worked as a Quant on Wall Street
- Conceived, built, and led Data team at Facebook
  - Nearly 30 amazing engineers and data scientists
  - Several open source projects and research papers
- Founder of Cloudera
  - Building cost-effective data management tools for the world

Monday, June 22, 2009
Presentation Outline
Exceedingly Unlikely to Be Completed

- Hadoop overview and sample use cases
- Cloudera and Hadoop
- Hadoop project mechanics
- Cluster facilities, hardware, and system software
- Installation and configuration
- HDFS (main focus with limited time)
- MapReduce
- Cluster lifecycle and maintenance
- Questions and discussion

Monday, June 22, 2009
Presentation Sources

For Further Reading

▪ “Hadoop: The Definitive Guide”
  ▪ Tom White’s book from O’Reilly
  ▪ Many figures in this presentation taken from the book
▪ “Hadoop Cluster Management”
  ▪ Marco Nicosia’s USENIX 2009 presentation
▪ Cloudera blog and Get Satisfaction page
▪ Hadoop documentation
▪ MarkMail mailing list archives
▪ Hadoop wiki
What is Hadoop?

- Apache Software Foundation project, mostly written in Java
- Inspired by Google infrastructure
- Software for programming warehouse-scale computers (WSCs)
- Hundreds of production deployments

Project structure

- Hadoop Distributed File System (HDFS)
- Hadoop MapReduce
- Hadoop Common (formerly “Hadoop Core”)
- Other subprojects
  - Avro, HBase, Hive, Pig, Zookeeper
Anatomy of a Hadoop Cluster

- Commodity servers
  - 1 RU, 2 x 4 core CPU, 8 GB RAM, 4 x 1 TB SATA, 2 x 1 gE NIC
- Typically arranged in 2 level architecture
  - 40 nodes per rack
- Inexpensive to acquire and maintain
HDFS

- Pool commodity servers into a single hierarchical namespace
- Break files into 128 MB blocks and replicate blocks
- Designed for large files written once but read many times
- Two major daemons: NameNode and DataNode
  - NameNode manages file system metadata
  - DataNode manages data using local filesystem
- HDFS manages checksumming, replication, and compression
- Throughput scales nearly linearly with node cluster size
- Access from Java, C, command line, FUSE, WebDAV, or Thrift
  - Generally not mounted like a usual file system
HDFS
HDFS distributes file blocks among servers
Hadoop MapReduce

- Fault tolerant execution layer and API for parallel data processing
- Can target multiple storage systems
- Key/value data model
- Two major daemons: JobTracker and TaskTracker
- Many client interfaces
  - Java
  - C++
  - Streaming
  - Pig
  - SQL (Hive)
MapReduce

MapReduce pushes work out to the data
Hadoop Subprojects

- Avro
  - Cross-language serialization for RPC and persistent storage
- HBase
  - Table storage on top of HDFS, modeled after Google’s BigTable
- Hive
  - SQL interface to structured data stored in HDFS
- Pig
  - Language for dataflow programming
- Zookeeper
  - Coordination service for distributed systems
Facebook Data Infrastructure

2008

Scribe Tier

MySQL Tier

Hadoop Tier

Oracle RAC Servers
Hadoop at Yahoo!

- Jan 2006: Hired Doug Cutting
- Apr 2006: Sorted 1.9 TB on 188 nodes in 47 hours
- Apr 2008: Sorted 1 TB on 910 nodes in 209 seconds
- Aug 2008: Deployed 4,000 node Hadoop cluster
- May 2009: Sorted 1 TB on 1,460 nodes in 62 seconds
- Data Points
  - Over 25,000 nodes running Hadoop
  - Hundreds of thousands of jobs per day
  - Typical HDFS cluster: 1,400 nodes, 2 PB capacity
  - Sorted 1 PB on 3,658 nodes in 16.25 hours
Hadoop at Your Company

- Sample projects
  - Log or message warehouse
  - Database archival store
  - ETL into an existing data warehouse
  - Search team projects, e.g. autocomplete
  - Targeted web crawls
- Sample clusters
  - Retired database servers
  - Unused desktops
  - Amazon EC2
The Hadoop Community

- Over 750 (paid!) attendees at Hadoop Summit two weeks ago
  - Hadoop Summit East in New York in October
- Books from O’Reilly, Apress, and Manning
- Training videos free online
- Very active mailing lists and IRC channel
- Regular user group meetings in cities around the world
- University courses, also around the world
- Growing consultant and systems integrator expertise
- Commercial training, certification, and support from Cloudera
Cloudera and Hadoop

- Training: online, certification, and on site
- Support: yearly contract to get the most out of Hadoop
- Cloudera’s Distribution for Hadoop (Apache 2.0 licensed)
  - Simplifies upgrades and installation
  - Foundation and glue for Hadoop ecosystem
  - Dozens of supported clusters with thousands of nodes
  - Hundreds of unsupported clusters
- Exposure to a wide range of enterprise workloads
  - Computer vision, financial services, high-energy physics, telecom, bioinformatics, retail, media, and web
Hadoop Project Mechanics

- Trademark owned by Apache Software Foundation
- Apache 2.0 license used for code
- Related tools
  - Subversion for version control
  - JIRA for issue tracking
  - Ant for builds
  - Ivy for dependency tracking
  - JUnit for testing
  - Hudson for continuous integration
  - Javadoc and Forrest for documentation
Hadoop Project Mechanics

- Four classes of people in the Hadoop community
  - Hadoop PMC
  - Subproject committers
  - Subproject contributors
  - The unwashed masses

- Major organizations committing code
  - Yahoo!: Pig, Capacity Scheduler, Avro, etc.
  - Facebook: Hive, Fair Share scheduler, etc.
  - Cloudera: MRUnit, Sqoop, PyZK, Avro C bindings, etc.
  - You: http://wiki.apache.org/hadoop/HowToContribute
Hadoop Project Mechanics

- Release cycle of 3 months (–ish)
  - Last release: 0.20 on April 22, 2009
  - Subprojects on different release cycles
- Voting for a release
  - Feature freeze votes before release date
  - Releases put to a vote according to Apache guidelines
- Cutting an Apache release
  - Releases made available as tarballs on Apache and mirrors
  - Release notes at http://tinyurl.com/hadoop-releasenotes
Cluster Facilities and Hardware

▪ Data center: run Hadoop in a single data center, please
▪ Servers
  ▪ Clusters are often either capacity bound or CPU bound
  ▪ The 1U configuration specified previously is mostly standard
  ▪ Many organizations now testing 2U, 12 drive configurations
  ▪ Use ECC RAM and cheap hard drives: 7200 RPM SATA
  ▪ Start with standard 64-bit box for masters and workers
▪ Network
  ▪ Gigabit ethernet, 2 level tree, 5:1 oversubscription to core
  ▪ May want redundancy at top of rack and core
System Software

- Operating system: Linux, CentOS mildly preferred
- Local file system
  - ext3 versus xfs
  - Mount with noatime for performance improvements
- RAID configuration: RAID0 versus JBOD
- Java 6, update 14 or later (compressed ordinary object pointers)
- Useful unix utilities
  - sar, iostat, iftop, vmstat, nfsstat, strace, dmesg, and friends
- Useful java utilities
  - jps, jstack, jconsole
Installation and Configuration

  - Get Hadoop as RPM, Debs, AMI, or tarballs
  - Will put configuration in `/etc`, logs in `/var/log`
  - Registers services with `/etc/init.d`
  - Matches versions across subprojects
  - Backported bug fixes and extra Cloudera features

- Configuration: [http://my.cloudera.com](http://my.cloudera.com)
  - Need to decide if JT and NN live on same machine
  - Will have to manually specify topology
  - Can save your configuration for updating later

Monday, June 22, 2009
Installation
Hadoop Modes

- Standalone mode
  - Run all mappers and single reducer inside one JVM
- Pseudo-distributed mode
  - Run all daemons on single machine and use sockets for IPC
- Distributed mode
  - For production deployments
  - Can run master daemons on same box or separate boxes
Configuration

- org.apache.hadoop.conf package has Configuration class
- Configurations read their properties from resources
- Properties in later resources override those defined earlier
  - final keyword will prevent a property from being overwritten
- Site files contain site-specific configuration
  - core-site.xml
  - hdfs-site.xml
  - mapred-site.xml
- Default configurations in .template site files
Installation and Configuration

Operator Utilities

▪ Distributed shell
  ▪ Nice to have something like dsh
▪ Configuration management
  ▪ cfengine, Puppet, bcfg2, Chef, etc.
▪ Hadoop utilities
  ▪ hadoop-env.sh
  ▪ [start|stop]-dfs.sh
  ▪ [start|stop]-mapred.sh
Installation and Configuration

Common Problems

- Todd Lipcon: “the problem is almost always DNS”
- Open the necessary ports in your firewall
- Distribute ssh keys
- Make sure you have permission to write directories
- Use all of your disks
- Don’t share an NFS mount for large clusters
- Set JAVA_HOME appropriately
HDFS
NameNode

▪ VERSION specifies layoutVersion, among other information
▪ Two major data structures
  ▪ filesystem image
  ▪ edit log
▪ Secondary NameNode
  ▪ Checkpoints filesystem image and truncates edit log
  ▪ In 0.21, renamed to “checkpoint node”
  ▪ Also in 0.21, “backup node” added
    ▪ Replaces need to write data structures to NFS mount for durability
HDFS

DataNode

- Also has a VERSION file with layoutVersion information
- Stores data in local filesystem under `${dfs.data.dir}/current`
  - Data stored in blk_<id> files
  - Metadata (checksums) stored in blk_<id>.meta files
  - New subdirectory created for every dfs.data.numblocks
  - Round-robin blocks across directories
- `dfs.hosts[.exclude]` specifies allowed/removed DataNodes
- Serves data to client using a socket, not Hadoop RPC
HDFS
Client

- Can use Java libraries or command line for access
  - `libhdfs` has been behind the Java interface in last few releases
  - FUSE interface is unstable, so filesystem is not mounted
- Client only contacts NameNode for metadata
- Read path
  - Client keeps a list of block locations ranked by distance
- Write path
  - Client maintains two queues: data queue and ack queue
HDFS Read Path

Filters can only act on a file's name, as represented by a Path. They can't use a file's properties, such as creation time, as the basis of the filter. Nevertheless, they can perform matching that neither glob patterns nor regular expressions can achieve.

Deleting Data

Use the delete method on FileSystem to permanently remove files or directories:

```java
public boolean delete(Path f, boolean recursive) throws IOException
```

If `f` is a file or an empty directory, then the value of `recursive` is ignored. A non-empty directory is only deleted, along with its contents, if `recursive` is true (otherwise an IOException is thrown).

Data Flow

Anatomy of a File Read

To get an idea of how data flows between the client interacting with HDFS, the name-node and the datanode, consider Figure 3-1 which shows the main sequence of events when reading a file.
HDFS Write Path

1: create

2: create

3: write

4: write packet

5: ack packet

6: close

7: complete

dfs.replication.min

dfs.replication

Figure 3-3. A client writing data to HDFS Data Flow
HDFS
Operator Utilities

- Safe mode
- Filesystem check (fsck)
- dfsadmin
- Block scanner
- balancer
- archive
- distcp
- quotas: name space and disk space
HDFS
More Operator Utilities

- Users, groups, and permissions
- Audit logs
- Topology
- Web UIs
- Trash
- HDFS Proxy and Thriftfs
- Benchmarks and load testing
HDFS

Safe Mode

- NameNode automatically enters “safe mode” at startup
  - Loads the image file and applies edits from the edit log
  - Only metadata reads will work during safe mode
  - DataNodes send block lists to NameNode
  - Once 99.9% of blocks have reported, exit safe mode

- Configuration parameters
  - dfs.replication.min
  - dfs.safemode.threshold.pct
  - dfs.safemode.extension
HDFS
Filesystem Check

- Run with hadoop fsck
  - Speaks to NameNode and only examines metadata
- Evaluate health of file system
  - Minimally, over-, under-, and misreplicated blocks
  - Corrupt blocks
  - Missing replicas
- Can also be used to determine blocks for a file
  - hadoop fsck /path/to/file -files -blocks
HDFS
dfsadmin

- Tool used to perform most administrative operations
- Run via hadoop dfsadmin
  - Run with no arguments to see options
  - Most operations require superuser
  - Administer quotas
  - Commission or decommission DataNodes
  - Checkpoint the filesystem image
  - Check upgrade progress or finalize an upgrade
HDFS

DataBlockScanner

- Each DataNode runs its own block scanner
- Periodically verifies the checksum for each block
  - Reports corrupt blocks to NameNode for correction
- Built-in throttling to conserve bandwidth
- Runs every three weeks by default
  - Frequency controlled by dfs.datanode.scan.period.hours
- Web interface to block scanner
HDFS
Balancer

- Examines ratio of used space to total capacity
- Looks at this ratio for each node and the entire cluster
- Tries to bring all nodes within a configurable threshold of mean
- Run as background process
  - start-balancer.sh
- Only one balancer can be run against a single cluster
- Tries to throttle bandwidth used to 1 MB/s
  - Controlled via dfs.balance.bandwidthPerSec

Monday, June 22, 2009
HDFS Archive Tool

- HAR files are Hadoop Archives and use the .tar extension
- Conceptually similar to a .tar file
- Used to conserve namespace utilization
- Run via `hadoop archive -archiveName my.har /file1 ...`
- Will generate two index files and a number of part files
- Many files are concatenated into a small number of part files
- Index files enable lookup of individual files in the part files
- HAR files don’t support compression and are immutable
HDFS
distcp

- Distributed copy utility to move large amounts of data in parallel
- Can be controlled with some granularity
  - overwrite and update options
- Preserve attributes, ignore failures, throttle space used
- File globbing and filtering also supported
- Implemented as a MapReduce job with no reducers

Use cases
- Transfer data between clusters
- Bulk load data into a cluster
HDFS
Quotas

- Used to prevent runaway resource consumption
- Quotas apply to directories, not users or groups
- Quotas must be manually applied; no default quotas
- Namespace quotas
  - hadoop dfsadmin -[set|clr]Quota
- Disk space quotas
  - hadoop dfsadmin -[set|clr]SpaceQuota
HDFS
Users, Groups, and Permissions

- Enabled by default; control via dfs.permissions
- Every file and directory has an owner, group, and a mode
- Three types of permissions: read (r), write (w), execute (x)
  - Must have write permission on a directory to create/delete files
  - Must have execute permission to access children of directory
- The super-user is the identity of the NameNode process
- Client is assigned user and group of local process
  - Easy to spoof, so limit access to “gateway” cluster
HDFS
Audit Logs

- Not configured by default
- Particularly useful given the current state of security
- Can turn on by editing log4j.properties
  - Should also have it write to a separate file
- See http://wiki.apache.org/hadoop/HowToConfigure
HDFS
Topology

- Replica placement dictated by rack topology
- Distance calculated in multiple levels
  - node, rack, core switch
- Topology normally specified using ScriptBasedMapping
  - Control via topology.script.file.name
- Recent work on inferring topology from IP
HDFS
Web UIs

- Simple jsp user interfaces
- Can make edits from web UI
  - Runs with user and group set by dfs.web.ugi
- Web interfaces (port numbers)
  - NameNode: 50070, dfs.http.address
    - Also /metrics, /logLevel, /stacks
  - DataNode: 50075, dfs.datanode.http.address
  - Secondary NameNode: 50090, dfs.secondary.http.address
HDFS
HDFS Proxy and Thriftfs

▪ HDFS Proxy
  ▪ HTTP server that allows access by non-HDFS clients
▪ Thriftfs
  ▪ Thrift server that allows access by non-HDFS clients
Each user has a .Trash directory in their home directory

Files will remain in the trash for `fs.trash.interval` minutes

- Set to zero to disable the trash
- Trash is disabled by default

Enable the trash to prevent mistaken deletions

Programmatic access

- `moveToTrash()` and `expunge()`
HDFS
Benchmarks and Load Testing

▪ TestDFSIO
  ▪ Use a MapReduce job to read and write files in parallel
  ▪ Run without arguments to get options
    ▪ Can run read and write benchmarks
  ▪ Files are written under /benchmarks/TestDFSIO by default
    ▪ Control with test.build.data

▪ NNBench
  ▪ Load test the NameNode before deployment
HDFS
Common Problems

- Disk capacity!
  - Especially due to log file sizes
  - Crank up dfs.datanode.du.reserved
- Slow, but not dead, disks
- Checkpointing and backing up metadata
- Losing a write pipeline for long-lived writes
- Upgrades
- Many small files
Hadoop MapReduce

Overview

- **JobTracker**
  - Long-lived master daemon which distributes tasks
  - Maintains a job history of job execution statistics
- **TaskTrackers**
  - Long-lived client daemon which executes Map and Reduce tasks
- **Client**
  - Submits processing logic and necessary libraries to JobTracker
  - Performs input split calculation and waits for job success
The data flow for the general case of multiple reduce tasks is illustrated in Figure 2-3. This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as "the shuffle", as each reduce task is fed by many map tasks. Finally, it’s also possible to not have any reduce tasks. This can be appropriate when you don’t need the shuffle since the processing can be carried out entirely in parallel (a few examples are discussed in "NLineInputFormat" on page 186). In this case the only off-node data transfer is when the map tasks write to HDFS (see Figure 2-4).
Hadoop MapReduce Process Communication

In this chapter we'll look at how MapReduce in Hadoop works in detail. This knowledge provides a good foundation for writing more advanced MapReduce programs, which we will cover in the following two chapters.

Anatomy of a MapReduce Job Run

You can run a MapReduce job with a single line of code:

```
JobClient.runJob(conf)
```

It's very short, but it conceals a great deal of processing behind the scenes. This section uncovers the steps Hadoop takes to run a job.

The whole process is illustrated in Figure 6-1. At the highest level there are four independent entities:

- The client, which submits the MapReduce job.
- The jobtracker, which coordinates the job run. The jobtracker is a Java application whose main class is `JobTracker`.
- The tasktrackers, which run the tasks that the job has been split into. Tasktrackers are Java applications whose main class is `TaskTracker`.
- The distributed filesystem (normally HDFS, covered in Chapter 3), which is used for sharing job files between the other entities.

Figure 6-1. How Hadoop runs a MapReduce job

1: run job

2: get new job ID

4: submit job

6: retrieve input splits

7: heartbeat (returns task)

8: retrieve job resources

9: launch

10: run

MapTask or ReduceTask

TaskTracker

JobTracker

JobClient

MapReduce program

Shared filesystem (e.g. HDFS)

client JVM

client node

jobtracker node

tasktracker node

Monday, June 22, 2009
Hadoop MapReduce
Operator Utilities

- Tool interface
- Fair Share and Capacity schedulers
- Distributed Cache
- MRUnit
- IsolationRunner
- JobControl
- Web UIs
- Sqoop
Hadoop MapReduce
More Operator Utilities

- Counters
- Metrics
- Profiling tasks with HPROF
- Job History
- Benchmarks and load testing: Sort, MRBench, Gridmix
- Recover running jobs after restart
- JVM reuse
Hadoop MapReduce

Common Problems

- Debugging and testing large jobs
- Memory utilization of tasks
- Large jobs holding a cluster hostage
- Multi-stage MapReduce
- Overall cluster utilization
- JobTracker stability and memory utilization
- Distributing shared libraries
- Access to distributed logfiles
Cluster Lifecycle and Maintenance
Metrics and Monitoring

- Ganglia, jconsole, Nagios
- Metrics belong to a context
  - dfs, mapred, rpc, and jvm are current contexts
  - Metrics are aggregated at worker and at master daemons
  - Configured via conf/hadoop-metrics.properties
- Canary jobs
- Should also monitor some system properties
  - Ensure disks are writable and NICs remain configured correctly
Cluster Lifecycle and Maintenance

Upgrades

- Prepare for the upgrade
  - Clear out the temp directory
  - Run fsck to make sure the filesystem is healthy
  - Finalize the previous upgrade
  - Shut down MapReduce and kill any orphaned processes
  - Shut down HDFS and backup the NameNode directories
- Install the new version of HDFS and MapReduce on the cluster
- Start HDFS with the -upgrade option
- Sanity check and finalize the upgrade