This talk was originally presented at Apachecon Europe 2009 as part of Yahoo!’s outreach to the fledgeling Hadoop community. Since that time, a lot of advances have been made. The state of the art in Hadoop and Hadoop operations has moved forward significantly. In order to maintain some historical accuracy, this document contains the original slide deck with the only changes being “obsolete” marks to show information that is out of date and some minor tweaking of the speaker notes.

Following the original content is an addendum that has updated information on the “obsolete” slides and some additional tips/techniques that you will hopefully find useful.

Allen Wittenauer
aw@apache.org
Dear SysAdmin,

Please set up Hadoop using these machines. Let us know when they are ready for use.

Thanks,
The Users

Those of us in operations have all gotten a request like this at some point in time. What makes Hadoop a bit worse is that it doesn’t follow the normal rules of enterprise software. My hope with this talk is to help you navigate the waters for a successful deployment.
Of course, first, you need some hardware. :) At Yahoo!, we do things a bit on the extreme end. This is a picture of one of our data centers during one of our build outs. When finished, there will be about 10,000 machines here that will get used strictly for Hadoop.
Individual Node Configuration

- MapReduce slots tied to # of cores vs. memory

- DataNode reads/writes spread (statistically) even across drives

- hadoop-site.xml dfs.data.dir:
  
  ```xml
  <property>
    <name>dfs.data.dir</name>
    <value>/hadoop0,/hadoop1,/hadoop2,/hadoop3</value>
  </property>
  ```

- RAID
  - If any, mirror NameNode only
  - Slows DataNode in most configurations

Since we know from other presentations that each node runs X amount of tasks, it is important to note that slot usage is specific to the hardware in play vs. the task needs. If you only have 8GB of RAM on your nodes, you don’t want to configure 5 tasks that use 10GB of RAM each...

Disk space-wise, we are currently using generic 1u machines with four drives. We divide the file systems up so that we have 2 partitions, 1 for either root or swap and one for hadoop.

You’ll note that we want to use JBOD for compute node layout and use RAID only on the master nodes. HDFS will take care of the redundancy that RAID provides. The other reason we don’t want to use RAID is because we’ll take a pretty big performance hit. The speed of RAID is the speed of the slowest disk. Over time, the drive performance will degrade. In our tests, we saw a 30% performance degradation!
Hadoop has some key files it needs configured to know where hosts are at. The slaves file is only used by the start and stop scripts. This is also the only place where ssh is used.

dfs.include is the list of ALL of the datanodes in the system. dfs.exclude contains the nodes to exclude from the system. This doesn’t seem very useful on the surface... but we’ll get to why in the next slide.

This means the active list is the first list minus the last list.
• Add nodes
  – Add new nodes to dfs.include

• (Temporarily) Remove Nodes
  – Add nodes to dfs.exclude

• Update Node Lists and Decommission
  – hadoop dfsadmin -refreshNodes
    • Replicates blocks from any live nodes in the exclude list
  – Hint: Do not decommission too many nodes (200+) at once! Very easy to saturate namenode!

HDFS has the ability to dynamically shrink and grow itself using the two dfs files. Thus you can put nodes in the exclude file, trigger a refresh, and now the system will shrink itself on the fly!

When doing this at large scales, one needs to take care to not saturate the namenode with too many RPC requests. Additionally, we need to wary of the network and the node topology when we do a decommission...
Racks Of Nodes

• Each node
  – 1 connection to network switch
  – 1 connection to console server

• Dedicated
  – Name Nodes
  – Job Tackers
  – Data Loaders
  – ...

• More and More Racks...

In general, you’ll configure one switch and optionally one console server or OOB console switch per rack. It is important to designate gear on a per-rack basis to make sure you know the impact of a given rack going down. So one or more racks would be dedicated to your administrative needs while other racks would be dedicated to your compute nodes.
At Yahoo!, we use a mesh network so that the gear is essentially point-to-point via a Layer 3 network. Each switch becomes a very small network, with just enough IP addresses to cover those hosts. In our case, this is a /26.

We also have some protection against network issues as well as increasing the bandwidth by making sure each switch is tied to multiple cores.
Rack Awareness (HADOOP-692)

• Hadoop needs node layout (really network) information
  – Speed:
    • read/write prioritization (*)
      – local node
      – local rack
      – rest of system
  – Data integrity:
    • 3 replicas: write local -> write off-rack -> write on-the-other-rack -> return

• Default: flat network == all nodes of cluster are in one rack

• Topology program (provided by you) gives network information
  – hadoop-site.xml parameter: topology.script.file.name
  – Input: IP address  Output: /rack information

* or perhaps gettext("prioritization") ?

Where computes nodes are located network-wise is known as Rack Awareness or topology.

Rack awareness is important to Hadoop so that it can properly place code next to data as well as insure data integrity. It is so vital, that this should be done prior to placing any data on the system.

In order to tell Hadoop where the compute nodes are located in association with each other, we need to provide a very simple program that takes a hostname or ip address as input and provides a rack location as output.

In our design, we can easily leverage the “each rack as a network” to create a topology based upon the netmask.
Rack Awareness Example

- Four racks of /26 networks:
  - 192.168.1.1-63, 192.168.1.65-127,

- Four hosts on those racks:
  - sleepy 192.168.1.20  mars 192.168.1.73
  - frodo 192.168.1.145  athena 192.168.1.243

<table>
<thead>
<tr>
<th>Host to lookup</th>
<th>Topology Input</th>
<th>Topology Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleepy</td>
<td>192.168.1.20</td>
<td>/192.168.1.0</td>
</tr>
<tr>
<td>frodo</td>
<td>192.168.1.145</td>
<td>/192.168.1.128</td>
</tr>
<tr>
<td>mars</td>
<td>192.168.1.73</td>
<td>/192.168.1.64</td>
</tr>
<tr>
<td>athena</td>
<td>192.168.1.243</td>
<td>/192.168.1.192</td>
</tr>
</tbody>
</table>

So, let's take our design and see what would come out...
• Time passes
  – Blocks Added/Deleted
  – New Racks/Nodes

• Rebalancing places blocks uniformly across nodes
  – throttled so not to saturate network or name node
  – live operation; does not block normal work

• hadoop balancer [ -t <threshold> ]
  – (see also bin/start-balancer.sh)
  – threshold is % of over/under average utilization
    • 0 = perfect balance = balancer will likely not ever finish
  – Bandwidth Limited: 5 MB/s default, dfs.balance.bandwidthPerSec
    • per datanode setting, need to bounce datanode proc after changing!

• When to rebalance?

Over time, of course, block placement even with a topology in place can cause things to get out of balance. A recently introduced feature allows you to rebalance blocks across the grid without taking any downtime.

Of course, when should you rebalance? This is really part of a bigger set of questions...
• “What nodes are in what racks?”
• “How balanced is the data across the nodes?”
• “How much space is really used?”

• The big question is really:

“What is the health of my HDFS?”

• Primary tools
  – hadoop dfsadmin -fsck
  – hadoop dfsadmin -report
  – namenode status web page

The answer to these questions are generally available via three places: fsck output, the dfsadmin report, and on the namenode status page.
Yahoo! @ ApacheCon

hadoop fsck /

• Checks status of blocks, files and directories on the file system
  – Hint: Partial checks ok; provide path other than /
  – Hint: Run this nightly to watch for corruption

• Common Output:
  – A bunch of dots
    • Good blocks
  – Under replicated blk_XXXX. Target Replication is X but found Y replica(s)
    • Block is under replicated and will be re-replicated by namenode automatically
  – Replica placement policy is violated for blk_XXXX.
    • Block violates topology; need to fix this manually
  – MISSING X blocks of total size Y B
    • Block from the file is completely missing

A full fsck output should run at least nightly to verify the integrity of the system.

This is your EARLY WARNING SYSTEM. When things seem out of the ordinary, even things like under-replication, that’s your call to action to do something. It may be doing something as simple as increasing replication, waiting, then decreasing replication to force the NN to re-replicate something under replicated. It might be digging into the namenode logs to see why it thinks a block is corrupt. But in all cases, there is usually something for you to do.
Total size: 506115379265905 B (Total open files size: 4165942598 B)
Total dirs: 358015
Total files: 10488573 (Files currently being written: 246)
Total blocks (validated): 12823618 (avg. block size 39467440 B) (Total open file blocks (not validated): 51)
Minimally replicated blocks: 12823618 (100.000001 %)
Over-replicated blocks: 25197 (0.196489 %)
Under-replicated blocks: 9 (7.0183E-5 %)
**Mis-replicated blocks:** 1260 (0.00982562 %)
Default replication factor: 3
Average block replication: 3.005072
Corrupt blocks: 0
Missing replicas: 10 (2.5949832E-5 %)
Number of data-nodes: 1507
Number of racks: 42

The filesystem under path '/' is HEALTHY

Even though this is “good” as in “healthy”, we can see there are still some problems. They just aren’t “system down”-bad. We have 9 blocks that are unreplicated and 1260 that are in violation of our topology. Given the number of over-replicated blocks, there is a good chance that the system is or has recently undergone some compute nodes being added or removed recently.
Bad fsck Summary

Status: CORRUPT
Total size: 505307372371599 B (Total open files size: 2415919104 B)
Total dirs: 356465
Total files: 10416773 (Files currently being written: 478)
Total blocks (validated): 12763719 (avg. block size 39589352 B) (Total open file blocks (not validated): 288)

********************************
CORRUPT FILES: 1
MISSING BLOCKS: 1
MISSING SIZE: 91227974 B
CORRUPT BLOCKS: 1
********************************

Minimally replicated blocks: 12763718 (99.99999 %)
Over-replicated blocks: 970560 (7.6040535 %)
Under-replicated blocks: 4 (3.133883E-5 %)
Mis-replicated blocks: 1299 (0.0101772845 %)
Default replication factor: 3
Average block replication: 3.0837624
Corrupt blocks: 1
Missing replicas: 5 (1.2703163E-5 %)
Number of data-nodes: 1509
Number of racks: 42

The filesystem under path '/ is CORRUPT

Of course, corrupt is always bad and almost always signifies data loss. In this case, we lost a single block so have a corrupted file.

Which file had this and any other problems listed prior to this output.
dfsadmin -report will give you some highlights about the nodes that make up HDFS. In particular, you can see which nodes are a part of which racks in the topology. This is key, especially when working on a problem like mis-replicated block issues.

Sharp-eyed viewers will also see the bug on this page. :)

```
• hadoop dfsadmin -report

Total raw bytes: 233878511754448 (2.08 PB)
Remaining raw bytes: 237713230031670 (216.2 TB)
Used raw bytes: 1538976032374394 (1.37 PB)
% used: 65.8%

Total effective bytes: 0 (0 KB)
Effective replication multiplier: Infinity
-----------------------------------------------
Datanodes available: 1618

Name: 192.168.1.153:50010
Rack: /192.168.1.128
State : In Service
Total raw bytes: 1959385432064 (1.78 TB)
Remaining raw bytes: 234818330641(218.69 GB)
Used raw bytes: 1313761392777 (1.19 TB)
% used: 67.05%
Last contact: Thu Feb 19 21:57:01 UTC 2009
```
Yahoo! @ ApacheCon

NameNode Status

NameNode 'mynamenode.example.com:8020'

Started: Wed Feb 04 16:18:50 UTC 2009
Version: 0.18.3-2486615, r
Compiled: Thu Jan 29 16:43:04 UTC 2009 by hadoopqa
Upgrades: There are no upgrades in progress.

Browse the filesystem

Cluster Summary

10877074 files and directories, 12860721 blocks = 23737795 total. Heap Size is 13.57 GB / 13.57 GB (100%)
Capacity : 2.08 PB
DFS Remaining : 216.08 TB
DFS Used : 1.37 PB
DFS Used% : 65.81 %
Live Nodes : 1403
Dead Nodes : 215

Live Datanodes : 1403

<table>
<thead>
<tr>
<th>Node</th>
<th>Last Contact</th>
<th>Admin State</th>
<th>Size (TB)</th>
<th>Used (%)</th>
<th>Remaining (TB)</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleepy</td>
<td>2</td>
<td>In Service</td>
<td>1.42</td>
<td>67.19</td>
<td>0.11</td>
<td>25705</td>
</tr>
<tr>
<td>dopey</td>
<td>0</td>
<td>In Service</td>
<td>1.43</td>
<td>66.83</td>
<td>0.12</td>
<td>26399</td>
</tr>
<tr>
<td>grumpy</td>
<td>0</td>
<td>In Service</td>
<td>1.42</td>
<td>67.46</td>
<td>0.11</td>
<td>25789</td>
</tr>
</tbody>
</table>

Many people are familiar with this output. :)

This page is undergoing constant change and new features being added.
• Manages HDFS Metadata
  – in-memory (Java heap determines size of HDFS!)
  – on-disk

• Image file
  – static version that gets re-read on startup

• Edits file
  – log of changes to the static version since startup
  – Restarting namenode applies edits to the image file

• hadoop-site.xml:
  <property>
    <name>dfs.name.dir</name>
    <value>/hadoop/var/hdfs/name</value>
  </property>

There are a lot of details about the namenode that are probably low level, but important for ops people to know. The metadata about HDFS sits completely in memory while the namenode is running. It keeps on disk two important files that are only re-read upon startup: the image file and the edits file. You can think of the edits file as a database journal, as it only lists the changes to the image file.

The location of these files are dictated by a parameter called dfs.name.dir. This is the most important parameter to determine your state of recoverability....
NameNode: Your Single Point of Failure

- When NameNode dies, so does HDFS
- In practice, does not happen very often
- Multiple directories can be used for the on-disk image
  - `<value>/hadoop0/var/hdfs/name,/hadoop1/var/hdfs/name</value>`
  - sequentially written
  - 2nd directory on NFS means always having a copy
- Hint: Watch the disk space!
  - Namenode logs
  - image and edits file
  - audit logs (more on that later)

Yes, yes. There is a single point of failure. In our experiences, it actually isn’t that big of a deal for our batch loads. At this point in time, we’ve had approximately 3 failures where HA would have saved us.

Some other failure scenarios have to do with disk space. So watch it on the namenode machine!
Why NameNodes Fail

• Usually not a crash; brownout
  – Hint: Monitoring
    • Checking for dead process is a fail
    • Must check for service!

• Bugs
  – No, really.

• Hardware
  – Chances are low

• Misconfiguration
  – Not enough Java heap
  – Not enough physical RAM
    • swap=death

• As HDFS approaches full DataNodes cannot write add’l blocks
  – inability to replicate can send NameNode into death spiral

• Users doing bad things

So just why do NameNodes fail?
• When NN dies, bring up namenode on another machine
  – mount image file from NFS
  – create local directory path
  – change config to point to new name node
  – restart HDFS
  – NameNode process will populate local dir path with copy of NFS version

• Hint: Use an A/CNAME with small TTL for namenode in hadoop-site.xml
  – Move the A/CNAME to the new namenode
    • No config changes required on individual nodes
  – For CNAMEs, restart the DataNodes to pick up changes
    • See HADOOP-3988 for details

• But what about the secondary?

When it does fail, what should you do to recover? How to speed up recovery?
Something else to keep in mind: Java is notorious about caching IP addresses of things...
But what about this secondary name node thing?
First off, the 2nn is not about availability. It’s primary purpose is to compact the edits file into the image file.

While the 2nn is ‘technically’ optional, you really want to run it, especially for large grids. We had a problem at Yahoo! where the 2nn was down for quite a while because we didn’t have any monitoring on it. We discovered that it hadn’t been running the middle of a maintenance... which caused an extra 6 hours of downtime while the primary did the merge.

It *does* provide some level of redundancy in that the image file that will located on it will be X hours old, where X is the last time the edit compaction happened.
• Major user-created problems
  – Too many metadata operations
  – Too many files
  – Too many blocks

• Namespace quotas (0.18 HADOOP-3187)
  – Limit the number of files per directory
  – hadoop dfsadmin -setQuota # dir1 [dir2 ... dirn]
  – hadoop dfsadmin -clrQuota dir1 [dir2 ... dirn]

• Size quotas (0.19 HADOOP-3938, 0.18 HADOOP-5158)
  – Limit the total space used in a directory
  – hadoop dfsadmin -setspaceQuota # dir1 [dir2 ... dirn]
    • defaults to bytes, but can use abbreviations (e.g., 200g)
  – hadoop dfsadmin -clrspaceQuota dir1 [dir2 ... dirn]

Of course, what about those pesky users? There are some things you can do to help keep things in check. Quotas are one of the best tools in the system to keep processes from running amok.
More on Quotas

• Reminder: Directory-based, not User-based
  – /some/directory/path has a limit
  – user allen does not

• No defaults
  – User creation scripts need to set “default” quota

• No config file
  – Store as part of the metadata
  – HDFS must be up; no offline quota management

• Quota Reporting
  – hadoop fs -count -q dir1 [dir2 ...]
  – There is no “give me all quotas on the system” command
    • HADOOP-5290
Trash, The Silent Killer That Users Love

- Recovery of multi-TB files is hard

- `hadoop fs -rm /` client-side only feature
  - MR, Java API will not use `.Trash`

- Deleted files sent to HOMEDIR/`.Trash/Current`
  - "poor man’s snapshot"
  - `hadoop-site.xml`: `fs.trash.interval`
    - Number of minutes between cleanings

- Typical scenario:
  - Running out of space
  - Users delete massive amounts of files
  - Still out of space
  - Need to remove files out of trash to reclaim
Hadoop Permission System

• "Inspired" by POSIX and AFS
  – users, groups, world
  – read/write/execute
  – Group inheritance

• User and Group
  – Retrieved from client
  – Output of whoami, id, groups

• hadoop-site.xml: dfs.umask
  – umask used when creating files/dirs
  – **Decimal**, not octal
    • 63 not 077
HADOOP IS NOT SECURE! RUN FOR YOUR LIVES!

- Server never checks client info

- Permission checking is easily circumvented
  - App asks namenode for block #’s that make up file (regardless of read perms)
  - App asks datanode for those blocks

- Strategy 1: Who cares?

- Strategy 2: User/Data Separation
  - Firewall around Hadoop
  - Provision users only on grids with data they can use
  - Trust your users not to break the rules

At Yahoo!, we have some data that we need to prevent some users from seeing. In these instances we create separate grids and use a firewall around those grids to prevent unauthorized access.
Audit Logs (HADOOP-3336)

- When, Who, Where, How, What

2009-02-27 00:00:00,299 INFO org.apache.hadoop.fs.FSNamesystem.audit: ugi=allenw, users ip=/192.168.1.100 cmd=create src=/project/cool/data/ file1 dst=null perm=allenw:users:rw-------

- log4j.properties

```
log4j.logger.org.apache.hadoop.fs.FSNamesystem.audit=INFO,DRFAUDIT
log4j.additivity.org.apache.hadoop.fs.FSNamesystem.audit=false
log4j.appender.DRFAUDIT=org.apache.log4j.DailyRollingFileAppender
log4j.appender.DRFAUDIT.File=/var/log/hadoop-audit.log
log4j.appender.DRFAUDIT.DatePattern=.yyyy-MM-dd
log4j.appender.DRFAUDIT.layout=org.apache.log4j.PatternLayout
log4j.appender.DRFAUDIT.layout.ConversionPattern=%d{ISO8601} %p %c: %m%n
```

Audit logs are a great way to keep track of what files are being accessed, by who, when, etc. This has security uses of course but it also can give you an idea of data that may need a higher replication factor or maybe even be removed.
Multiple Grids

• Needed for
  – Security
  – Data redundancy

• How separate should they be?
  – Separate user for namenode, datanode, etc, processes?
  – Separate ssh keys?
  – Separate home directories for users?

• Data redundancy
  – Dedicated loading machines
  – Copying data between grids

So why would people need multiple grids? We’ve already talked about the security requirements but you might also need it for data and process redundancy.

When you need to make the decision about having multiple grids, there are a lot of key questions that need to be answered.

You also need to figure out how to keep data replicated, the architecture around that, etc.
distcp - distributed copy

- `hadoop distcp [flags] URL [URL ...] URL`
  - submits a map/reduce job to copy directories/files

  - copies block by block using Hadoop RPC
  - very fast

- **Important flags**
  - `p` = preserve various attributes, except modification time
  - `i` = ignore failures
  - `log` = write to a log file
  - `m` = number of copies (maps)
    - very easy to flood network if too many maps are used!
  - `filelimit / sizelimit` = limits the quantity of data to be copied
    - Another safety check against eating all bandwidth

One tool in the arsenal to use is distcp. It can be used to copy data back and forth between grids.
• hdfs method uses Hadoop RPC
  – versions of Hadoop must match!

• hadoop distcp hftp://nn1:50070/path/to/a/file hdfs://nn2:8020/another/path
  – file-level copy
  – slow
  – fairly version independent
  – must run on destination cluster
  – cannot write via hftp

• Uses a single port for copying

Of course, multiple grids also usually means points in time where there are multiple versions in play. When this happens, you’ll need to use hftp instead of distcp. Note that hftp is read only so you’ll need to run this on the grid that you want to write on.
Q & A
Addendum for USENIX ;login
December 2012
Rack Awareness: Pluggable

HADOOP-8469
• Topology as a pluggable class

Big benefits
• faster!
• fork()-less!
• Memory savings!

core-site.xml:

```xml
<property>
   <name>net.topology.node.switch.mapping.impl</name>
   <value>com.example.hadoop.network.myCoolTopology</value>
</property>
```

As part of Hadoop 1.x and later, the topology code now supports two modes of operation, the original ‘external’ version and now as a pluggable interface. The pluggable interface requires that one write a Java class that runs as part of the NameNode process.

Doing topology this way has some big benefits. Oracle/Sun Java 1.6 uses a “real” fork() instead of vfork() so this lowers the memory requirements of the NN, JT, and 2NN substantially. Additionally, since you don’t have to fire off another process, you get some speed savings as a bonus.
Quite a few changes have happened in and around the NameNode. Most of these are invisible to end users, but one is definitely worth highlighting against the backdrop of this old presentation.

In Apache Hadoop 2.0 and higher, the NameNode has gained a High Availability mode. This fixes one of the most requested features of HDFS, by finally providing a redundant metadata system with automatic failover.

As far as the implementation, this had some impacts on how things like the edits file was maintained/created. So no more single edits file. The current design also requires NFS in order for the two NameNodes to share state. A future design (which will likely be committed to the branch by the time you read this) uses other daemons to maintain the edits and state information without the use of NFS.
With Hadoop 1.x and higher, security has finally started to take root in the system. The goal was to make sure that users are who they say they are and we can do proper authorizations to resources based upon that identity. On the flip side, this means that Hadoop doesn’t encrypt its network traffic. (It should be noted, however, that the socket factory is pluggable if one really desires this...)

On the authentication front, there is now SASL support in place to make security extensible, with GSSAPI w/Kerberos included. One of the key points is that many many places have Active Directory up and running. With its built-in support for krb5, there is a very high chance that there is already a working Kerberos infrastructure in place. Because the system is SASL-based, it is possible for users to replace the Kerberos support with something else if they so desire. Just a bit of coding required...

With authentication in place, authorization and access controls are now possible. So those user and group permissions have now moved beyond just preventing users from accidentally deleting other users data. Additionally, queues and jobs have ACLs associated to prevent or enable people from working with the correct set of resources.

Since there is also web access to Hadoop resources, a new web filter is also bundled that provides SPNEGO support. This way, SSO with Kerberos works across the entire platform as expected. User’s browsing via the NameNode will have their identity recognized and can view the data their privileges allow them to view.
Permissions

- Overhauled to use octal instead of decimal
  - `dfs.umask = 022` works now

WebHDFS

- REST-based HDFS access
- Preferred over hftp for cross-version distcp
  - bidirectional
    - `hadoop distcp webhdfs://nn1:50070/path/to/a/file webhdfs://nn2:50070/another/path`

A few other corrections/procedure changes from all that time ago...

WebHDFS was added as a better way to connect to an HDFS without invoking all the version dependency issues. It uses a REST-style interface and allows for reads and writes. This is much preferred over using the HFTP method when copying data from one version of Hadoop to another.
Simple Performance Tips

Avoid the Interactive Trap
• Workflow Efficiency > Job Efficiency
• Grid Efficiency > Workflow Efficiency

Scheduling effort vs. Task CPU cycles

It is incredibly tempting to tune individual jobs to run as fast as possible. During the development cycle, we want to iterate quickly so take advantage of any extra resources that we can. But the long term results can be devastating, especially for batch processing. It isn’t unusual to see an optimization made early in a string of jobs causing performance issues later on. The holistic approach to tuning is key.

Something else to consider: if a dataset is to be updated once a day, does it matter if it takes an extra 30 minutes? The cost of that extra 30 minutes needs to be weighted heavily against resource availability for any other other work that may need to be run at the same time. This might mean tune to be a little slower while freeing up task slots or network or disk IO.

Along that line, make your CPU cycles count. If your tasks are whizzing by in times measured in seconds, that probably means there was more effort spent on scheduling the task than actually executing it. One might considering bumping the block size or somehow increasing the amount of data getting sent to a task. Sometimes compromises have to be made here for an overall efficiency goal, but those are usually edge cases.
Similarly, experience has shown that with many workloads, turning on compression is a must. There are two types of compression, one for the intermediate output used during the shuffle and one for the final output. In both cases, making sure that compression is in play can have a significant performance difference.

Compressing data during the shuffle can greatly reduce network overhead by simply making reducing the amount of data that needs to get transmitted across the wire in exchange for some CPU cycles. Unless the system is on a superfast network, this trade-off is almost certainly worth it.

Output compression has similar impacts. Not only do we reduce the amount of data written to disk and across the network (block replication), we also help insure that when our tasks are asked to execute, they get more input to process.

Another trick for multi-tenancy systems is to make sure that the value of slowstart makes sense for your computing environment. Slowstart controls the percentage of map tasks that need to be complete before the reducers fire off. By default, this is 5% which is almost certainly too low for any multi-tenant situation or jobs with long tails. These days, I’m personally configuring this to be 85% which seems to be a good value for a mix of workloads.
Links

Hadoop Wiki
  • http://wiki.apache.org/hadoop

Hadoop FAQ
  • http://wiki.apache.org/hadoop/FAQ