1. What is Hadoop?

Hadoop is a distributed computing platform written in Java. It incorporates features similar to those of the Google File System and of MapReduce.

2. What platforms and Java versions does Hadoop run on?

Java 1.6.x or higher, preferably from Sun -see HadoopJavaVersions in apache website for more details

Linux and Windows are the supported operating systems, but BSD, Mac OS/X, and OpenSolaris are known to work. (Windows requires the installation of Cygwin).

3. How well does Hadoop scale?

Hadoop has been demonstrated on clusters of up to 4000 nodes. Sort performance on 900 nodes is good (sorting 9TB of data on 900 nodes takes around 1.8 hours) and improving using these non-default configuration values:

```
dfs.block.size = 134217728
dfs.namenode.handler.count = 40
mapred.reduce.parallel.copies = 20
mapred.child.java.opts = -Xmx512m
fs.inmemory.size.mb = 200
io.sort.factor = 100
io.sort.mb = 200
io.file.buffer.size = 131072
```

Sort performances on 1400 nodes and 2000 nodes are pretty good too - sorting 14TB of data on a 1400-node cluster takes 2.2 hours; sorting 20TB on a 2000-node cluster takes 2.5 hours. The updates to the above configuration being:

```
mapred.job.tracker.handler.count = 60
mapred.reduce.parallel.copies = 50
tasktracker.http.threads = 50
mapred.child.java.opts = -Xmx1024m
```

4. What kind of hardware scales best for Hadoop?

The short answer is dual processor/dual core machines with 4-8GB of RAM using ECC memory, depending upon workflow needs. Machines should be moderately high-end commodity machines to be most cost-effective and typically cost 1/2 - 2/3 the cost of normal production application servers but are not desktop-class machines. This cost tends to be $2-5K.

5. I have a new node I want to add to a running Hadoop cluster; how do I start services on just one node?

This also applies to the case where a machine has crashed and rebooted, etc, and you need to get it to rejoin the cluster. You do not need to shutdown and/or restart the entire cluster in this case.

First, add the new node's DNS name to the conf/slaves file on the master node.

Then log in to the new slave node and execute:

```
$ cd path/to/hadoop
$ bin/hadoop-daemon.sh start datanode
$ bin/hadoop-daemon.sh start tasktracker
```

If you are using the dfs.include/mapred.include functionality, you will need to additionally add the node to the dfs.include/mapred.include file, then issue `hadoop dfsadmin -refreshNodes` and `hadoop mradmin -refreshNodes` so that the NameNode and JobTracker know of the additional node that has been added.

6. Is there an easy way to see the status and health of a cluster?

There are web-based interfaces to both the JobTracker (MapReduce master) and NameNode (HDFS master) which display status pages about the state of the entire system. By default, these are located at http://job.tracker.addr:50030/ and http://name.node.addr:50070/.

The JobTracker status page will display the state of all nodes, as well as the job queue and status about all currently running jobs and tasks. The NameNode status page will display the state of all nodes and the amount of free space, and provides the ability to browse the DFS via the web.

You can also see some basic HDFS cluster health data by running:

```
$ bin/hadoop dfsadmin -report
```

7. How much network bandwidth might I need between racks in a medium size (40-80 node) Hadoop cluster?

The true answer depends on the types of jobs you're running. As a back of the envelope calculation one might figure something like this:

60 nodes total on 2 racks = 30 nodes per rack Each node might process about 100MB/sec of data In the case of a sort job where the intermediate data is the same size as the input data, that means each node needs to shuffle 100MB/sec of data In aggregate, each rack is then producing about 3GB/sec of data However, given even reducer spread across the racks, each rack will need to send 1.5GB/sec to reducers running on the other rack. Since the connection is full duplex, that means you need 1.5GB/sec of bisection bandwidth for this theoretical job. So that's 12Gbps.

However, the above calculations are probably somewhat of an upper bound. A large number of jobs have significant data reduction during the map phase, either by some kind of filtering/selection going on in the Mapper itself, or by good usage of Combiners. Additionally, intermediate data compression can cut the intermediate data transfer by a significant factor. Lastly, although your disks can probably provide 100MB sustained throughput, it's rare to see a MR job which can sustain disk speed IO through the entire pipeline. So, I'd say my estimate is at least a factor of 2 too high.

So, the simple answer is that 4-6Gbps is most likely just fine for most practical jobs. If you want to be extra safe, many inexpensive switches can operate in a "stacked" configuration where the bandwidth

between them is essentially backplane speed. That should scale you to 96 nodes with plenty of headroom. Many inexpensive gigabit switches also have one or two 10GigE ports which can be used effectively to connect to each other or to a 10GE core.

8. How can I help to make Hadoop better?

If you have trouble figuring how to use Hadoop, then, once you've figured something out (perhaps with the help of the mailing lists), pass that knowledge on to others by adding something to this wiki.

9. I am seeing connection refused in the logs. How do I troubleshoot this?

---

10. Why is the 'hadoop.tmp.dir' config default user.name dependent?

We need a directory that a user can write and also not to interfere with other users. If we didn't include the username, then different users would share the same tmp directory. This can cause authorization problems, if folks' default umask doesn't permit write by others. It can also result in folks stomping on each other, when they're, e.g., playing with HDFS and re-format their filesystem.

11. Does Hadoop require SSH?

Hadoop provided scripts (e.g., start-mapred.sh and start-dfs.sh) use ssh in order to start and stop the various daemons and some other utilities. The Hadoop framework in itself does not **require** ssh. Daemons (e.g. TaskTracker and DataNode) can also be started manually on each node without the script's help.

12. What mailing lists are available for more help?

A description of all the mailing lists are on the http://hadoop.apache.org/mailing_lists.html page. In general:

general is for people interested in the administrivia of Hadoop (e.g., new release discussion). user@hadoop.apache.org is for people using the various components of the framework.

-dev mailing lists are for people who are changing the source code of the framework. For example, if you are implementing a new file system and want to know about the FileSystem API, hdfs-dev would be the appropriate mailing list.

13. What does "NFS: Cannot create lock on (some dir)" mean?

This actually is not a problem with Hadoop, but represents a problem with the setup of the environment it is operating.

Usually, this error means that the NFS server to which the process is writing does not support file system locks. NFS prior to v4 requires a locking service daemon to run (typically rpc.lockd) in order to provide this functionality. NFSv4 has file system locks built into the protocol.

In some (rarer) instances, it might represent a problem with certain Linux kernels that did not implement the flock() system call properly.

It is highly recommended that the only NFS connection in a Hadoop setup be the place where the

NameNode writes a secondary or tertiary copy of the fsimage and edits log. All other users of NFS are not recommended for optimal performance.

14. Do I have to write my job in Java?

No. There are several ways to incorporate non-Java code.

HadoopStreaming permits any shell command to be used as a map or reduce function.

libhdfs, a JNI-based C API for talking to hdfs (only).

Hadoop Pipes, a SWIG-compatible C++ API (non-JNI) to write map-reduce jobs.

15. How do I submit extra content (jars, static files, etc) for my job to use during runtime?

The distributed cache feature is used to distribute large read-only files that are needed by map/reduce jobs to the cluster. The framework will copy the necessary files from a URL (either hdfs: or http:) on to the slave node before any tasks for the job are executed on that node. The files are only copied once per job and so should not be modified by the application.

For streaming, see the HadoopStreaming wiki for more information.

Copying content into lib is not recommended and highly discouraged. Changes in that directory will require Hadoop services to be restarted.

16. How do I get my MapReduce Java Program to read the Cluster's set configuration and not just defaults?

The configuration property files ({core|mapred|hdfs}-site.xml) that are available in the various **conf/** directories of your Hadoop installation needs to be on the **CLASSPATH** of your Java application for it to get found and applied. Another way of ensuring that no set configuration gets overridden by any Job is to set those properties as final; for example:

```
<name>mapreduce.task.io.sort.mb</name>
<value>400</value>
<final>true</final>
```

Setting configuration properties as final is a common thing Administrators do, as is noted in the Configuration API docs.

A better alternative would be to have a service serve up the Cluster's configuration to you upon request, in code. https://issues.apache.org/jira/browse/HADOOP-5670 may be of some interest in this regard, perhaps.

17. Can I write create/write-to hdfs files directly from map/reduce tasks?

Yes. (Clearly, you want this since you need to create/write-to files other than the output-file written out by OutputCollector.)

Caveats:

${mapred.output.dir} is the eventual output directory for the job (JobConf.setOutputPath / JobConf.getOutputPath).

${taskid} is the actual id of the individual task-attempt (e.g. task_200709221812_0001_m_000000_0), a TIP is a bunch of ${taskid}s (e.g. task_200709221812_0001_m_000000).

With *speculative-execution* **on**, one could face issues with 2 instances of the same TIP (running simultaneously) trying to open/write-to the same file (path) on hdfs. Hence the app-writer will have to pick unique names (e.g. using the complete taskid i.e. task_200709221812_0001_m_000000_0) per task-attempt, not just per TIP. (Clearly, this needs to be done even if the user doesn't create/write-to files directly via reduce tasks.)

To get around this the framework helps the application-writer out by maintaining a special **$ {mapred.output.dir}/_${taskid}** sub-dir for each reduce task-attempt on hdfs where the output of the reduce task-attempt goes. On successful completion of the task-attempt the files in the $ {mapred.output.dir}/_${taskid} (of the successful taskid only) are moved to ${mapred.output.dir}. Of course, the framework discards the sub-directory of unsuccessful task-attempts. This is completely transparent to the application.

The application-writer can take advantage of this by creating any side-files required in $ {mapred.output.dir} during execution of his reduce-task, and the framework will move them out similarly - thus you don't have to pick unique paths per task-attempt.

Fine-print: the value of ${mapred.output.dir} during execution of a particular *reduce* task-attempt is actually ${mapred.output.dir}/_{$taskid}, not the value set by JobConf.setOutputPath. *So, just create any hdfs files you want in ${mapred.output.dir} from your reduce task to take advantage of this feature.*

For *map* task attempts, the automatic substitution of ${mapred.output.dir}/_${taskid} for $ {mapred.output.dir} does not take place. You can still access the map task attempt directory, though, by using FileOutputFormat.getWorkOutputPath(TaskInputOutputContext). Files created there will be dealt with as described above.

The entire discussion holds true for maps of jobs with reducer=NONE (i.e. 0 reduces) since output of the map, in that case, goes directly to hdfs.

18. How do I get each of a job's maps to work on one complete input-file and not allow the framework to split-up the files?

Essentially a job's input is represented by the InputFormat(interface)/FileInputFormat(base class).

For this purpose one would need a 'non-splittable' FileInputFormat i.e. an input-format which essentially tells the map-reduce framework that it cannot be split-up and processed. To do this you need your particular input-format to return **false** for the isSplittable call.

E.g.
org.apache.hadoop.mapred.SortValidator.RecordStatsChecker.NonSplitableSequenceFileInputFormat in src/test/org/apache/hadoop/mapred/SortValidator.java

In addition to implementing the InputFormat interface and having isSplitable(...) returning false, it is also necessary to implement the RecordReader interface for returning the whole content of the input file. (default is LineRecordReader, which splits the file into separate lines)

The other, quick-fix option, is to set mapred.min.split.size to large enough value.

19. Why I do see broken images in jobdetails.jsp page?

In hadoop-0.15, Map / Reduce task completion graphics are added. The graphs are produced as SVG(Scalable Vector Graphics) images, which are basically xml files, embedded in html content. The graphics are tested successfully in Firefox 2 on Ubuntu and MAC OS. However for other browsers, one should install an additional plugin to the browser to see the SVG images. Adobe's SVG Viewer can be

found at http://www.adobe.com/svg/viewer/install/.

20. I see a maximum of 2 maps/reduces spawned concurrently on each TaskTracker, how do I increase that?

Use the configuration knob: mapred.tasktracker.map.tasks.maximum and mapred.tasktracker.reduce.tasks.maximum to control the number of maps/reduces spawned simultaneously on a TaskTracker. By default, it is set to *2*, hence one sees a maximum of 2 maps and 2 reduces at a given instance on a TaskTracker.

You can set those on a per-tasktracker basis to accurately reflect your hardware (i.e. set those to higher nos. on a beefier tasktracker etc.).

21. Submitting map/reduce jobs as a different user doesn't work.

The problem is that you haven't configured your map/reduce system directory to a fixed value. The default works for single node systems, but not for "real" clusters. I like to use:

```
<property>
   <name>mapred.system.dir</name>
   <value>/hadoop/mapred/system</value>
   <description>The shared directory where MapReduce stores control files.
   </description>
</property>
```

Note that this directory is in your default file system and must be accessible from both the client and server machines and is typically in HDFS.

22. How do Map/Reduce InputSplit's handle record boundaries correctly?

It is the responsibility of the InputSplit's RecordReader to start and end at a record boundary. For SequenceFile's every 2k bytes has a 20 bytes **sync** mark between the records. These sync marks allow the RecordReader to seek to the start of the InputSplit, which contains a file, offset and length and find the first sync mark after the start of the split. The RecordReader continues processing records until it reaches the first sync mark after the end of the split. The first split of each file naturally starts immediately and not after the first sync mark. In this way, it is guaranteed that each record will be processed by exactly one mapper.

Text files are handled similarly, using newlines instead of sync marks.

23. How do I change final output file name with the desired name rather than in partitions like part-00000, part-00001?

You can subclass the OutputFormat.java class and write your own. You can look at the code of TextOutputFormat MultipleOutputFormat.java etc. for reference. It might be the case that you only need to do minor changes to any of the existing Output Format classes. To do that you can just subclass that class and override the methods you need to change.

24. When writing a New InputFormat, what is the format for the array of string returned by InputSplit\#getLocations()?

It appears that DatanodeID.getHost() is the standard place to retrieve this name, and the machineName variable, populated in DataNode.java\#startDataNode, is where the name is first set. The first method

attempted is to get "slave.host.name" from the configuration; if that is not available, DNS.getDefaultHost is used instead.

25. How do you gracefully stop a running job?

```
hadoop job -kill JOBID
```

26. How do I limit (or increase) the number of concurrent tasks a job may have running total at a time?

---

27. How do I limit (or increase) the number of concurrent tasks running on a node?

---

28. If I add new DataNodes to the cluster will HDFS move the blocks to the newly added nodes in order to balance disk space utilization between the nodes?

No, HDFS will not move blocks to new nodes automatically. However, newly created files will likely have their blocks placed on the new nodes.

There are several ways to rebalance the cluster manually.

Select a subset of files that take up a good percentage of your disk space; copy them to new locations in HDFS; remove the old copies of the files; rename the new copies to their original names.
A simpler way, with no interruption of service, is to turn up the replication of files, wait for transfers to stabilize, and then turn the replication back down.
Yet another way to re-balance blocks is to turn off the data-node, which is full, wait until its blocks are replicated, and then bring it back again. The over-replicated blocks will be randomly removed from different nodes, so you really get them rebalanced not just removed from the current node.
Finally, you can use the bin/start-balancer.sh command to run a balancing process to move blocks around the cluster automatically. See
HDFS User Guide: Rebalancer;

HDFS Tutorial: Rebalancing;

HDFS Commands Guide: balancer.

29. What is the purpose of the secondary name-node?

The term "secondary name-node" is somewhat misleading. It is not a name-node in the sense that data-nodes cannot connect to the secondary name-node, and in no event it can replace the primary name-node in case of its failure.

The only purpose of the secondary name-node is to perform periodic checkpoints. The secondary name-node periodically downloads current name-node image and edits log files, joins them into new image and uploads the new image back to the (primary and the only) name-node. See User Guide.

So if the name-node fails and you can restart it on the same physical node then there is no need to shutdown data-nodes, just the name-node need to be restarted. If you cannot use the old node anymore you will need to copy the latest image somewhere else. The latest image can be found either on the node that used to be the primary before failure if available; or on the secondary name-node. The latter

will be the latest checkpoint without subsequent edits logs, that is the most recent name space modifications may be missing there. You will also need to restart the whole cluster in this case.

30. Does the name-node stay in safe mode till all under-replicated files are fully replicated?

No. During safe mode replication of blocks is prohibited. The name-node awaits when all or majority of data-nodes report their blocks.

Depending on how safe mode parameters are configured the name-node will stay in safe mode until a specific percentage of blocks of the system is *minimally* replicated dfs.replication.min. If the safe mode threshold dfs.safemode.threshold.pct is set to 1 then all blocks of all files should be minimally replicated.

Minimal replication does not mean full replication. Some replicas may be missing and in order to replicate them the name-node needs to leave safe mode.

Learn more about safe mode in the HDFS Users' Guide.

31. How do I set up a hadoop node to use multiple volumes?

Data-nodes can store blocks in multiple directories typically allocated on different local disk drives. In order to setup multiple directories one needs to specify a comma separated list of pathnames as a value of the configuration parameter dfs.datanode.data.dir. Data-nodes will attempt to place equal amount of data in each of the directories.

The *name-node* also supports multiple directories, which in the case store the name space image and the edits log. The directories are specified via the dfs.namenode.name.dir configuration parameter. The name-node directories are used for the name space data replication so that the image and the log could be restored from the remaining volumes if one of them fails.

32. What happens if one Hadoop client renames a file or a directory containing this file while another client is still writing into it?

Starting with release hadoop-0.15, a file will appear in the name space as soon as it is created. If a writer is writing to a file and another client renames either the file itself or any of its path components, then the original writer will get an IOException either when it finishes writing to the current block or when it closes the file.

33. I want to make a large cluster smaller by taking out a bunch of nodes simultaneously. How can this be done?

On a large cluster removing one or two data-nodes will not lead to any data loss, because name-node will replicate their blocks as long as it will detect that the nodes are dead. With a large number of nodes getting removed or dying the probability of losing data is higher.

Hadoop offers the *decommission* feature to retire a set of existing data-nodes. The nodes to be retired should be included into the *exclude file*, and the exclude file name should be specified as a configuration parameter dfs.hosts.exclude. This file should have been specified during namenode startup. It could be a zero length file. You must use the full hostname, ip or ip:port format in this file. (Note that some users have trouble using the host name. If your namenode shows some nodes in "Live" and "Dead" but not decommission, try using the full ip:port.) Then the shell command

```
bin/hadoop dfsadmin -refreshNodes
```

should be called, which forces the name-node to re-read the exclude file and start the decommission process.

Decommission is not instant since it requires replication of potentially a large number of blocks and we do not want the cluster to be overwhelmed with just this one job. The decommission progress can be monitored on the name-node Web UI. Until all blocks are replicated the node will be in "Decommission In Progress" state. When decommission is done the state will change to "Decommissioned". The nodes can be removed whenever decommission is finished.

The decommission process can be terminated at any time by editing the configuration or the exclude files and repeating the `-refreshNodes` command.

34. Wildcard characters doesn't work correctly in FsShell.

When you issue a command in FsShell, you may want to apply that command to more than one file. FsShell provides a wildcard character to help you do so. The * (asterisk) character can be used to take the place of any set of characters. For example, if you would like to list all the files in your account which begin with the letter **x**, you could use the ls command with the * wildcard:

```
bin/hadoop dfs -ls x*
```

Sometimes, the native OS wildcard support causes unexpected results. To avoid this problem, Enclose the expression in **Single** or **Double** quotes and it should work correctly.

```
bin/hadoop dfs -ls 'in*'
```

35. Can I have multiple files in HDFS use different block sizes?

Yes. HDFS provides api to specify block size when you create a file.
FileSystem.create(Path, overwrite, bufferSize, replication, blockSize, progress) for details more.

36. Does HDFS make block boundaries between records?

No, HDFS does not provide record-oriented API and therefore is not aware of records and boundaries between them.

37. What happens when two clients try to write into the same HDFS file?

HDFS supports exclusive writes only.
When the first client contacts the name-node to open the file for writing, the name-node grants a lease to the client to create this file. When the second client tries to open the same file for writing, the name-node will see that the lease for the file is already granted to another client, and will reject the open request for the second client.

38. How to limit Data node's disk usage?

Use dfs.datanode.du.reserved configuration value in $HADOOP_HOME/conf/hdfs-site.xml for limiting disk usage.

```
<property>
  <name>dfs.datanode.du.reserved</name>
  <!-- cluster variant -->
  <value>182400</value>
```

```
    <description>Reserved space in bytes per volume. Always leave this much space
free for non dfs use.
    </description>
  </property>
```

39. On an individual data node, how do you balance the blocks on the disk?

Hadoop currently does not have a method by which to do this automatically. To do this manually:

Take down the HDFS
Use the UNIX mv command to move the individual blocks and meta pairs from one directory to another on each host
Restart the HDFS

40. What does "file could only be replicated to 0 nodes, instead of 1" mean?

The NameNode does not have any available DataNodes. This can be caused by a wide variety of reasons. Check the DataNode logs, the NameNode logs, network connectivity, ... Please see the page: CouldOnlyBeReplicatedTo

41. If the NameNode loses its only copy of the fsimage file, can the file system be recovered from the DataNodes?

No. This is why it is very important to configure dfs.namenode.name.dir to write to two filesystems on different physical hosts, use the SecondaryNameNode, etc.

42. I got a warning on the NameNode web UI "WARNING : There are about 32 missing blocks. Please check the log or run fsck." What does it mean?

This means that 32 blocks in your HDFS installation don't have a single replica on any of the live DataNodes.
Block replica files can be found on a DataNode in storage directories specified by configuration parameter dfs.datanode.data.dir. If the parameter is not set in the DataNode's `hdfs-site.xml`, then the default location `/tmp` will be used. This default is intended to be used only for testing. In a production system this is an easy way to lose actual data, as local OS may enforce recycling policies on `/tmp`. Thus the parameter must be overridden.
If dfs.datanode.data.dir correctly specifies storage directories on all DataNodes, then you might have a real data loss, which can be a result of faulty hardware or software bugs. If the file(s) containing missing blocks represent transient data or can be recovered from an external source, then the easiest way is to remove (and potentially restore) them. Run fsck in order to determine which files have missing blocks. If you would like (highly appreciated) to further investigate the cause of data loss, then you can dig into NameNode and DataNode logs. From the logs one can track the entire life cycle of a particular block and its replicas.

43. If a block size of 64MB is used and a file is written that uses less than 64MB, will 64MB of disk space be consumed?

Short answer: No.

Longer answer: Since HFDS does not do raw disk block storage, there are two block sizes in use when writing a file in HDFS: the HDFS blocks size and the underlying file system's block size. HDFS will

create files up to the size of the HDFS block size as well as a meta file that contains CRC32 checksums for that block. The underlying file system store that file as increments of its block size on the actual raw disk, just as it would any other file.

44. What are the problems if you are using hadoop and problems building the C/C++ Code

While most of Hadoop is built using Java, a larger and growing portion is being rewritten in C and C++. As a result, the code portability between platforms is going down. Part of the problem is the lack of access to platforms other than Linux and our tendency to use specific BSD, GNU, or System V functionality in places where the POSIX-usage is non-existent, difficult, or non-performant.

That said, the biggest loss of native compiled code will be mostly performance of the system and the security features present in newer releases of Hadoop. The other Hadoop features usually have Java analogs that work albeit slower than their C cousins. The exception to this is security, which absolutely requires compiled code.

45. What are the problems if you are using hadoop on Mac OS X 10.6

Be aware that Apache Hadoop 0.22 and earlier require Apache Forrest to build the documentation. As of Snow Leopard, Apple no longer ships Java 1.5 which Apache Forrest requires. This can be accomplished by either copying /System/Library/Frameworks/JavaVM.Framework/Versions/1.5 and 1.5.0 from a 10.5 machine or using a utility like Pacifist to install from an official Apple package. http://chxor.chxo.com/post/183013153/installing-java-1-5-on-snow-leopard provides some step-by-step directions.

46. Why do files and directories show up as DrWho and/or user names are missing/weird?

Prior to 0.22, Hadoop uses the 'whoami' and id commands to determine the user and groups of the running process. whoami ships as part of the BSD compatibility package and is normally not in the path. The id command's output is System V-style whereas Hadoop expects POSIX. Two changes to the environment are required to fix this:

Make sure /usr/ucb/whoami is installed and in the path, either by including /usr/ucb at the tail end of the PATH environment or symlinking /usr/ucb/whoami directly.
In hadoop-env.sh, change the HADOOP_IDENT_STRING thusly:

```
export HADOOP_IDENT_STRING=`/usr/xpg4/bin/id -u -n`
```

47. Hadoop Reported disk capacities are wrong

Hadoop uses du and df to determine disk space used. On pooled storage systems that report total capacity of the entire pool (such as ZFS) rather than the filesystem, Hadoop gets easily confused. Users have reported that using fixed quota sizes for HDFS and MapReduce directories helps eliminate a lot of this confusion.

48. What are the problems if you are using hadoop or Building / Testing Hadoop on Windows

The Hadoop build on Windows can be run from inside a Windows (not cygwin) command prompt window.

Whether you set environment variables in a batch file or in System->Properties->Advanced->Environment Variables, the following environment variables need to be set:

```
set ANT_HOME=c:\apache-ant-1.7.1
set JAVA_HOME=c:\jdk1.6.0.4
set PATH=%PATH%;%ANT_HOME%\bin
```

then open a command prompt window, cd to your workspace directory (in my case it is c:\workspace\hadoop) and run ant. Since I am interested in running the contrib test cases I do the following:

```
ant -l build.log -Dtest.output=yes test-contrib
```

other targets work similarly. I just wanted to document this because I spent some time trying to figure out why the ant build would not run from a cygwin command prompt window. If you are building/testing on Windows, and haven't figured it out yet, this should get you started.

1. What is Hadoop framework?
Ans: Hadoop is a open source framework which is written in java by apche software foundation. This framework is used to wirite software application which requires to process vast amount of data (It could handle multi tera bytes of data). It works in-paralle on large clusters which could have 1000 of computers (Nodes) on the clusters. It also process data very reliably and fault-tolerant manner.

2. On What concept the Hadoop framework works?
Ans : It works on MapReduce, and it is devised by the Google.

3. What is MapReduce ?
Ans: Map reduce is an algorithm or concept to process Huge amount of data in a faster way. As per its name you can divide it Map and Reduce.
 The main MapReduce job usually splits the input data-set into independent chunks. (Big data sets in the multiple small datasets)

 MapTask: will process these chunks in a completely parallel manner (One node can process one or more chunks).

 The framework sorts the outputs of the maps.

 Reduce Task : And the above output will be the input for the reducetasks, produces the final result.

Your business logic would be written in the MappedTask and ReducedTask.
Typically both the input and the output of the job are stored in a file-system (Not database). The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

4. What is compute and Storage nodes?
Ans:
Compute Node: This is the computer or machine where your actual business logic will be executed.
Storage Node: This is the computer or machine where your file system reside to store the processing data.
In most of the cases compute node and storage node would be the same machine.

5. How does master slave architecture in the Hadoop?
Ans: The MapReduce framework consists of a single master **JobTracker** and multiple slaves, each cluster-node will have one **TaskskTracker**.
The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

6. How does an Hadoop application look like or their basic components?
Ans: Minimally an Hadoop application would have following components.
- Input location of data
- Output location of processed data.
- A map task.
- A reduced task.
- Job configuration

The Hadoop job client then submits the job (jar/executable etc.) and configuration to the **JobTracker** which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

7. Explain how input and output data format of the Hadoop framework?
Ans: The MapReduce framework operates exclusively on pairs, that is, the framework views the input to the job as a set of pairs and produces a set of pairs as the output of the job, **conceivably of different types**. See the flow mentioned below
(input) -> **map** -> -> **combine/sorting** -> -> **reduce** -> (output)

8. What are the restriction to the key and value class ?
Ans: The key and value classes have to be serialized by the framework. To make them serializable Hadoop provides a **Writable** interface. As you know from the java itself that the key of the Map should be comparable, hence the key has to implement one more interface **WritableComparable.**

9. Explain the WordCount implementation via Hadoop framework ?
Ans: We will count the words in all the input file flow as below
- **input**

Assume there are two files each having a sentence
```
Hello World Hello World (In file 1)
Hello World Hello World (In file 2)
```
- **Mapper : There would be each mapper for the a file**

For the given sample input the first map output:
```
< Hello, 1>
< World, 1>
< Hello, 1>
< World, 1>
```
The second map output:
```
< Hello, 1>
< World, 1>
```

< Hello, 1>
< World, 1>

- **Combiner/Sorting (This is done for each individual map)**

So output looks like this
The output of the first map:
< Hello, 2>
< World, 2>
The output of the second map:
< Hello, 2>
< World, 2>

- **Reducer :**

It sums up the above output and generates the output as below
< Hello, 4>
< World, 4>

- **Output**

Final output would look like
Hello 4 times
World 4 times

10. Which interface needs to be implemented to create Mapper and Reducer for the Hadoop?
Ans:
`org.apache.hadoop.mapreduce.Mapper`
`org.apache.hadoop.mapreduce.Reducer`

11. What Mapper does?
Ans: Maps are the individual tasks that transform i
nput records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.

12. What is the InputSplit in map reduce software?
Ans: An **InputSplit** is a logical representation of a unit (A chunk) of input work for a map task; e.g., a filename and a byte range within that file to process or a row set in a text file.

13.What is the InputFormat ?
Ans: The **InputFormat** is responsible for enumerate (itemise) the **InputSplits**, and producing a **RecordReader** which will turn those logical work units into actual physical input records.

14. Where do you specify the Mapper Implementation?
Ans: Generally mapper implementation is specified in the Job itself.

15. How Mapper is instantiated in a running job?
Ans: The Mapper itself is instantiated in the running job, and will be passed a MapContext object which it can use to configure itself.

16. Which are the methods in the Mapper interface?
Ans : The Mapper contains the run() method, which call its own setup() method only once, it also call a map() method for each input and finally calls it cleanup() method. All above methods you can override in your code.

17. What happens if you don't override the Mapper methods and keep them as it is?
Ans: If you do not override any methods (leaving even map as-is), it will act as the identity function, emitting each input record as a separate output.

18. What is the use of Context object?
Ans: The Context object allows the mapper to interact with the rest of the Hadoop system. It Includes configuration data for the job, as well as interfaces which allow it to emit output.

19. How can you add the arbitrary key-value pairs in your mapper?
Ans: You can set arbitrary (key, value) pairs of configuration data in your Job, e.g. with *Job.getConfiguration().set("myKey", "myVal")*, and then retrieve this data in your mapper with *Context.getConfiguration().get("myKey")*. This kind of functionality is typically done in the Mapper's setup() method.

20. How does Mapper's run() method works?
Ans: The Mapper.run() method then calls map*(KeyInType, ValInType, Context)* for each key/value pair in the **InputSplit** for that task

21. Which object can be used to get the progress of a particular job ?
Ans: `Context`

22. What is next step after Mapper or MapTask?
Ans : The output of the Mapper are sorted and Partitions will be created for the output. Number of partition depends on the number of reducer.

23. How can we control particular key should go in a specific reducer?
Ans: Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner.

24. What is the use of Combiner?
Ans: It is an optional component or class, and can be specify via **Job.setCombinerClass(ClassName)**, to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.

25. How many maps are there in a particular Job?
Ans: The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.
Generally it is around 10-100 maps per-node. Task setup takes awhile, so it is best if the maps take at least a minute to execute.
Suppose, if you expect 10TB of input data and have a blocksize of 128MB, you'll end up with 82,000 maps, to control the number of block you can use the **mapreduce.job.maps** parameter (which only provides a hint to the framework).
Ultimately, the number of tasks is controlled by the number of splits returned by the **InputFormat.getSplits()** method (which you can override).

26. What is the Reducer used for?
Ans: **Reducer** reduces a set of intermediate values which share a key to a (usually smaller) set of values.
The number of reduces for the job is set by the user via **Job.setNumReduceTasks(int)**.

27. Explain the core methods of the Reducer?
Ans: The API of **Reducer** is very similar to that of Mapper, there's a **run()** method that receives a **Context** containing the job's configuration as well as interfacing methods that return data from the reducer itself back to the framework. The **run()** method calls **setup()** once, **reduce()** once for each key associated with the reduce task, and **cleanup()** once at the end. Each of these methods can access the job's configuration data by using **Context.getConfiguration()**.
As in Mapper, any or all of these methods can be overridden with custom implementations. If none of these methods are overridden, the default reducer operation is the identity function; values are passed through without further processing.
The heart of Reducer is its **reduce**() method. This is called once per key; the second argument is an **Iterable** which returns all the values associated with that key.

28.What are the primary phases of the Reducer?
Ans: Shuffle, Sort and Reduce

29. Explain the shuffle?
Ans: Input to the `Reducer` is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via HTTP.

30. Explain the Reducer's Sort phase?
Ans: The framework groups `Reducer` inputs by keys (since different mappers may have output the same key) in this stage. The shuffle and sort phases occur simultaneously; while map-outputs are being fetched they are merged (It is similar to merge-sort).

31. Explain the Reducer's reduce phase?
Ans: In this phase the ***reduce(MapOutKeyType, Iterable, Context)*** method is called for each pair in the grouped inputs. The output of the reduce task is typically written to the **FileSystem** via ***Context.write(ReduceOutKeyType, ReduceOutValType)***. Applications can use the Context to report progress, set application-level status messages and update Counters, or just indicate that they are alive. The output of the Reducer is not sorted.

32. How many Reducers should be configured?
Ans: The right number of reduces seems to be `0.95` or `1.75` multiplied by (*<no. of nodes>* * `mapreduce.tasktracker.reduce.tasks.maximum`).
With 0.95 all of the reduces can launch immediately and start transfering map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing. Increasing the number of reduces increases the framework overhead, but increases load balancing and lowers the cost of failures.

33. It can be possible that a Job has 0 reducers?
Ans: It is legal to set the number of reduce-tasks to zero if no reduction is desired.

34. What happens if number of reducers are 0?
Ans: In this case the outputs of the map-tasks go directly to the FileSystem, into the output path set by **setOutputPath(Path)**. The framework does not sort the map-outputs before writing them out to the FileSystem.

35. How many instances of JobTracker can run on a Hadoop Cluser?
Ans: Only one

36. What is the JobTracker and what it performs in a Hadoop Cluster?
Ans: JobTracker is a daemon service which submits and tracks the MapReduce tasks to the Hadoop cluster. It runs its own JVM process. And usually it run on a separate machine, and each slave node is configured with job tracker node location.
The JobTracker is single point of failure for the Hadoop MapReduce service. If it goes down, all running jobs are halted.
JobTracker in Hadoop performs following actions

- Client applications submit jobs to the Job tracker.
- The JobTracker talks to the NameNode to determine the location of the data
- The JobTracker locates TaskTracker nodes with available slots at or near the data
- The JobTracker submits the work to the chosen TaskTracker nodes.
- The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.
- A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may may even blacklist the TaskTracker as unreliable.
- When the work is completed, the JobTracker updates its status.
- Client applications can poll the JobTracker for information.

37. How a task is scheduled by a JobTracker?
Ans: The TaskTrackers send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive. These messages also inform the JobTracker of the number of available slots, so the JobTracker can stay up to date with where in the cluster work can be delegated. When the JobTracker tries to find somewhere to schedule a task within the MapReduce operations, it first looks for an empty slot on the same server that hosts the DataNode containing the data, and if not, it looks for an empty slot on a machine in the same rack.

38. How many instances of Tasktracker run on a Hadoop cluster?
Ans: There is one Daemon Tasktracker process for each slave node in the Hadoop cluster.

39. What are the two main parts of the Hadoop framework?
Ans: Hadoop consists of two main parts
 Hadoop distributed file system, a distributed file system with high throughput,

 Hadoop MapReduce, a software framework for processing large data sets.

40.Explain the use of TaskTracker in the Hadoop cluster?
Ans: A Tasktracker is a slave node in the cluster which that accepts the tasks from JobTracker like Map, Reduce or shuffle operation. Tasktracker also runs in its own JVM Process.
Every TaskTracker is configured with a set of slots; these indicate the number of tasks that it can accept. The TaskTracker starts a separate JVM processes to do the actual work (called as Task Instance) this is to ensure that process failure does not take down the task tracker.
The Tasktracker monitors these task instances, capturing the output and exit codes. When the Task instances finish, successfully or not, the task tracker notifies the JobTracker.
The TaskTrackers also send out heartbeat messages to the JobTracker, usually every few

minutes, to reassure the JobTracker that it is still alive. These messages also inform the JobTracker of the number of available slots, so the JobTracker can stay up to date with where in the cluster work can be delegated.

41. What do you mean by TaskInstance?
Ans: Task instances are the actual **MapReduce jobs** which run on each slave node. The TaskTracker starts a separate JVM processes to do the actual work (called as Task Instance) this is to ensure that process failure does not take down the entire task tracker.Each Task Instance runs on its own JVM process. There can be multiple processes of task instance running on a slave node. This is based on the number of slots configured on task tracker. By default a new task instance JVM process is spawned for a task.

42. How many daemon processes run on a Hadoop cluster?
Ans: Hadoop is comprised of five separate daemons. Each of these daemons runs in its own JVM.
Following 3 Daemons run on Master nodes.**NameNode** - This daemon stores and maintains the metadata for HDFS. **Secondary NameNode** - Performs housekeeping functions for the NameNode. **JobTracker** - Manages MapReduce jobs, distributes individual tasks to machines running the Task Tracker. Following 2 Daemons run on each Slave nodes **DataNode** – Stores actual HDFS data blocks. **TaskTracker** – It is Responsible for instantiating and monitoring individual Map and Reduce tasks.

43. How many maximum JVM can run on a slave node?
Ans: One or Multiple instances of Task Instance can run on each slave node. Each task instance is run as a separate JVM process. The number of Task instances can be controlled by configuration. Typically a high end machine is configured to run more task instances.

44. What is NAS?
Ans: It is one kind of file system where data can reside on one centralized machine and all the cluster member will read write data from that shared database, which would not be as efficient as HDFS.

45. How HDFA differs with NFS?
Ans: Following are differences between HDFS and NAS
1.
- In HDFS Data Blocks are distributed across local drives of all machines in a cluster. Whereas in NAS data is stored

on dedicated hardware.
- HDFS is designed to work with MapReduce System, since computation is moved to data. NAS is not suitable for MapReduce since data is stored separately from the computations.
- HDFS runs on a cluster of machines and provides redundancy using replication protocol. Whereas NAS is provided by a single machine therefore does not provide data redundancy.

46. How does a NameNode handle the failure of the data nodes?
Ans: HDFS has master/slave architecture. An HDFS cluster consists of a single **NameNode**, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.
The NameNode and DataNode are pieces of software designed to run on commodity machines.

NameNode periodically receives a Heartbeat and a Block report from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode. When NameNode notices that it has not received a heartbeat message from a data node after a certain amount of time, the data node is marked as dead. Since blocks will be under replicated the system begins replicating the blocks that were stored on the dead DataNode. The NameNode Orchestrates the replication of data blocks from one DataNode to another. The replication data transfer happens directly between DataNode and the data never passes through the NameNode.

47. Can Reducer talk with each other?
Ans: No, Reducer runs in isolation.

48. Where the Mapper's Intermediate data will be stored?
Ans: The mapper output (intermediate data) is stored on the Local file system (NOT HDFS) of each individual mapper nodes. This is typically a temporary directory location which can be setup in config by the Hadoop administrator. The intermediate data is cleaned up after the Hadoop Job completes.

49. What is the use of Combiners in the Hadoop framework?
Ans: Combiners are used to increase the efficiency of a MapReduce program. They are used to aggregate intermediate map output locally on individual mapper outputs. Combiners can help you reduce the amount of data that needs to be transferred across to the reducers.
You can use your reducer code as a combiner if the operation performed is commutative and associative.
The execution of combiner is not guaranteed; Hadoop may or may not execute a combiner. Also, if required it may execute it more than 1 times. Therefore your MapReduce jobs should not depend on the combiners' execution.

50. What is the Hadoop MapReduce API contract for a key and value Class?
Ans:
◦The Key must implement the **org.apache.hadoop.io.WritableComparable** interface.
◦The value must implement the **org.apache.hadoop.io.Writable** interface.

51. What is a IdentityMapper and IdentityReducer in MapReduce?
◦**org.apache.hadoop.mapred.lib.IdentityMapper**: Implements the identity function, mapping inputs directly to outputs. If MapReduce programmer does not set the
Mapper Class using **JobConf.setMapperClass** then **IdentityMapper.class** is used as a default value.
◦**org.apache.hadoop.mapred.lib.IdentityReducer** : Performs no reduction, writing all input values directly to the output. If MapReduce programmer does not set the Reducer Class using **JobConf.setReducerClass** then **IdentityReducer.class** is used as a default value.

52. What is the meaning of speculative execution in Hadoop? Why is it important?
Ans: Speculative execution is a way of coping with individual Machine performance. In large clusters where hundreds or thousands of machines are involved there may be machines which are not performing as fast as others.
This may result in delays in a full job due to only one machine not performaing well. To avoid this, speculative execution in hadoop can run multiple copies of same map or reduce task on different slave nodes. The results from first node to finish are used.

53. When the reducers are are started in a MapReduce job?

Ans: In a MapReduce job reducers do not start executing the reduce method until the all Map jobs have completed. Reducers start copying intermediate key-value pairs from the mappers as soon as they are available. The programmer defined reduce method is called only after all the mappers have finished.
If reducers do not start before all mappers finish then why does the progress on MapReduce job shows something like Map(50%) Reduce(10%)? Why reducers progress percentage is displayed when mapper is not finished yet?
Reducers start copying intermediate key-value pairs from the mappers as soon as they are available. The progress calculation also takes in account the processing of data transfer which is done by reduce process, therefore the reduce progress starts showing up as soon as any intermediate key-value pair for a mapper is available to be transferred to reducer.
Though the reducer progress is updated still the programmer defined reduce method is called only after all the mappers have finished.

54. What is HDFS ? How it is different from traditional file systems?
HDFS, the Hadoop Distributed File System, is responsible for storing huge data on the cluster. This is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant.
◦HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
◦HDFS provides high throughput access to application data and is suitable for applications that have large data sets.
◦HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files.

55. What is HDFS Block size? How is it different from traditional file system block size?
In HDFS data is split into blocks and distributed across multiple nodes in the cluster. Each block is typically 64Mb or 128Mb in size.
Each block is replicated multiple times. Default is to replicate each block three times. Replicas are stored on different nodes. HDFS utilizes the local file system to store each HDFS block as a separate file. HDFS Block size can not be compared with the traditional file system block size.

57. What is a NameNode? How many instances of NameNode run on a Hadoop Cluster?
The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself.
There is only One NameNode process run on any hadoop cluster. NameNode runs on its own JVM process. In a typical production cluster its run on a separate machine.
The NameNode is a Single Point of Failure for the HDFS Cluster. When the NameNode goes down, the file system goes offline.
Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives.

58. What is a DataNode? How many instances of DataNode run on a Hadoop Cluster?
A DataNode stores data in the Hadoop File System HDFS. There is only One DataNode process run on any hadoop slave node. DataNode runs on its own JVM process. On startup, a DataNode connects to the NameNode. DataNode instances can talk to each other, this is mostly during replicating data.

59. How the Client communicates with HDFS?
The Client communication to HDFS happens using Hadoop HDFS API. Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file on HDFS. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives. Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data.

60. How the HDFS Blocks are replicated?
HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.
The NameNode makes all decisions regarding replication of blocks. HDFS uses rack-aware replica placement policy. In default configuration there are total 3 copies of a datablock on HDFS, 2 copies are stored on datanodes on same rack and 3rd copy on a different rack.