1:

Computer cluster - a computer cluster used for Hadoop is called Hadoop Cluster. Hadoop cluster is a type of computational cluster designed for store and analyze huge amount of data which is unstructured in a distributed computing System.The clusters mostly process on low cost commodity computers.

Client machines, Masters nodes, and Slave nodes are the machine roles in a Hadoop deployment . The Master nodes look after two key functions storing lots of data (HDFS), and running parallel computations on all that data (Map Reduce). The Name Node controlsthe storage (HDFS) and Job Tracker coordinates to Map Reduce. Slave Nodes do work of storing the data and running the computations. Each slave runs both a Data Node and Task Tracker daemon that communicate with each and follow instructions by master. The Task Tracker daemon is a slave to the Job Tracker, the Data Node daemon a slave to the Name Node.

Hadoop-cluster - Client machines have Hadoop installed with all the cluster settings, but are neither a Master or a Slave. the Client machine load data into the cluster, submit Map Reduce jobs describing how that data should be processed, and then process results of the job when done. Hadoop runs best on Linux machines, working directly with the underlying hardware. That said, Hadoop does work in a virtual machine. That’s a great way to learn and get Hadoop up and running fast and cheap. In typical architecture of a Hadoop cluster(Refer attachment assisgn2.5.1). We have rack servers (not blades) populated in racks connected to a top of rack switch usually with 1 or 2 GE boned links. 10GE nodes are uncommon but gaining interest as machines continue to get more dense with CPU cores and disk drives. The rack switch has uplinks connected to another tier of switches connecting all the other racks with uniform bandwidth, forming the cluster. The majority of the servers will be Slave nodes with lots of local disk storage and moderate amounts of CPU and DRAM. Some of the machines will be Master nodes that might have a slightly different configuration favoring more DRAM and CPU, less local storage.

2:-

Hadoop components are rack-aware. lets consider, HDFS block placement will use rack awareness for fault tolerance by placing one block replica on a different rack,by replication of contains. This provides data availability in the event of a network switch failure or partition within the cluster. Hadoop master daemons obtain the rack id of the cluster slaves by invoking either an external script or java class by configuration files. Using either the java class or external script for topology, output should report to the java org.apache.hadoop.net.DNSToSwitchMapping interface.

The interface expects a one-to-one correspondence to be maintained and the topology information in the format of ‘/myrack/myhost’, where ‘/’ is the topology delimiter, ‘myrack’ is the rack identifier, and ‘myhost’ is the individual host. Assuming a single /20 subnet per rack, one could use the format of ‘/111.118.100.1/111.118.100.5’ as a unique rack-host topology mapping.

lets consider, NetworkTopology.java, is included with the hadoop distribution and can be customized by the Hadoop administrator. Using a Java class instead of an external script has a performance benefit in that Hadoop doesn’t need to fork an external process when a new slave node registers itself. If implementing an external script, it will be specified with the topology.script.file.name parameter in the configuration files. Unlike the java class, the external topology script is not included with the Hadoop distribution and is provided by the administrator. Hadoop will send multiple IP addresses to ARGV when forking the topology script. The number of IP addresses sent to the topology script is controlled with net.topology.script.number.args and defaults to 100. If net.topology.script.number.args was changed to 1, a topology script would get forked for each IP submitted by DataNodes and/or NodeManagers. If topology.script.file.name or topology.node.switch.mapping.impl is not set, the rack id ‘/default-rack’ is returned for any passed IP address. While this behavior appears desirable, it can cause issues with HDFS block replication as default behavior is to write one replicated block off rack , and is unable to do so as there is only a single rack named ‘/default-rack’. An additional configuration setting is mapreduce.jobtracker.taskcache.levels which determines the number of levels (in the network topology) of caches MapReduce will use. So, for example, if it is the default value of 2, two levels of caches will be constructed - one for hosts (host -> task mapping) and, another for racks (rack -> task mapping). Giving us our one-to-one mapping of ‘/myrack/myhost’.

+----------+ +--------+

|jobtracker| |datanode|

+----------+ +--------+

\ /

+--------+ +--------+ +--------+ |datanode|--| switch |--|datanode| +--------+ +--------+ +--------+ / \ +--------+ +--------+ |datanode| |namenode| +--------+ +--------+

We are aware of the fact that hadoop divides the data into multiple file blocks and stores them on different machines. If Rack Awareness is not configured, there may be a possibility that , hadoop will place all the copies of the block in same rack which results in loss of data when that rack fails.

Configuring Rack awareness in hadoop involves two steps configure the “topology.script.file.name” in core-site.xml , topology.node.switch.mapping.impl org.apache.hadoop.net.ScriptBasedMapping The default implementation of the DNSToSwitchMapping. It invokes a script specified in topology.script.file.name to resolve node names. If the value for topology.script.file.name is not set, the default value of DEFAULT\_RACK is returned for all node names. topology.script.file.name core/rack-awareness.sh