**[How to perform capacity planning for a Hadoop cluster](http://www.bigdatatidbits.cc/2014/08/how-to-perform-capacity-planning-for.html)**

**Provisioning Hadoop machines**

Recently I had a customer ask what kind of machine to purchase to be used in a Hadoop environment, and what configuration to use. The answer to this can be essentially derived from some simple calculations that I want to write about and demonstrate.

The number of machines, and specs of the machines, depends on a few factors: the volume of data (obviously), the data retention policy (how much can you afford to keep before throwing away), the type of workload you have (data science/CPU driven vs “vanilla” use case/IO-bound), and also the data storage mechanism (data container, type of compression used if any). We have to make some assumptions from the beginning; otherwise there are just too many parameters to deal with. These assumptions drive the data nodes configuration.

The other types of machines (Name Node/Job tracker, in Hadoop 1) will need different specs, and are generally more straightforward. We’ll just talk about data nodes in this post.

**Capacity planning for your data**

            The number of machines to purchase will depend on the volume of data to store and analyze, which will drive the number of spinning disks to get on a per machine basis (usually a fixed number of hard drives/machine). The below applies mainly to Hadoop 1.x versions. We will talk about Yarn later on.  
Capacity planning usually flows from a top-down approach of understanding:  
- How many nodes you need  
- What's the capacity of each node, on the CPU side  
- What's the capacity of each node, on the memory side.

Let's do a back-of-the-envelope calculation. This is usually the first estimation you need to make when assessing what you need and budget for the machines.

**Data nodes**

The HDFS’ configuration is usually set up to replicate the data 3 ways. So you will need 3x the actual storage capacity for your data. In addition, you will need to sandbag the machine capacity for temporary storage for computation (i.e. storage for transient Map outputs stays local to the machine, it doesn’t get stored on HDFS. Also, local storage for compression is needed). A good rule of thumb is to keep the disks at 70% capacity. Then we also need to take into account the compression ratio.

Let’s take an example:

Say we have 70TB of raw data to store on a yearly basis (i.e. moving window of 1 year). So after compression (say, with Gzip) we will get 70 – (70 \* 60%) = 28Tb that will multiply by 3x = 84, but keep 70% capacity: 84Tb = x \* 70% thus x = 84/70% = 120Tb is the value we need for capacity planning.

**Number of nodes**

Here are the recommended specifications for DataNode/TaskTrackers in a balanced Hadoop cluster [from Cloudera](http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/):

12-24 1-4TB hard disks in a JBOD (Just a Bunch Of Disks) configuration (no RAID, please!)

multi-core CPUs, running at least 2-2.5GHz

So let’s divide up the value we have in capacity planning by the number of hard disks we need in a way that makes sense: 120Tb/12 1Tb = 10 nodes.

**Number of tasks per node**

First, let's figure out the # of tasks per node.

Usually count 1 core per task. If the job is not too heavy on CPU, then the number of tasks can be greater than the number of cores.

Example: 12 cores, jobs use ~75% of CPU  
Let's assign free slots= 14 (slightly > # of cores is a good rule of thumb), maxMapTasks=8, maxReduceTasks=6.

**Memory**

Now let's figure out the memory we can assign to these tasks. By default, the tasktracker and datanode take up each 1 GB of RAM per default. For each task calculate *mapred.child.java.opts* (200MB per default) of RAM. In addition, count 2 GB for the OS. So say, having 24 Gigs of memory available, 24-2= 22 Gig available for our 14 tasks – thus we can assign 1.5 Gig for each of our tasks (14 \* 1.5 = 21 Gigs).

**Yarn**

**I**n YARN, the arbitrary fixed limits put in place on the number of mappers and reducer slots assigned on each cluster node disappears:  the notion of fixed slots has been discarded, and resources are now configured in terms of amounts of memory (in megabytes) and CPU (“v-cores”).  Instead, YARN uses *yarn.nodemanager.resource.memory-mb*and *yarn.nodemanager.resource.cpu-vcores*, which control the amount of memory and CPU on each node, both available to both maps and reduces. If configuring these manually, simply set these to the amount of memory and number of *cores* on the machine after subtracting out resources needed for other services. See <http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH5/latest/CDH5-Installation-Guide/cdh5ig_mapreduce_to_yarn_migrate.html> for more details.