Hadoop Cluster planning ::

Sizing Hadoop cluster is very important as it forms the backbone of your future endeavours.

One of the first thing to decide for which version and which distribution of Hadoop cluster to install

This is decided with consultation of end users of cluster like BI systems, developers, operating system admin and database admins and users who will extract data and ingest data into cluster

Also we need to decide whether we want to install only core Hadoop implementation or other components like hive.

Which processing engine we need to use depends upon which resources we have

Cloudera distribution including apache Hadoop

Cloudera , a company which provide consultation, support and management tools for Hadoop.

Cloudera distribution of Hadoop also known as CDH is also open source.IT will have whole components of Hadoop.

It will have a stable version of Hadoop.

Hardware Selection

:: It is very important to decide what kind of hardware we need. It will depend on how much data we will be ingesting ,how many jobs will run for extraction of data etc.

No one likes to extend ram ,cpu and disk after setting up cluster of 50 to 100 nodes.

As you probably already know, one of the major advantages of Hadoop is its ability to run on so-called commodity hardware. This isn’t just a function of cost, although that certainly plays a large role. One example of this is Hadoop’s preference for JBOD1 and how its I/O patterns fit this model explicitly. This isn’t to say production Hadoop clusters commonly run on $1,000 machines—your expectations of what is meant by commodity may need adjustment—but rather that you won’t need to break the bank by purchasing top-end servers. Hadoop hardware comes in two distinct classes: masters and workers. Master nodes are typically more robust to hardware failure and run critical cluster services. Loss of a master almost certainly means some kind of service disruption. On the other hand, worker nodes are expected to fail regularly. This directly impacts the type of hardware as well as the amount of money spent on these two classes of hardware. It is common that administrators, in an effort to reduce the proliferation of hardware profiles in the data center, will select a single hardware profile for all masters and a single profile for all workers. Those with deep pockets may find it even easier to purchase a single

The distinction between a node and the service assigned to said node is important. When talking about machines, there are masters and workers. These designations reflect the class of hardware. Separately, there are the five core Hadoop services: namenode, secondary namenode, datanode, jobtracker, and tasktracker, each a separate daemon. Services are run on nodes of the cluster. Worker services such as the datanode and tasktracker always run together. In smaller clusters, it sometimes makes sense to run the master services—the namenode, secondary namenode, and jobtracker—together. As the cluster grows, these services are separated and dedicated hardware for each is provisioned. When you hear “master,” the next question is always “what process?” “Slave,” or “worker,” will always mean the datanode and tasktracker pair.

Now we have to decide what hardware we need for master node and which for worker node.

Master hardware selection :

For master nodes on which namenode,snn,resourcemanager etc run should be on enterprise level hardware. While proponents of Hadoop beat the commodity hardware drum, this is the place where people spend more money and spring for the higher-end features. Dual power supplies, bonded network interface cards (NICs), and sometimes even RAID 10 in the case of the namenode storage device, are not uncommon to find in the wild. In general, master processes tend to be RAM-hungry but low on disk space consumption. The namenode and jobtracker are also rather adept at producing logs on an active cluster, so plenty of space should be reserved on the disk or partition on which logs will be stored.

The operating system device for master nodes should be highly available. This usually means RAID-1 (a mirrored pair of disks). Since the OS does not consume a significant amount of space, RAID-10 or RAID-5 would be overkill and lead to unusable capacity. Most of the real work is done on the data devices, while the OS device usually only has to contend with logfiles in /var/log.

Small clusters—clusters with fewer than 20 worker nodes—do not require much for master nodes in terms of hardware. A solid baseline hardware profile for a cluster of this size is a dual quad-core 2.6 Ghz CPU, 24 GB of DDR3 RAM, dual 1 Gb Ethernet NICs, a SAS drive controller, and at least two SATA II drives in a JBOD configuration

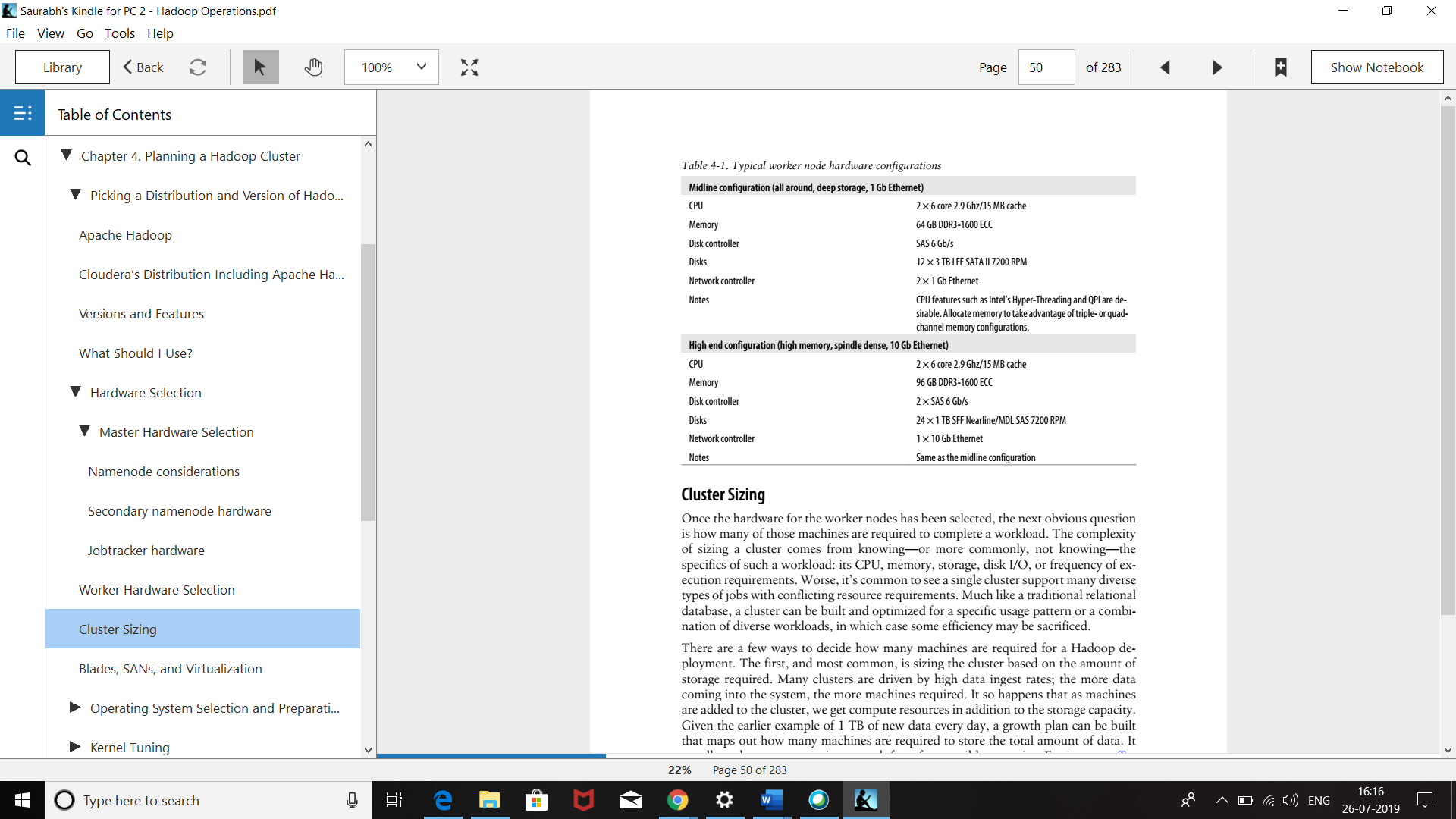
in addition to the host OS device. Clusters of up to 300 nodes fall into the mid-size category and usually benefit from an additional 24 GB of RAM for a total of 48 GB. Master nodes in large clusters should have a total of 96 GB of RAM. Remember that these are baseline numbers meant to give you a place from which to start.

Namenode considerations The namenode is absolutely critical to a Hadoop cluster and usually receives special treatment. There are three things a healthy namenode absolutely requires in order to function properly: RAM, modest but dedicated disk, and to be left alone! As we covered previously, the namenode serves all of its metadata directly from RAM. This has the obvious implication that all metadata must fit in physical memory. The exact amount of RAM required depends on how much metadata there is to maintain. Remember that the metadata contains the filename, permissions, owner and group data, list of blocks that make up each file, and current known location of each replica of each block. As you’d expect, this adds up. There are subtleties to the namenode metadata that you might not otherwise think much about. One instance of this is that the length of filenames actually starts to matter at scale; the longer the filename, the more bytes it occupies in memory. More dubious, though, is the small files problem. Each file is made up of one or more blocks and has associated metadata. The more files the namenode needs to track, the more metadata it maintains, and the more memory it requires as a result. As a base rule of thumb, the namenode consumes roughly 1 GB for every 1 million blocks. Again, this is a guideline and can easily be invalidated by the extremes. Namenode disk requirements are modest in terms of storage. Since all metadata must fit in memory, by definition, it can’t take roughly more than that on disk. Either way, the amount of disk this really requires is minimal—less than 1 TB. While namenode space requirements are minimal, reliability is paramount. When provisioning, there are two options for namenode device management: use the namenode’s ability to write data to multiple JBOD devices, or write to a RAID device. No matter what, a copy of the data should always be written to an NFS (or similar) mounted volume in addition to whichever local disk configuration is selected. This NFS mount is the final hope for recovery when the local disks catch fire or when some equally unappealing, apocalyptic event occurs.3 The storage configuration selected for production usage is usually dictated by the decision to purchase homogeneous hardware versus specially configured machines to support the master daemons. There’s no single correct answer and as mentioned earlier, what works for you depends on a great many factors.

Secondary namenode hardware The secondary namenode is almost always identical to the namenode. Not only does it require the same amount of RAM and disk, but when absolutely everything goes wrong, it winds up being the replacement hardware for the namenode. Future versions of Hadoop (which should be available by the time you read this) will support a highly available namenode (HA NN) which will use a pair of identical machines. When running a cluster with an HA namenode, the standby or inactive namenode instance performs the checkpoint work the secondary namenode normally does. Jobtracker hardware Similar to the namenode and secondary namenode, the jobtracker is also memoryhungry, although for a different reason. In order to provide job and task level-status, counters, and progress quickly, the jobtracker keeps metadata information about the last 100 (by default) jobs executed on the cluster in RAM. This, of course, can build up very quickly and for jobs with many tasks, can cause the jobtracker's JVM heap to balloon in size. There are parameters that allow an administrator to control what information is retained in memory and for how long, but it’s a trade-off; job details that are purged from the jobtracker’s memory no longer appear in its web UI. Due to the way job data is retained in memory, jobtracker memory requirements can grow independent of cluster size. Small clusters that handle many jobs, or jobs with many tasks, may require more RAM than expected. Unlike the namenode, this isn’t as easy to predict because the variation in the number of tasks from job to job can be much greater than the metadata in the namenode, from file to file.

Worker Hardware

Selection When sizing worker machines for Hadoop, there are a few points to consider. Given that each worker node in a cluster is responsible for both storage and computation, we need to ensure not only that there is enough storage capacity, but also that we have the CPU and memory to process that data. One of the core tenets of Hadoop is to enable access to all data, so it doesn’t make much sense to provision machines in such a way that prohibits processing. On the other hand, it’s important to consider the type of applications the cluster is designed to support. It’s easy to imagine use cases where the cluster’s primary function is long-term storage of extremely large datasets with infrequent processing. In these cases, an administrator may choose to deviate from the balanced CPU to memory to disk configuration to optimize for storage-dense configurations. Starting from the desired storage or processing capacity and working backward is a technique that works well for sizing machines. Consider the case where a system ingests new data at a rate of 1 TB per day. We know Hadoop will replicate this data three times by default, which means the hardware needs to accommodate 3 TB of new data every day! Each machine also needs additional disk capacity to store temporary data during



Cluster Sizing :: Once the master and worker hardware is decided now is the time for determining what would be capacity or size of Hadoop cluster.

Average daily ingest rate 1 TB Replication factor 3 (copies of each block) Daily raw consumption 3 TB Ingest × replication Node raw storage 24 TB 12 × 2 TB SATA II HDD MapReduce temp space reserve 25% For intermediate MapReduce data Node-usable raw storage 18 TB Node raw storage – MapReduce reserve 1 year (flat growth) 61 nodesa Ingest × replication × 365 / node raw storage 1 year (5% growth per monthb ) 81 nodesa 1 year (10% growth per month) 109 nodesa

we assume 12 × 2 TB hard drives per node, but we could have just as easily used half the number of drives per node and doubled the number of machines. This is how we can adjust the ratio of resources such as the number of CPU cores to hard drive spindles. This leads to the realization that we could purchase machines that are half as powerful and simply buy twice as many. The trade-off, though, is that doing so would require significantly more power, cooling, rack space, and network port density. For these reasons, it’s usually preferable to purchase reasonably dense machines without falling outside the normal boundaries of what is considered commodity hardware.

Projecting cluster size based on the completion time of specific jobs is less common, but still makes sense in certain circumstances. This tends to be more complicated and requires significantly more information than projections based solely on data size. Calculating the number of machines required to complete a job necessitates knowing, roughly, the amount of CPU, memory, and disk I/O used while performing a previous invocation of the same job. There’s a clear chicken and egg problem; a job must be run with a subset of the data in order to understand how many machines are required to run the job at scale. An interesting property of MapReduce jobs is that map tasks are almost always uniform in execution. If a single map task takes one minute to execute and consumes some amount of user and system CPU time, some amount of RAM and some amount of I/ O, 100 map tasks will simply take 100 times the resources. Reduce tasks, on the other hand, don’t have this property. The number of reducers is defined by the developer rather than being based on the size of the data, so it’s possible to create a situation where the job bottlenecks on the number of reducers or an uneven distribution of data between the reducers. The latter problem is referred to as reducer skew and is covered in greater detail in

Operating System Selection and Preparation While most of Hadoop is written in Java, enough native code and Linux-isms are in its surrounding infrastructure to make Linux the only production-quality option today. A significant number of production clusters run on RedHat Enterprise Linux or its freely available sister, CentOS. Ubuntu, SuSE Enterprise Linux, and Debian deployments also exist in production and work perfectly well. Your choice of operating system may be influenced by administration tools, hardware support, or commercial software support; the best choice is usually to minimize the variables and reduce risk by picking the distribution with which you’re most comfortable. Preparing the OS for Hadoop requires a number of steps, and repeating them on a large number of machines is both time-consuming and error-prone. For this reason, it is strongly advised that a software configuration management system be used. Puppet and Chef are two open source tools that fit the bill. Extolling the virtues of these tools is beyond the scope of what can be accomplished in this section, but there’s a breadth of documentation for both to get you going. No matter what, find a configuration management suite that makes sense to you and get familiar with it. It will save you hours (or more) of tinkering and debugging down the road. Deployment Layout Hadoop uses a number of directories on the host filesystem. It’s important to understand what each location is for and what the growth patterns are. Some directories, for instance, are used for long-term block storage of HDFS data, and others contain temporary data while MapReduce jobs are running. Each of these directories has different

Hadoop home This is the directory in which the Hadoop software is installed. Despite the name, it is commonly not installed in a user’s home directory. This directory can be made to be read only when configured correctly and usually lives in /usr/local, /opt, or /usr when Hadoop is installed via packages. Datanode data directories One or more of these directories are used by the datanode to store HDFS block data. The datanode assumes that each directory provided is a separate physical device with independent spindles and round-robin blocks between disks. These directories occupy the vast majority of disk space and act as the long-term storage for data, and they are often put on the same devices as the tasktracker MapReduce local directories. Namenode directories One or more of these directories are used by the namenode to store filesystem metadata. The namenode assumes that each directory provided is a separate physical device and replicates all writes to each device synchronously to ensure data availability in the event of disk failure. These directories will all require the same amount of space and generally do not use more than 100 GB. One of these directories is usually an NFS mount, so data is written off the physical machine. MapReduce local directories One or more directories used by the tasktracker to store temporary data during a MapReduce job. More spindles usually means better performance as MapReduce tasks interfere with one another to a lesser degree. These directories store a moderate amount, depending on what the MapReduce job is doing, and are often put on the same devices as the datanode data directories. Hadoop log directory This is a common directory used by all daemons to store log data as well as joband task-level data. It’s normal for Hadoop to generate log data proportional to cluster usage; more MapReduce jobs means more logs. Hadoop pid directory This is a directory used by all daemons to store pid files. This data is very small and doesn’t grow. Hadoop temp directory Hadoop uses a temp directory for small, short-lived files it sometimes needs to create. The temp directory is most notably used on the machines from which MapReduce jobs are submitted and contains a copy of the JAR file that ultimately gets sent to the jobtracker. This is /tmp/hadoop-<${user.name}> by default and many

**Softwares:**

**JAVA JDK and JRE**

**Cron – for scheduling jobs ,cleaning up temp files, compression and running configuration management tools**

**Ntp—to synchronize clock between various servers in cluster**

**Ssh –secure shell –very important for server to server secure communication**

**Postfic/sendmail – to send mails to admins for various alerts**

**Rysnc -- One of the most underrated tools, rsync allows administrators to copy files efficiently locally and between hosts. If you’re not already familiar with rsync, learn it.**

**Hostnames, DNS, and Identification Let’s just get this out of the way: when it comes to host identification, discovery, and the treatment of hostnames, Hadoop is complicated and extremely picky. This topic is responsible for a fair number of cries for support on the mailing lists and almost certainly an equal amount of lost sleep on the part of many who are new to Hadoop. But before we get into the list of things that can go wrong, let’s first talk about how Hadoop actually discovers and identifies hosts. As we discussed previously, Hadoop worker processes such as the tasktracker and datanodes heartbeat into the jobtracker and namenode (respectively) every few seconds. The first time this occurs, Hadoop learns about the worker’s existence. Part of this heartbeat includes the identity of the machine, either by hostname or by IP address. This identifier—again, either the hostname or the IP address—is how Hadoop will refer to this machine. This means that when an HDFS client, for instance, asks the namenode to open a file, the namenode will return this identifier to the client as the proper way in which to contact the worker. The exact implications of this are far-reaching; both the client and the worker now must be able to directly communicate, but the client must also be able to resolve the hostname and communicate with the worker using the identifier as it was reported to the namenode. But what name does the datanode report to the namenode? That’s the real question.**