# **Hardware Provisioning**

A common question received by Spark developers is how to configure hardware for it. While the right hardware will depend on the situation, we make the following recommendations.

# **Storage Systems**

Because most Spark jobs will likely have to read input data from an external storage system (e.g. the Hadoop File System, or HBase), it is important to place it ****as close to this system as possible****. We recommend the following:

If at all possible, run Spark on the same nodes as HDFS. The simplest way is to set up a Spark [standalone mode cluster](http://spark.apache.org/docs/latest/spark-standalone.html) on the same nodes, and configure Spark and Hadoop’s memory and CPU usage to avoid interference (for Hadoop, the relevant options aremapred.child.java.opts for the per-task memory and mapred.tasktracker.map.tasks.maximum andmapred.tasktracker.reduce.tasks.maximum for number of tasks). Alternatively, you can run Hadoop and Spark on a common cluster manager like [Mesos](http://spark.apache.org/docs/latest/running-on-mesos.html) or [Hadoop YARN](http://spark.apache.org/docs/latest/running-on-yarn.html).

If this is not possible, run Spark on different nodes in the same local-area network as HDFS.

For low-latency data stores like HBase, it may be preferrable to run computing jobs on different nodes than the storage system to avoid interference.

# **Local Disks**

While Spark can perform a lot of its computation in memory, it still uses local disks to store data that doesn’t fit in RAM, as well as to preserve intermediate output between stages. We recommend having ****4-8 disks**** per node, configured *without* RAID (just as separate mount points). In Linux, mount the disks with the [noatime option](http://www.centos.org/docs/5/html/Global_File_System/s2-manage-mountnoatime.html) to reduce unnecessary writes. In Spark, [configure](http://spark.apache.org/docs/latest/configuration.html) the spark.local.dir variable to be a comma-separated list of the local disks. If you are running HDFS, it’s fine to use the same disks as HDFS.

# **Memory**

In general, Spark can run well with anywhere from ****8 GB to hundreds of gigabytes**** of memory per machine. In all cases, we recommend allocating only at most 75% of the memory for Spark; leave the rest for the operating system and buffer cache.

How much memory you will need will depend on your application. To determine how much your application uses for a certain dataset size, load part of your dataset in a Spark RDD and use the Storage tab of Spark’s monitoring UI (http://<driver-node>:4040) to see its size in memory. Note that memory usage is greatly affected by storage level and serialization format – see the [tuning guide](http://spark.apache.org/docs/latest/tuning.html) for tips on how to reduce it.

Finally, note that the Java VM does not always behave well with more than 200 GB of RAM. If you purchase machines with more RAM than this, you can run *multiple worker JVMs per node*. In Spark’s [standalone mode](http://spark.apache.org/docs/latest/spark-standalone.html), you can set the number of workers per node with theSPARK\_WORKER\_INSTANCES variable in conf/spark-env.sh, and the number of cores per worker with SPARK\_WORKER\_CORES.

# **Network**

In our experience, when the data is in memory, a lot of Spark applications are network-bound. Using a ****10 Gigabit**** or higher network is the best way to make these applications faster. This is especially true for “distributed reduce” applications such as group-bys, reduce-bys, and SQL joins. In any given application, you can see how much data Spark shuffles across the network from the application’s monitoring UI (http://<driver-node>:4040).

# **CPU Cores**

Spark scales well to tens of CPU cores per machine because it performes minimal sharing between threads. You should likely provision at least ****8-16 cores**** per machine. Depending on the CPU cost of your workload, you may also need more: once data is in memory, most applications are either CPU- or network-bound.