Welcome to Hadoop Fundamentals, Hadoop Components.

In this unit I will discuss the MapReduce Philosophy, describe the usage of Pig, Hive, and Jaql, talk about how to get data into Hadoop through the use of Flume and Sqoop and finally describe how to schedule and control job execution using Oozie.

First I need to set the boundaries for this unit. The components presented in this unit are done so at a very high level. The Hadoop environment is littered with a number of open source components with funny sounding names. And to some people, it is difficult to understand their usage. This unit is merely an attempt to provide you with descriptions of some of those components. If you are interested in getting more detail about each of the components covered in this unit, then I would direct you to the other Big Data University courses that are specific to these components.

Let us take a look at MapReduce. MapReduce is designed to process very large datasets for certain types of distributable problems. It attempts to spread the work across a large number of nodes and allows those nodes to process the data in parallel. You cannot have dependencies within the data, meaning that you cannot have a requirement that one record in a dataset must be processed before another.

Results from the initial parallel processing are sent to additional nodes where the data is combined to allow for further reductions of the data.

Now let's take a look at how the map and reduce operations work in sequence on your data to produce the final output. In this case, we will have a job with a single map step and a single reduce step. The first step is the map step. It takes a subset of the full dataset called an input split and applies to each row in the input split an operation that you have written, such as parsing each character string.

The output data is buffered in memory and spills to disk. It is sorted and partitioned by key using the default partitioner. A merge sort sorts each partition.

There may be multiple map operations running in parallel with each other, each one processing a different input split.

The partitions are shuffled among the reducers. For example, partition 1 goes to reducer 1. The second map task also sends its partition 1 to reducer 1.

Partition 2 goes to another reducer.
Additional map tasks would act in the same way.
Each reducer does its own merge steps and executes the code of your reduce task.
For example, it could do a sum on the number of occurrences of a particular character string.
This produces sorted output at each reducer.
The data that flows into and out of the mappers and reducers takes a specific form.
Data enters Hadoop in unstructured form but before it gets to the first mapper, Hadoop has changed it into key-value pairs with Hadoop supplying its own key.
The mapper produces a list of key value pairs. Both the key and the value may change from the k1 and v1 that came in to a k2 and v2. There can now be duplicate keys coming out of the mappers. The shuffle step will take care of grouping them together.
The output of the shuffle is the input to the reducer step. Now, we still have a list of the v2's that came out of the mapper step, but they are grouped by their keys and there is no longer more than one record with the same key.
Finally, coming out of the reducer is, potentially, an entirely new key and value, k3 and v3. For example, if your reducer summed the values associated with each k2, your k3 would be equal to k2 and your v3 would be the sum of the list of v2s.
Let us look at an example of a simple data flow. Say we want to transform the input on the left to the output on the right. On the left, we just have letters. On the right, we have counts of the number of occurrences of each letter in the input.
Hadoop does the first step for us. It turns the input data into key-value pairs and supplies its own key: an increasing sequence number.
The function we write for the mapper needs to take these key-value pairs and produce something that the reduce step can use to count occurrences. The simplest solution is make each letter a key and make every value a 1.
The shuffle groups records having the same key together, so we see B now has two values, both 1, associated with it.

The reduce is simple: it just sums the values it is given to produce a sum for each key.

This lesson is continued in the next video.