Transcript name: MapReduce – Part 1

English
Welcome to the unit of Hadoop Fundamentals on MapReduce. In this unit, I will first explain what map and reduce operations are, then take you through what happens when you submit a MapReduce job to Hadoop. We will look at "the shuffle" that connects the output of each mapper to the input of a reducer. This will take us into the fundamental datatypes used by Hadoop and see an example data flow. Finally, we will examine Hadoop MapReduce fault tolerance, scheduling, and task execution optimizations. To understand MapReduce, we need to break it into its component operations map and reduce. Both of these operations come from functional programming languages. These are languages that let you pass functions as arguments to other functions. We'll start with an example using a traditional for loop. Say we want to double every element in an array. We would write code like that shown. The variable "a" enters the for loop as [1,2,3] and comes out as [2,4,6]. Each array element is mapped to a new value that is double the old value. The body of the for loop, which does the doubling, can be written as a function. We now say a[i] is the result of applying the function fn to a[i]. We define fn as a function that returns its argument multiplied by 2. This will allow us to generalize this code. Instead of only being able to use this code to double numbers, we could use it for any kind of map operation. We will call this function "map" and pass the function fn as an argument to map. We now have a general function named map and can pass our "multiply by 2" function as an argument. Writing the function definition in one statement is a common idiom in functional programming languages. In summary, we can rewrite a for loop as a map operation taking a function as an argument. Other than saving two lines of code, why is it useful to rewrite our code this way? Let's say that instead of looping over an array of three elements, we want to process a dataset with billions of elements and take advantage of a thousand computers running in parallel to quickly process those billions of elements. If we decided to add this parallelism to the original program, we would need to rewrite the whole program. But if we wanted to parallelize the program written as a call to map, we wouldn't need to change our program at all. We would just use a parallel

implementation of map.

Reduce is similar. Say you want to sum all the elements of an array. You could write a for loop that iterates over the array and adds each element to a single variable named sum. But we can we generalize this.

The body of the for loop takes the current sum and the current element of the array and adds them to produce a new sum. Let's replace this with a function that does the same thing.

We can replace the body of the for loop with an assignment of the output of a function fn to s. The fn function takes the sum s and the current array element a[i] as its arguments. The implementation of fn is a function that returns the sum of its two arguments.

We can now rewrite the sum function so that the function fn is passed in as an argument.

This generalizes our sum function into a reduce function. We will also let the initial value for the sum variable be passed in as an argument.

We can now call the function reduce whenever we need to combine the values of an array in some way, whether it is a sum, or a concatenation, or some other type of operation we wish to apply. Again, the advantage is that, should we wish to handle large amounts of data and parallelize this code, we do not need to change our program, we simply replace the implementation of the reduce function with a more sophisticated implementation. This is what Hadoop MapReduce is. It is a implementation of map and reduce that is parallel, distributed, fault-tolerant and network topology-aware. This lets you efficiently run map and reduce operations over large amounts of data.

This lesson is continued in the next video.