# MapReduce

A MapReduce job proceeds in two phases, called the map phase and the reduce phase. Each phase is simply a function that transforms the input to output. The inputs and outputs for each phase are a set of key-value pairs. MapReduce is typed, so the input key-value pairs can be of different types than the outputs. However, the map output and the reduce input must have the same types for keys and values.

E.g.

Here's a sample line from a 1-gram file used as our example dataset, showing the number of occurrences (20) for the word 'dobbs' in books from 2007: Each file has one n-gram per line containing the n-gram, the year, the number of occurrences in that year, the number of books it appeared in in that year, and the number of pages it appeared on in that year.

dobbs 2007  20    18    15

MapReduce is ideally suited for text processing, so let's use it to find the total count for each word in the dataset. For this, we need to use the 1-grams, and aggregate over the year field. Our input comprises lines of text, which in Hadoop are usually represented as key-value pairs where the key is the offset of the line within the file and the value is the line of text. Here's a small sample:

(0,  "dobbs 2007  20    18    15")

(20, "dobbs 2008  22    20    12")

(40, "doctor      2007  545525      366136      57313")

(72, "doctor      2008  668666      446034      72694")

Map Part

We care only about the word and the number of occurrences, so we ignore the other fields, and the key. The map emits the word as the key and the number of occurrences as the value:

("dobbs", 20)

("dobbs", 22)

("doctor", 545525)

("doctor", 668666)

This data is then fed into the reduce function. However, the MapReduce framework performs an important transformation before it does this: The values for a given key are brought together so that the reduce function processes them as a group. In this example, the input to the reduce function would look like this:

("dobbs", [20, 22])

("doctor", [545525, 668666])

The reduce function iterates over the values for each key and does whatever processing it likes. Here we want an aggregate count, so we simple sum the values:

("dobbs", 42)

("doctor", 1214191)

The final result is the number of times each word occurred in the whole dataset.



