How Map/Reduce works

Excerpted from R in a Nutshell, 2nd edition, Joseph Adler, O'Reilly 2012.

To help explain how Map/Reduce works, let's consider three problems:

Creating a web traffic report

> Suppose that you want to calculate the number of requests and bytes served by a set of web servers each hour. You are given a set of large log files from web servers in common log format. Each line in the file contains seven fields: the host name or IP address of the remote host, the remote logname of the user, the username of an authenticated user, the date and time of the request, the request itself, the HTTP status returned to the client, and the number of bytes served.

Reporting on web traffic by location

> Suppose that you want to calculate the number of requests and bytes served by a set of web servers by location. Just like above, suppose that you are given a set of large log files from web servers in common log format. But in this case, you're also given a database for mapping IP addresses to locations. The database will contain an entry for the first 3 bytes of every IP address, mapping that to a location.

Predicting user behavior

> Now, suppose that you want to predict how likely a user is to purchase an item from a website. Suppose that you have already computed (maybe using Map/Reduce), a set of variables describing each user: most common locations, the number of pages viewed, the number of purchases made in the past. Based on your experience, you'd like to calculate this forecast using random forests.

You can solve each of these problems efficiently with Map/Reduce. By implementing your solution with Map/Reduce, you can process your data in parallel on many servers, speeding up the computation immensely. Map/Reduce algorithms proceed in two steps:

How Map/Reduce works

Map step

> In the map step, tasks read in input data as a set of records, process the records, and send groups of similar records to reducers. In Hadoop terminology, the *mapper* extracts a *key*from each input record. Hadoop will then route all records with the same key to the same *reducer*.

Reduce step

> In the reduce step, tasks read in a set of related records, process the records, and outputs the results. In Hadoop terminology, the reducer will iterate through all results for the same key, processing the data and writing out the results.

Map/Reduce sounds very limiting, but it is a very flexible system for processing data. Programs can do many different things in the map step: they can drop fields or whole records, they can write out more than one output record for each input record, they can transform the input however the user would like. Similarly, programs can do many different thing in the reduce step. They can count input records, add fields from a set of records, combine records of different types, or do anything else the user would like to do with sets of records.

The great strength of Map/Reduce is that both maps and reducers are easily parallelizable. Each mapper processes a set of input records. Map tasks do not maintain state, nor do they need to communicate with one another. So it is possible to divide the mapping work across many map processes on many machines. Similarly, reduce tasks do not maintain state between keys, nor do they need to communicate with one another. Each reduce task will process sets of records (corresponding to sets of keys). Records with different keys are consistently sent to the same reduce task. So it is possible to efficiently process massive data sets with Map/Reduce.

To help show how this works, let’s explain how to use map/reduce to solve each of the examples above.

How Map/Reduce works

Creating a web traffic report

Each mapper will read in a web server log (or a set of logs). The mapper will process the logs one line at a time. For each record, the mapper will extract the timestamp and round the time to the nearest hour; the mapper will use this as the key. The mapper will then extract the number of bytes, which will be used as the value. The mapper will write out these fields.

The reducer will iterate over a set of byte counts for each key. The reducer will count the number of records and sum the number of bytes for each key, and then write out the results.

Reporting on web traffic by location

To solve this problem, we'll need our mappers to do two things. First, some mappers will read web server logs. They will look at the IP address in each entry, extracting the first 3 bytes to use as a key. They will then extract the number of bytes and use that as the value and send these entries to the reducers.

Next, some mappers will process the IP geolocation data. They will extract the first 3 bytes of an IP address as a key and the location as the value. They will send these results to the reducers.

The reducers will then collect two things. When they are processing input records, they will do two things. When they encounter a location record, they will record the location. When they encounter a traffic record, they will add the request to the total requests and add the number of bytes to the total bytes. When the reducer finishes iterating through the entries, it will then write out the location, the number of bytes, and the number of requests.

Predicting user behavior

As you may recall, random forests work by calculating a set of regression trees and then averaging them together to create a single model. It can be time consuming to fit the random trees to the data, but each new tree can be calculated independently.

There are many ways to accomplish this task. One way to tackle this problem is to use a set of map tasks to generate random trees, and then send the models to a single reducer task to average the results and produce the model.