6 824

wu

September 12, 2022

Contents

1 map reduce

1.1 programming model

the computation takes a set of **input** key/value pairs, and produces a set of **output** key/value pairs. The user of the MapReduce library expresses the computation as two functions: **Map** and **Reduce**

Map, written by the user, takes an input pair and produces a set of **intermediate** key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the **Reduce** function

The **Reduce** function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation.

1.1.1 example

```
map(String key, String value):
  // key: document name
```

```
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1")

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    results += ParseInt(v)
  Emit(AsString(result))
```

The `map` function emits each word plus an associated count of occurrences. The `reduce` function sums together all counts emitted for a particular word

1.1.2 Types

map	`(k1,v1)`	$\rightarrow$ `list(k2,v2)`
reduce	`(k2,list(v2))`	$\rightarrow$ `list(v2)`

1.1.3 More examples

Distributed Grep: the map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output

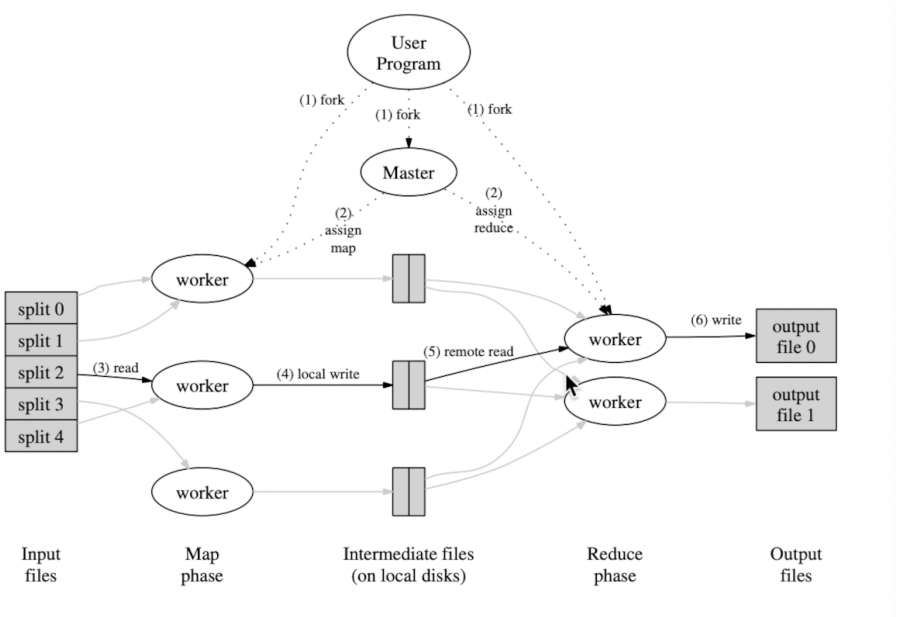
Count of URL Access Frequency: the map function processes logs of web page requests an outputs $\langle$`URL,1`$\rangle$. The reduce function adds together all values for the same URL and emits a $\langle$`URL,total count`$\rangle$ pair

Term-vector per Host: A term vector summarizes the most important words that occur in a document or a set of documents as a list of $\langle word, frequency \rangle$ pairs. The map function emits a $\langle$`hostname,term vector`$\rangle$ pair for each input document. The reduce function is passed all per-document term vectors for a given host. It add these term vectors together, throwing away infrequent terms, and then emits a final $\langle$`hostname,term vector`$\rangle$ pair

1.2 Implementation

1.2.1 Execution Overview

The *Map* invocations are distributed across multiple machines by automatically partitioning the input data into a set of M *splits*. The input splits can be processed in parallel by different machines. *Reduce* invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g., $hash(key) \mod R$). The number of partitions and the partitioning function are specified by the user



When the user program calls the `MapReduce` function, the following sequence of actions occurs

1. the MapReduce library in the user program first splits the input files into M pieces and starts up many copies of the program on a cluster of machines

2. one of the copies of the program is special - the master. The rest are workers that are assigned work by the master. there are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task

3.