MapReduce: Simplified Data Processing on Large Clusters

유연일 민철기

Introduction

- MapReduce is a programming model and an associated implementation for processing and generating large data set with parallel, distributed algorithm on cluster, introduced by Google at 2004.
- MapReduce designed to process large data by Massively Parallel Processing(MPP) with shared-nothing multi-pc-nodes.

Introduction - Related Works

- Hadoop
  - Apache Hadoop is an open-source software framework which Hadoop's MapReduce and HDFS components were inspired by Google's MapReduce and Google File System.
- Spark
  - Apache Spark was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs. Spark's resilient distributed dataset(RDD) data structure functions as a working set for distributed programs that offers a restricted form of distributed shared memory
- Cloud Dataflow
  - Google Cloud Dataflow aims to address the performance issues of MapReduce. Holzle, Google's senior VP of Technical infrastructure, stated that MapReduce performance started to sharply decline when handling multi-petabyte datasets, and Cloud Dataflow apparently offers much better performance on large datasets that Google has largely replaced MapReduce to Cloud Dataflow

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

Programming Model

- Map
  - Map 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 and passes them to the Reduce Function

Map:(key1, value1) → (key2, value2)

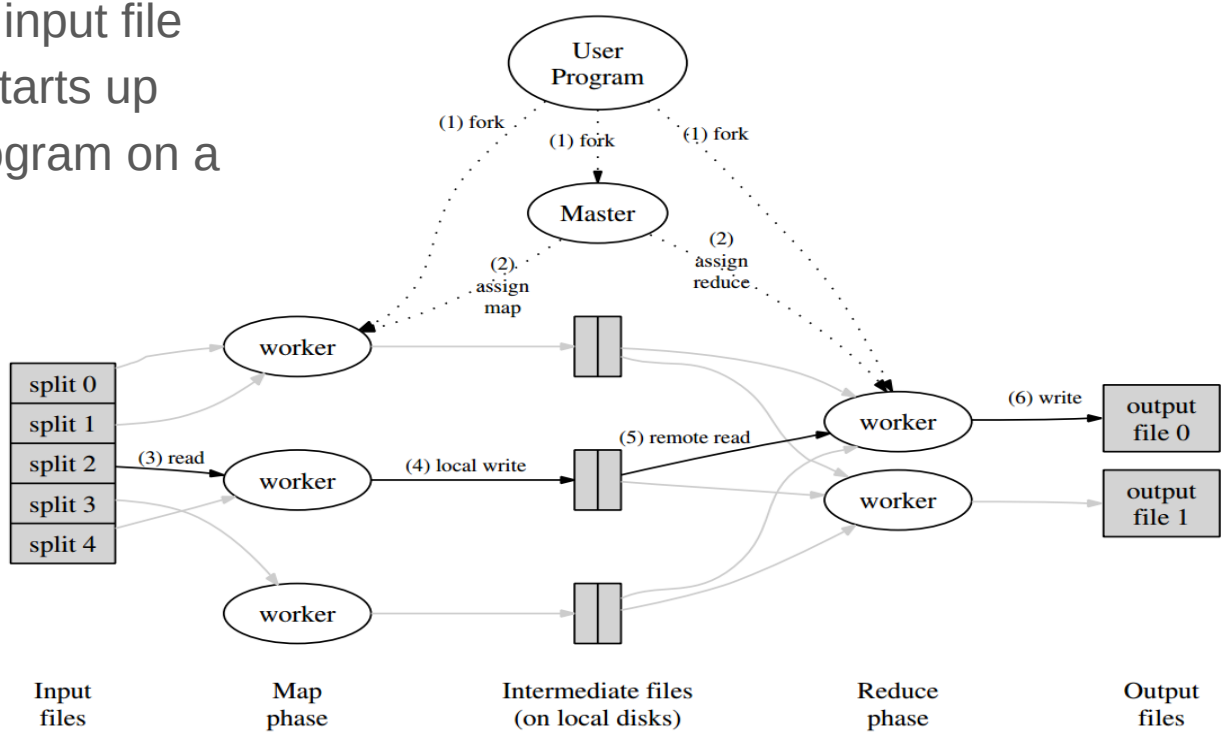
Programming Model

- Reduce
  - Reduce function accepts an intermediate key and a set of values for that key. It merges together those values to form a possibly smaller set of values. The intermediate values are supplied to user's reduce function via an iterator incase to handle lists of values that are too large to fit in memory.

Reduce:(key2, List of value2) → (key3, value3)

Implementation

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

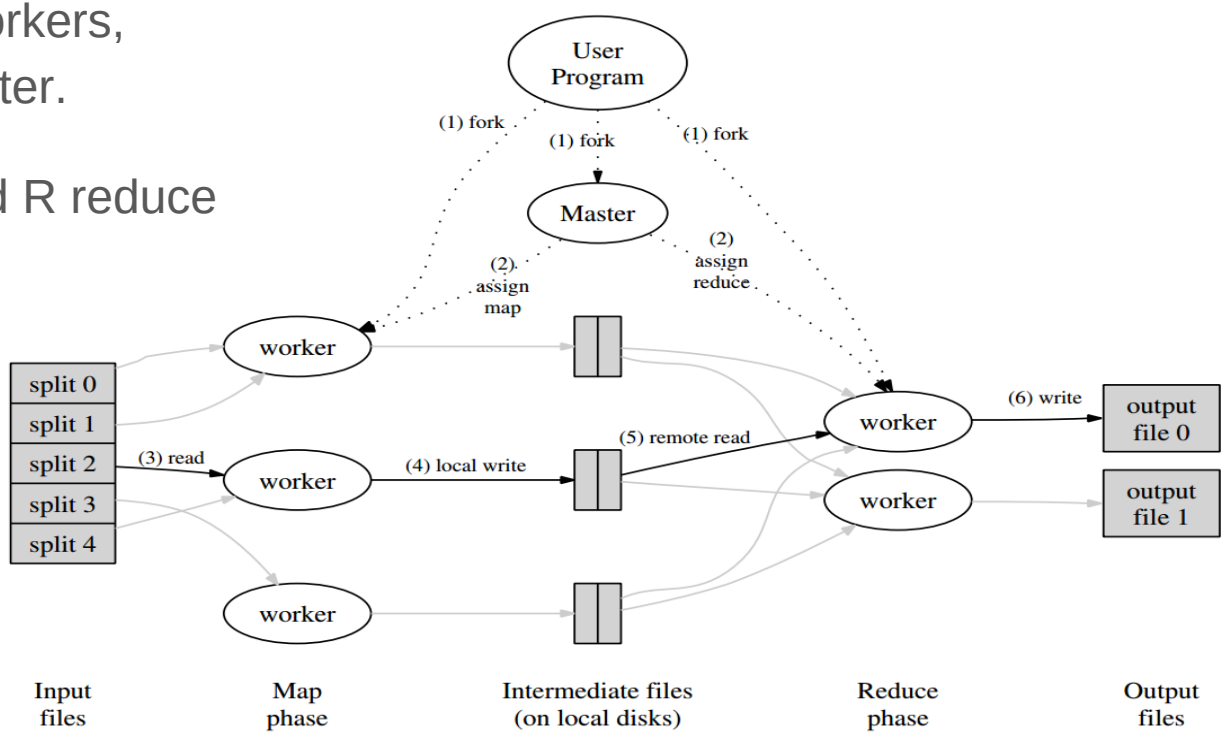


Implementation

2. One of copies of the program is master, and the rest are workers, which assigned by the master.

There are M map tasks and R reduce tasks to assign.

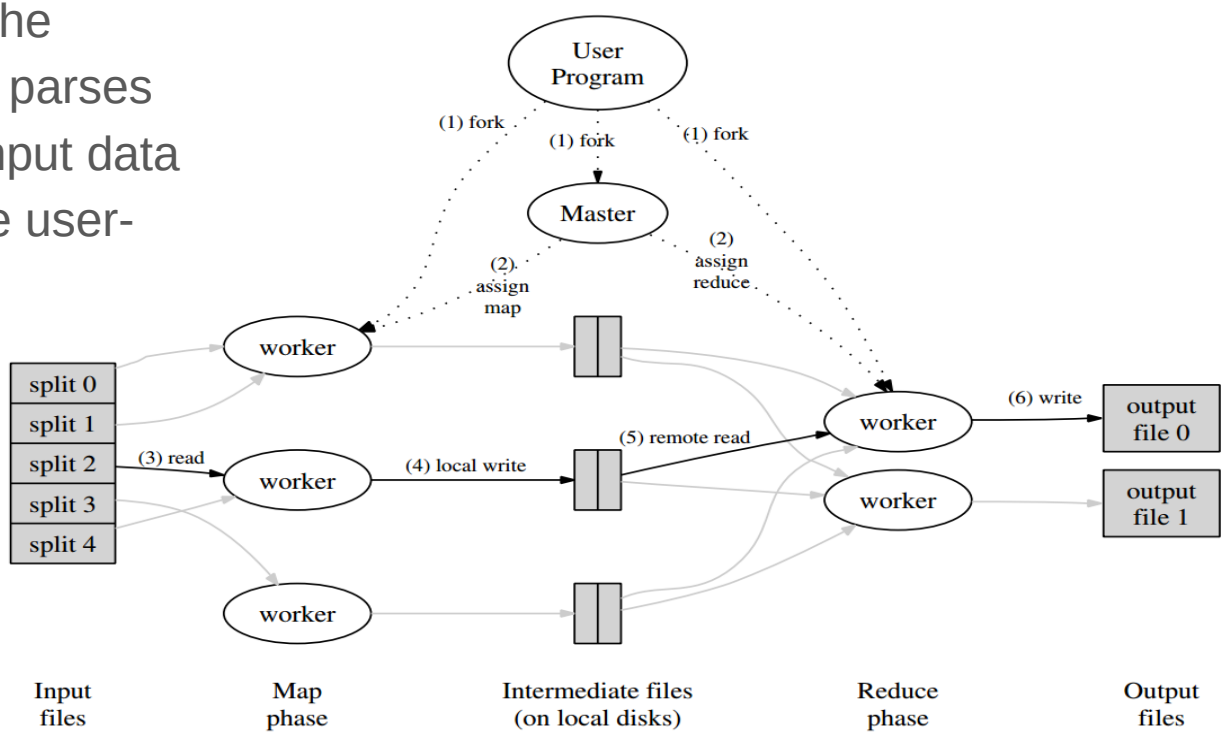
Master picks idle workers and assigns each one a map task or a reduce tasks.



Implementation

3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function.

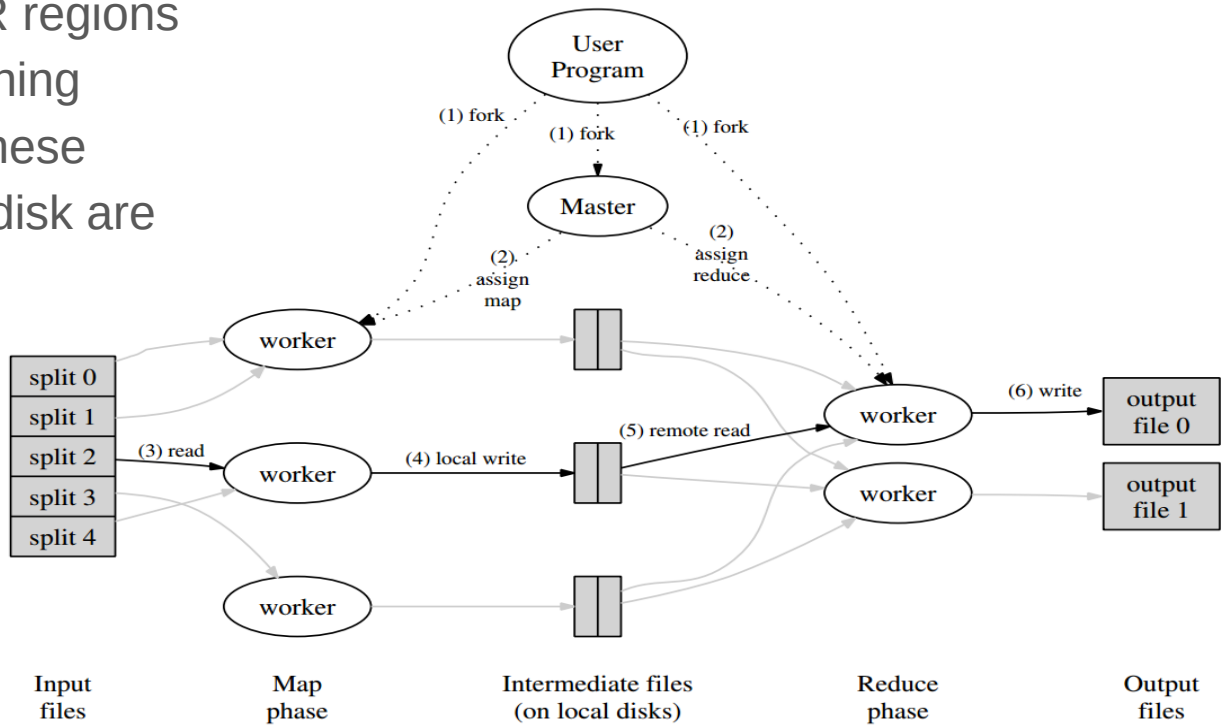
The intermediate key/value pairs produced by the Map function are buffered in memory.



Implementation

4. The buffered pairs are written to local disk, partitioned into R regions by the user-defined partitioning function. The locations of these buffered pairs on the local disk are passed back to the master.

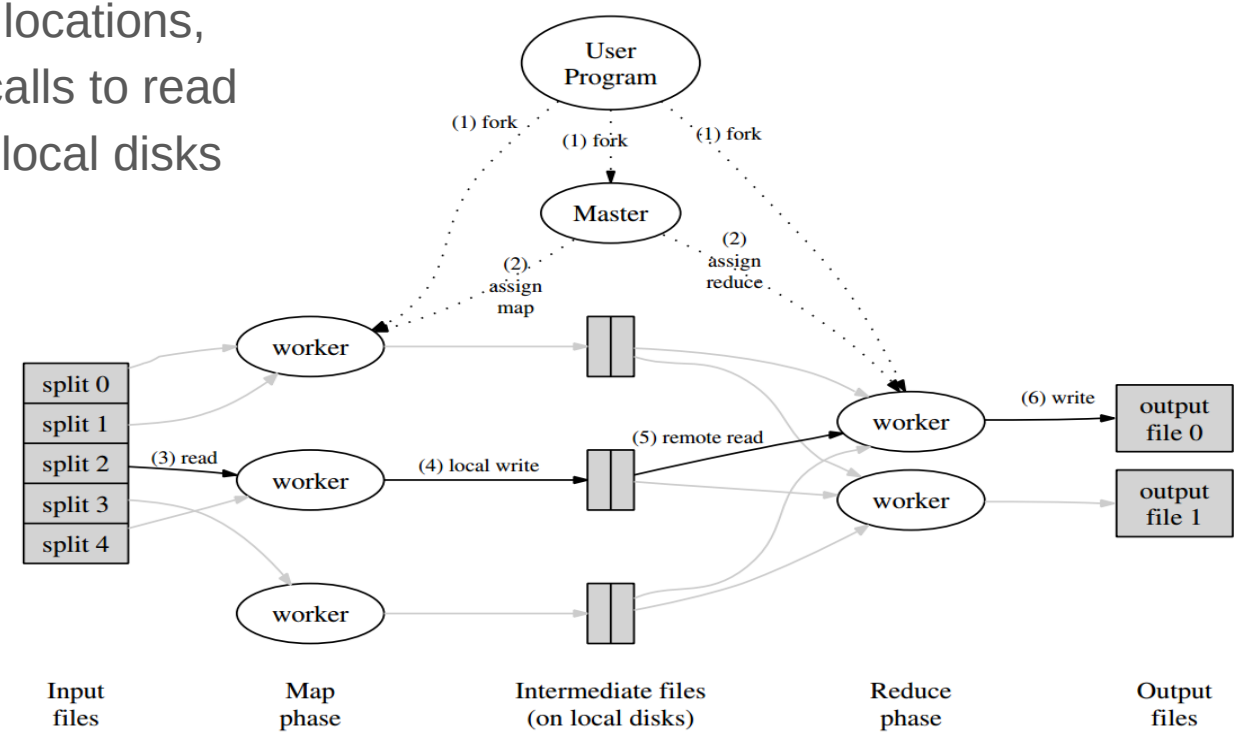
Master is responsible for forwarding these locations to the reduce workers.



Implementation

5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers.

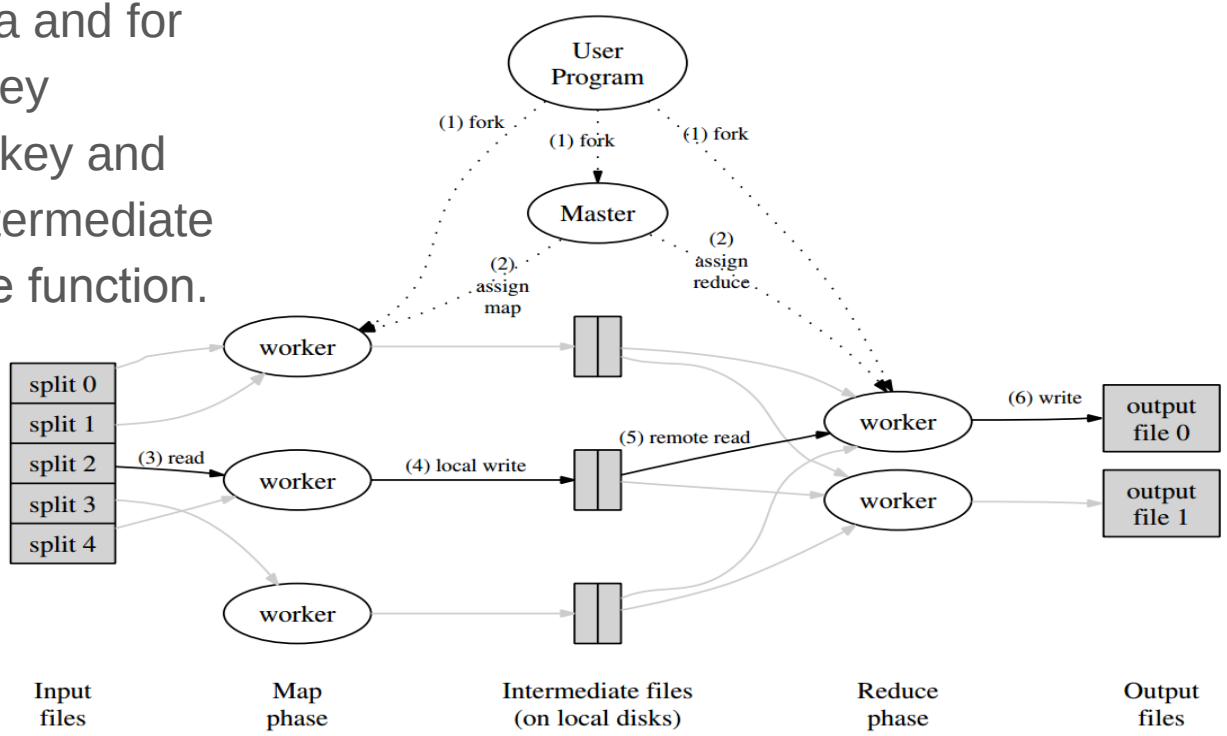
Reduce worker then sorts it by the intermediate keys



Implementation

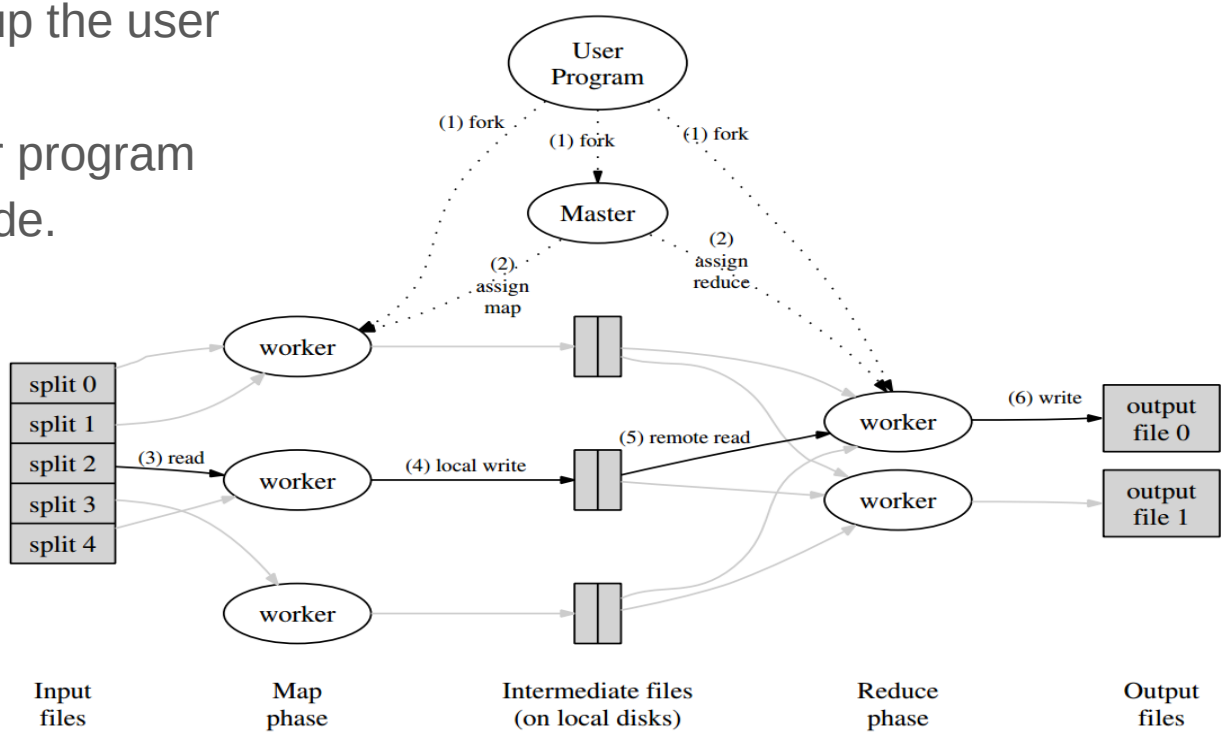
6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function.

The output of the reduce function is appended to a final output file for this reduce partition



Implementation

7. When all map tasks have been completed, master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

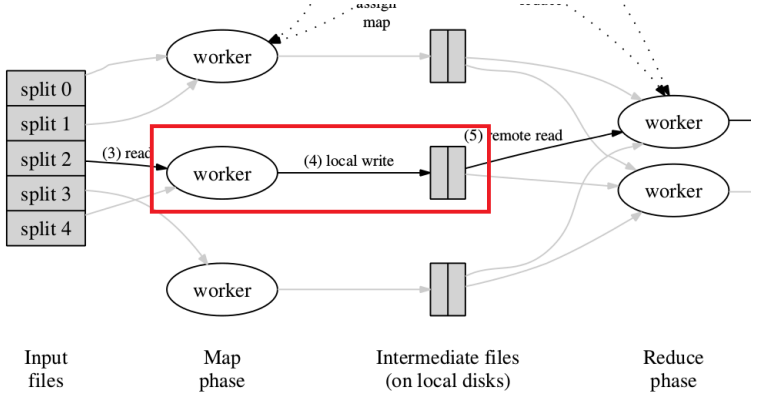


Implementation and Refinements

- Fault Tolerance
- Locality
- Task Granularity
- Backup Tasks
- Skipping Bad Records

...

Fault Tolerance



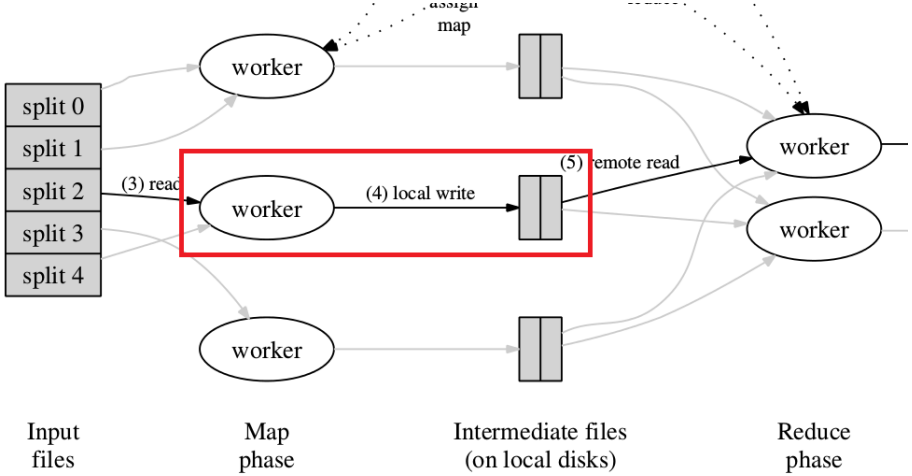
Worker Failure

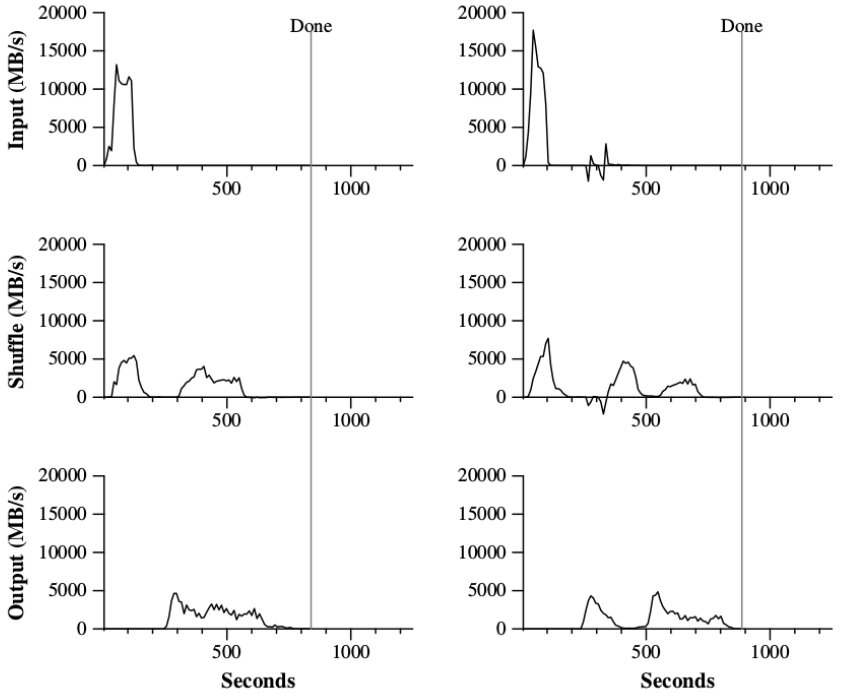
- The master pings every worker periodically.
- In-progress map / reduce tasks are re-executed on a failure.
- Completed map tasks are re-executed on a failure. (Because their output is stored on the local disk of the failed machine.)

Master Failure

- The master can write periodic checkpoints and a new copy can be started from them.
- But there is only a single master, its failure is unlikely.

Fault Tolerance





(a) Normal execution

(c) 200 tasks killed

Locality

The master takes the location information of the input files.

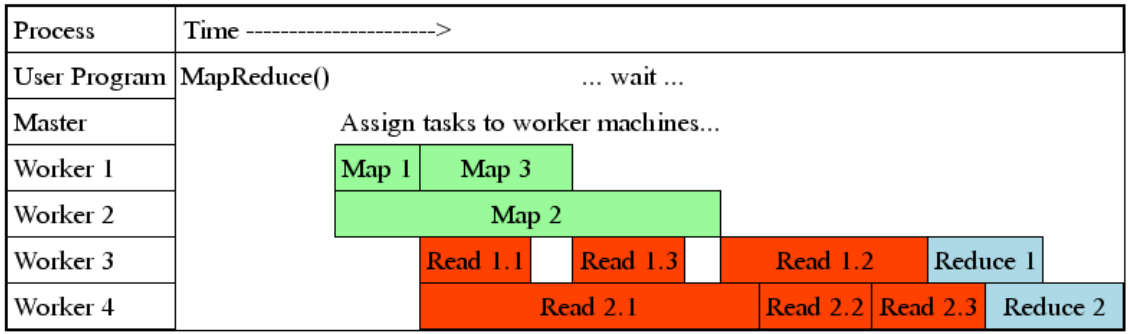
It attempts to schedule a map task on a machine that contains a replica of the corresponding input data.

Advantage

- Machines can read input at local disk speed.
- consumes no network bandwidth.

Task Granularity

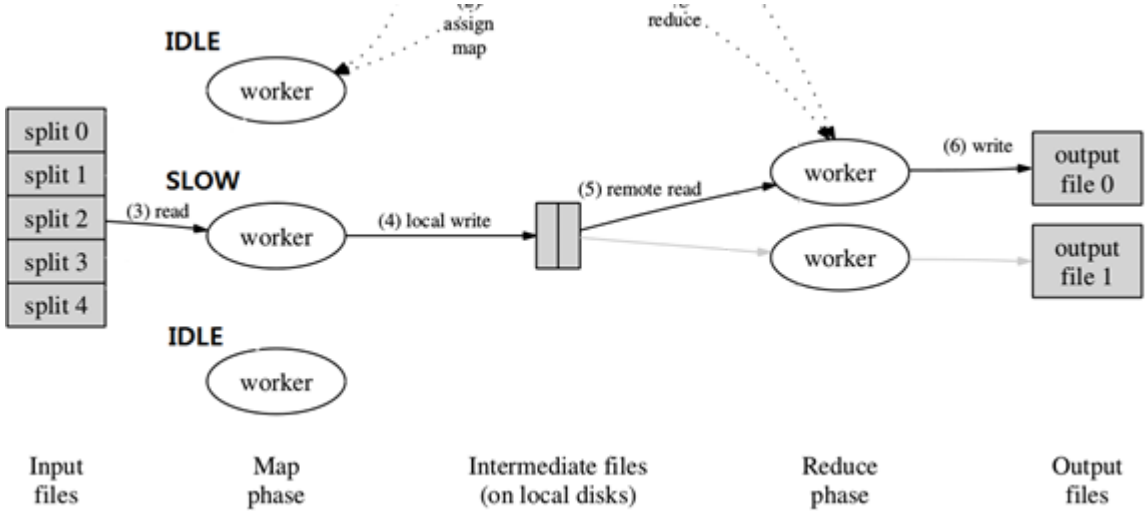
Ideally, number of map / reduce tasks should be much larger than number of workers (fine granularity tasks)



Advantage

- Minimizes fault recovery time.
- Improves dynamic load balancing.
- And pipeline...

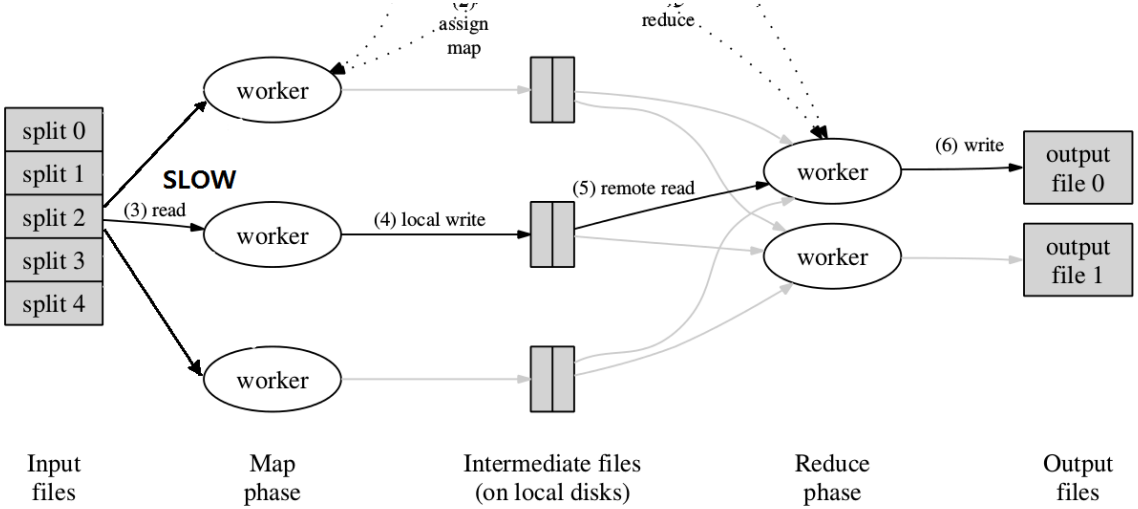
Backup Tasks



Slow workers(bad disks, bugs …) lengthen completion time.

Copies of tasks and redundant execution near of end phase.

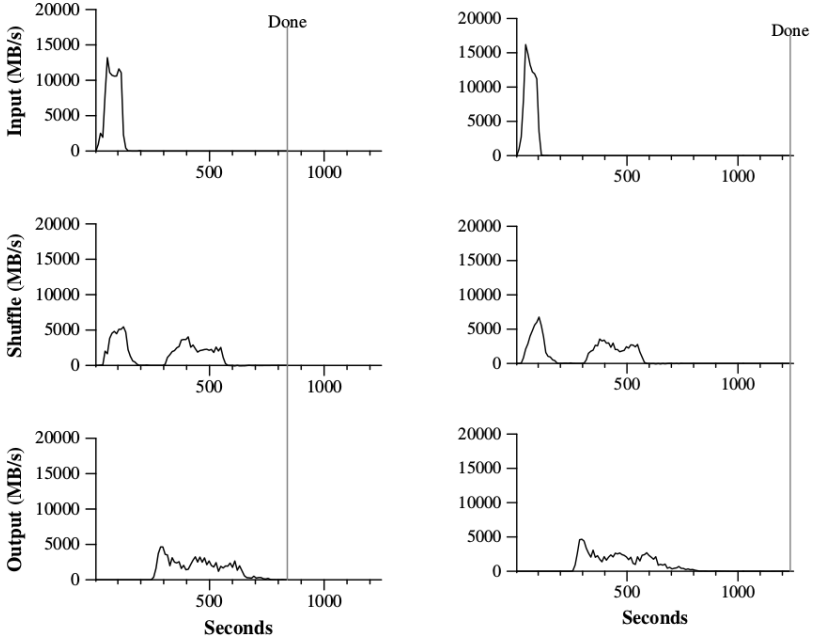
Backup Tasks



Slow workers(bad disks, bugs …) lengthen completion time.

Copies of tasks and redundant execution near of end phase.

Backup Tasks



(a) Normal execution

(b) No backup tasks

Skipping Bad Records

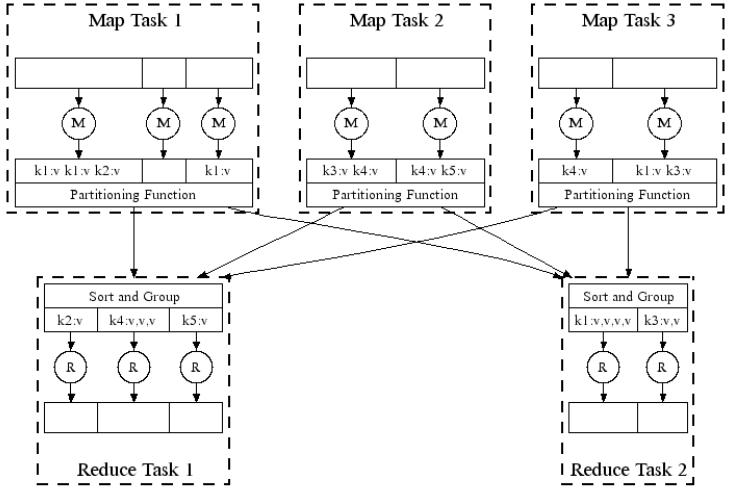
These records can be skipped.

- Records resulting bugs in library for which source code is unavailable.
- A few records which is acceptable to ignore in large statistical analysis.

When the master has seen more than one failure on a particular record, it indicates that the record should be skipped in next re-execution.

Partitioning Function

- Data is partitioned by a partitioning function
- e.g. hash(key) mod R



Combiner Function

- Combines data before it is sent over the network.
- e.g. <the, 1> X 5 ---> <the,5> in word count.
- Executed on each machine that performs a map task.

Ordering Guarantees

- Intermediate key/value pairs are processed in increasing key order.

Input and Output Types

- Provides support for reading input data in several formats.
- e.g. Read a file and generate key value pair as <number of line, contents of the line>
- e.g. Read data from database or mapped memory by *reader* interface

Side-effects

- Produces auxiliary files as additional outputs from map / reduce operators.

Counters

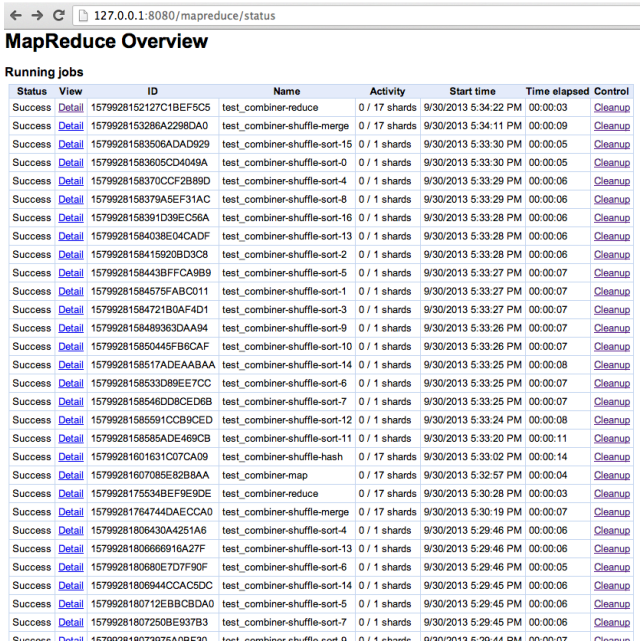
- User code creates a named counter object and increments in map / reduce function.
- Periodically propagated to the master.
- Useful for sanity checking the behavior of MapReduce operations.

```
Counter* uppercase;
uppercase = GetCounter("uppercase");

map(String name, String contents):
  for each word w in contents:
    if (IsCapitalized(w)):
      uppercase->Increment();
    EmitIntermediate(w, "1");
```

Local Execution

- Executes all of the work on the local machine.
- For debugging, profiling and small-scale testing.



Status	View	ID	Name	Activity	Start time	Time elapsed	Control
Success	Detail	1579928152127C1BEF5C5	test_combiner-reduce	0 / 17 shards	9/30/2013 5:34:22 PM	00:00:03	Cleanup
Success	Detail	1579928153286A2298DA0	test_combiner-shuffle-merge	0 / 17 shards	9/30/2013 5:34:11 PM	00:00:09	Cleanup
Success	Detail	15799281583506ADAD929	test_combiner-shuffle-sort-15	0 / 1 shards	9/30/2013 5:33:30 PM	00:00:05	Cleanup
Success	Detail	15799281583605CD4049A	test_combiner-shuffle-sort-0	0 / 1 shards	9/30/2013 5:33:30 PM	00:00:05	Cleanup
Success	Detail	1579928158370CCF2B89D	test_combiner-shuffle-sort-4	0 / 1 shards	9/30/2013 5:33:29 PM	00:00:06	Cleanup
Success	Detail	1579928158379A5EF31AC	test_combiner-shuffle-sort-8	0 / 1 shards	9/30/2013 5:33:29 PM	00:00:06	Cleanup
Success	Detail	1579928158391D39EC56A	test_combiner-shuffle-sort-16	0 / 1 shards	9/30/2013 5:33:28 PM	00:00:06	Cleanup
Success	Detail	1579928158038E04CADF	test_combiner-shuffle-sort-13	0 / 1 shards	9/30/2013 5:33:28 PM	00:00:06	Cleanup
Success	Detail	15799281584159 20BD3C8	test_combiner-shuffle-sort-2	0 / 1 shards	9/30/2013 5:33:28 PM	00:00:06	Cleanup
Success	Detail	1579928158443BFFCA9B9	test_combiner-shuffle-sort-5	0 / 1 shards	9/30/2013 5:33:27 PM	00:00:07	Cleanup
Success	Detail	1579928158457 5FABC011	test_combiner-shuffle-sort-1	0 / 1 shards	9/30/2013 5:33:27 PM	00:00:07	Cleanup
Success	Detail	15799281584721B0AF4D1	test_combiner-shuffle-sort-3	0 / 1 shards	9/30/2013 5:33:27 PM	00:00:07	Cleanup
Success	Detail	1579928158489363DAA94	test_combiner-shuffle-sort-9	0 / 1 shards	9/30/2013 5:33:26 PM	00:00:07	Cleanup
Success	Detail	15799281585044 5FB6CAF	test_combiner-shuffle-sort-10	0 / 1 shards	9/30/2013 5:33:26 PM	00:00:07	Cleanup
Success	Detail	1579928158517ADEAABAA	test_combiner-shuffle-sort-14	0 / 1 shards	9/30/2013 5:33:25 PM	00:00:08	Cleanup
Success	Detail	1579928158533D89EE7CC	test_combiner-shuffle-sort-6	0 / 1 shards	9/30/2013 5:33:25 PM	00:00:07	Cleanup
Success	Detail	1579928158546DD8CED6B	test_combiner-shuffle-sort-7	0 / 1 shards	9/30/2013 5:33:25 PM	00:00:07	Cleanup
Success	Detail	1579928158559 1CCB9CED	test_combiner-shuffle-sort-12	0 / 1 shards	9/30/2013 5:33:24 PM	00:00:08	Cleanup
Success	Detail	1579928158585ADE469CB	test_combiner-shuffle-sort-11	0 / 1 shards	9/30/2013 5:33:20 PM	00:00:11	Cleanup
Success	Detail	15799281601631C07CA09	test_combiner-shuffle-hash	0 / 17 shards	9/30/2013 5:33:02 PM	00:00:14	Cleanup
Success	Detail	15799281607085E82B8AA	test_combiner-map	0 / 17 shards	9/30/2013 5:32:57 PM	00:00:04	Cleanup
Success	Detail	1579928175534BEF9E9DE	test_combiner-reduce	0 / 17 shards	9/30/2013 5:30:28 PM	00:00:03	Cleanup
Success	Detail	15799281764744DAECCA0	test_combiner-shuffle-merge	0 / 17 shards	9/30/2013 5:30:19 PM	00:00:07	Cleanup
Success	Detail	15799281806430A4251A6	test_combiner-shuffle-sort-4	0 / 1 shards	9/30/2013 5:29:46 PM	00:00:06	Cleanup
Success	Detail	1579928180666916A27F	test_combiner-shuffle-sort-13	0 / 1 shards	9/30/2013 5:29:46 PM	00:00:06	Cleanup
Success	Detail	1579928180680E7D7F90F	test_combiner-shuffle-sort-6	0 / 1 shards	9/30/2013 5:29:46 PM	00:00:05	Cleanup
Success	Detail	15799281806944CCAC5DC	test_combiner-shuffle-sort-14	0 / 1 shards	9/30/2013 5:29:45 PM	00:00:06	Cleanup
Success	Detail	1579928180712EBBCBDA0	test_combiner-shuffle-sort-5	0 / 1 shards	9/30/2013 5:29:45 PM	00:00:06	Cleanup
Success	Detail	15799281807250BE937B3	test_combiner-shuffle-sort-7	0 / 1 shards	9/30/2013 5:29:45 PM	00:00:06	Cleanup

Status Information

- The master runs an internal HTTP server and shows status pages.

Summary

- MapReduce is a programming model and an associated implementation for processing and generating large data set with parallel

- A large variety of problems are easily expreeible as MapReduce computations.
- There are many refinements but MapReduce is easy to use since it hides the details.