MapReduce

Cyrus Harvesf

Who am I?

I was in your seat about 12 years ago.

- PhD’08
  - Replica placement for routing fault tolerance in structured distributed hash tables
- Software Engineer at Google since 2012
  - Previous: Software Engineer at Microsoft since 2009

Why am I here?

What is MapReduce?

MapReduce is a programming model for processing and generating large data sets.

Motivation

Google has loads of data!

Example: Google processes over 40,000 search queries per second worldwide.

Think of the:

- Amount of logs generated.
- Size of the search index.
- Number of links crawled.

And that’s just one Google product.

Google loves analyzing data!

- What were the most popular search queries in 2016?
- What have the people in Great Britain searched for most in the last hour?
- What are the most linked news articles from the last month?

See http://trends.google.com for examples of analysis we perform on search queries.

Quick analysis requires lots of machines!

A typical MapReduce process terabytes of data across thousands of machines using commodity hardware.

Sample data from 2004:

- 157 worker machines per job on average
- 1.2 worker deaths per job on average
- 634 seconds per job on average

Distributed computation is hard.

Google started out writing specialized programs to analyze large data sets.

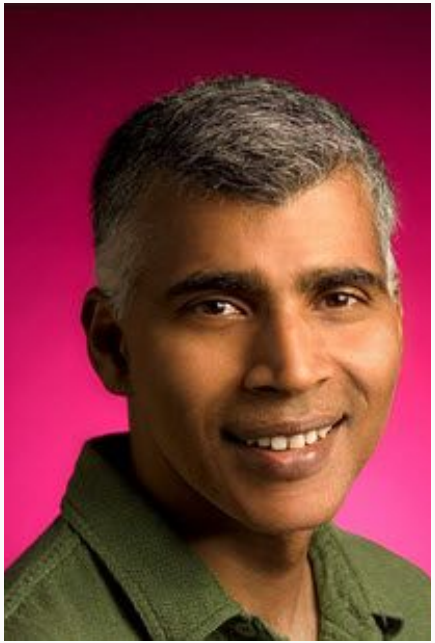
Each of these programs has to solve the same problems:

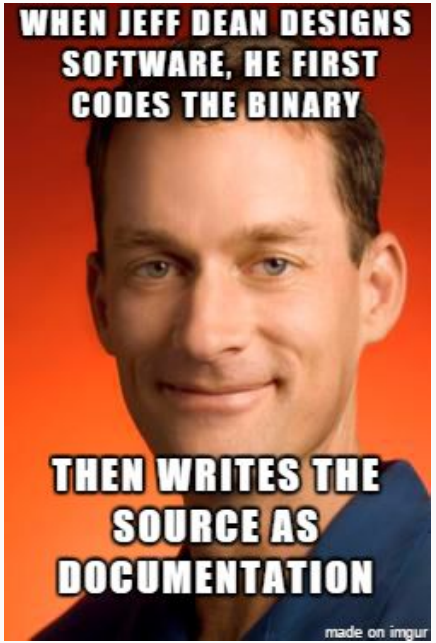
- How to parallelize the computation?
- How to distribute the data across machines?
- How to handle failures?

MapReduce Basics

MapReduce: Simplified Data Processing on Large Clusters

- Jeffrey Dean and Sanjay Ghemawat
- Presented at OSDI'04
- https://goo.gl/lK88am





`map` and `reduce`

MapReduce expresses the distributed computation as two simple functions:

```
map(k1, v1) → (k2, v2)

reduce(k2, list(v2)) → list(v3)
```

Allows you to write a parallelized, fault tolerant program distributed across thousands of machines in just **tens of lines of user code**!

How are these functions used?

1. Input is sharded and shards are assigned to workers called *mappers*.
2. Mappers apply `map` to every record in a shard, producing intermediate output.
3. Intermediate output is shuffled by key and passed to a worker called a *reducer*.
4. Reducers apply `reduce` to the intermediate output to produce the final output

Example Walkthrough

Word Count

Count the number of occurrences of each word in a large collection of documents.

Word Count: `map` function

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

Word Count: `reduce` function

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

`map`: Input and Intermediate Output

Input

```
We are not what
we want to be,
but at least
we are not what
we used to be.
```

Intermediate Output

```
<we, 1>
<are, 1>
<not, 1>
<what, 1>
<we, 1>
<want, 1>
<to, 1>
<be, 1>
...
```

reduce: Shuffled Input and Output

Shuffled Input

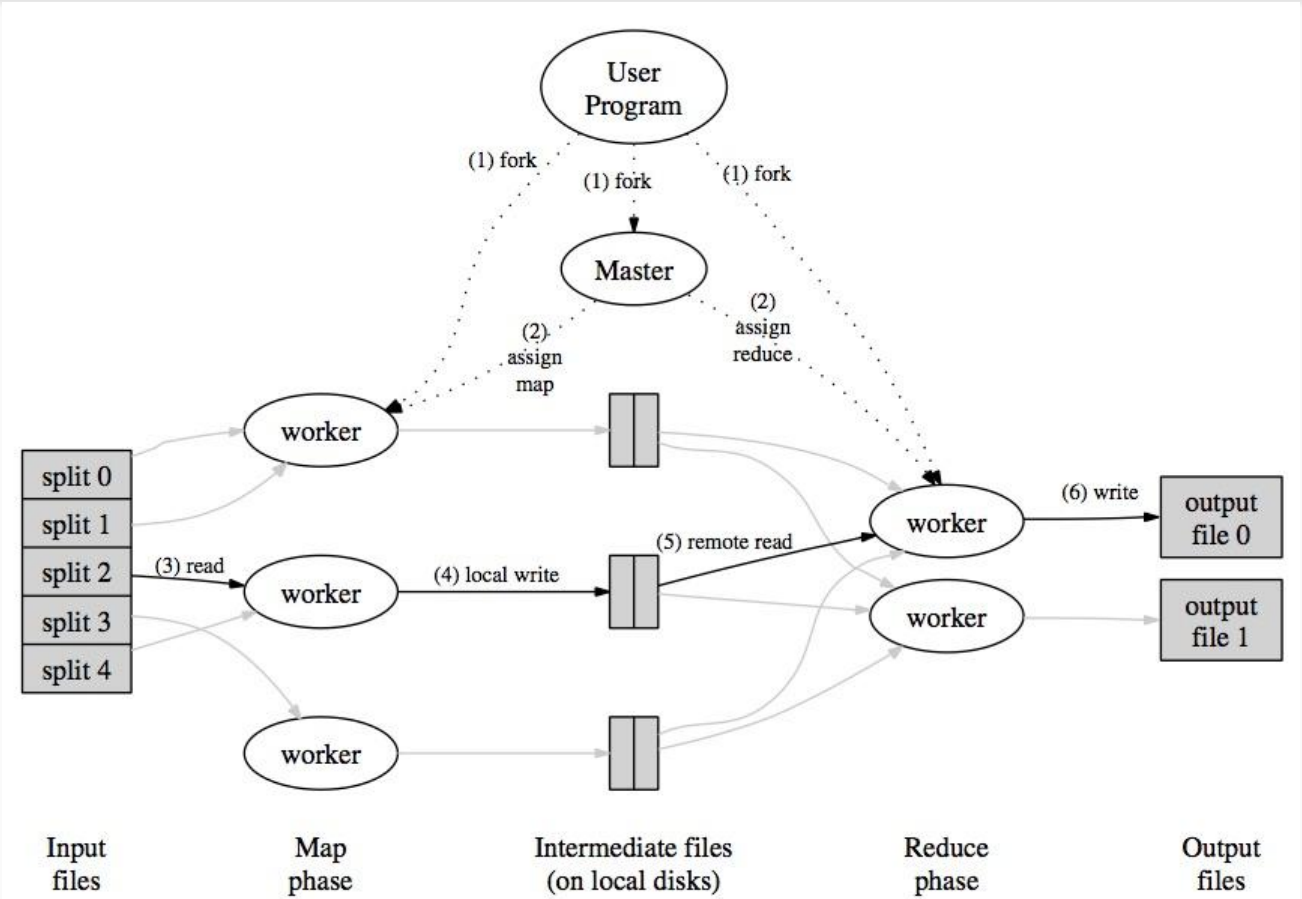
```
<we, (1, 1, 1, 1)>
<are, (1, 1)>
<not, (1, 1)>
<what, (1, 1)>
<want, (1)>
<to, (1, 1)>
<be, (1, 1)>
<but, (1)>
...
```

Output

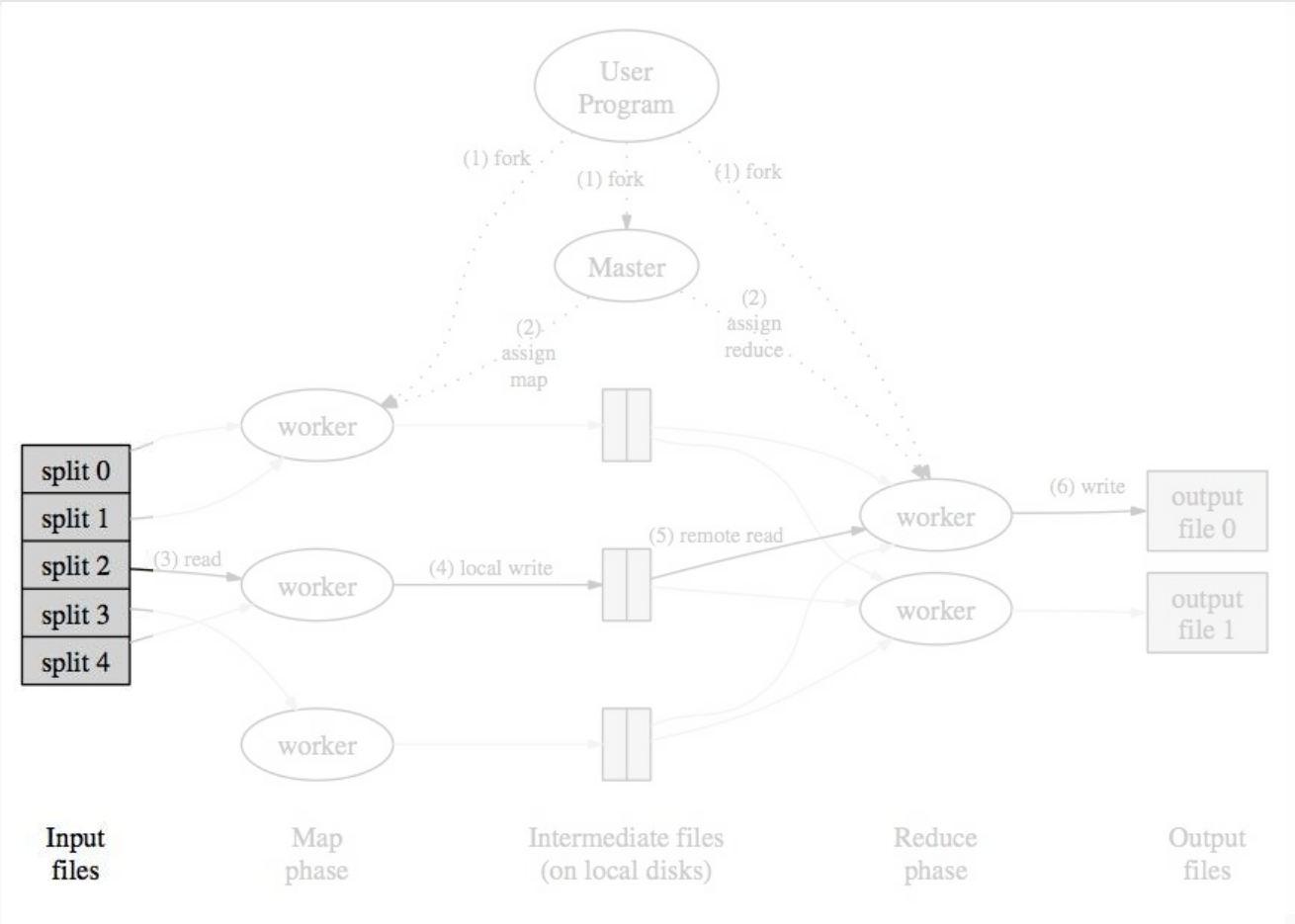
```
<we, 4>
<are, 2>
<not, 2>
<what, 2>
<want, 1>
<to, 2>
<be, 2>
<but, 1>
...
```

MapReduce Framework

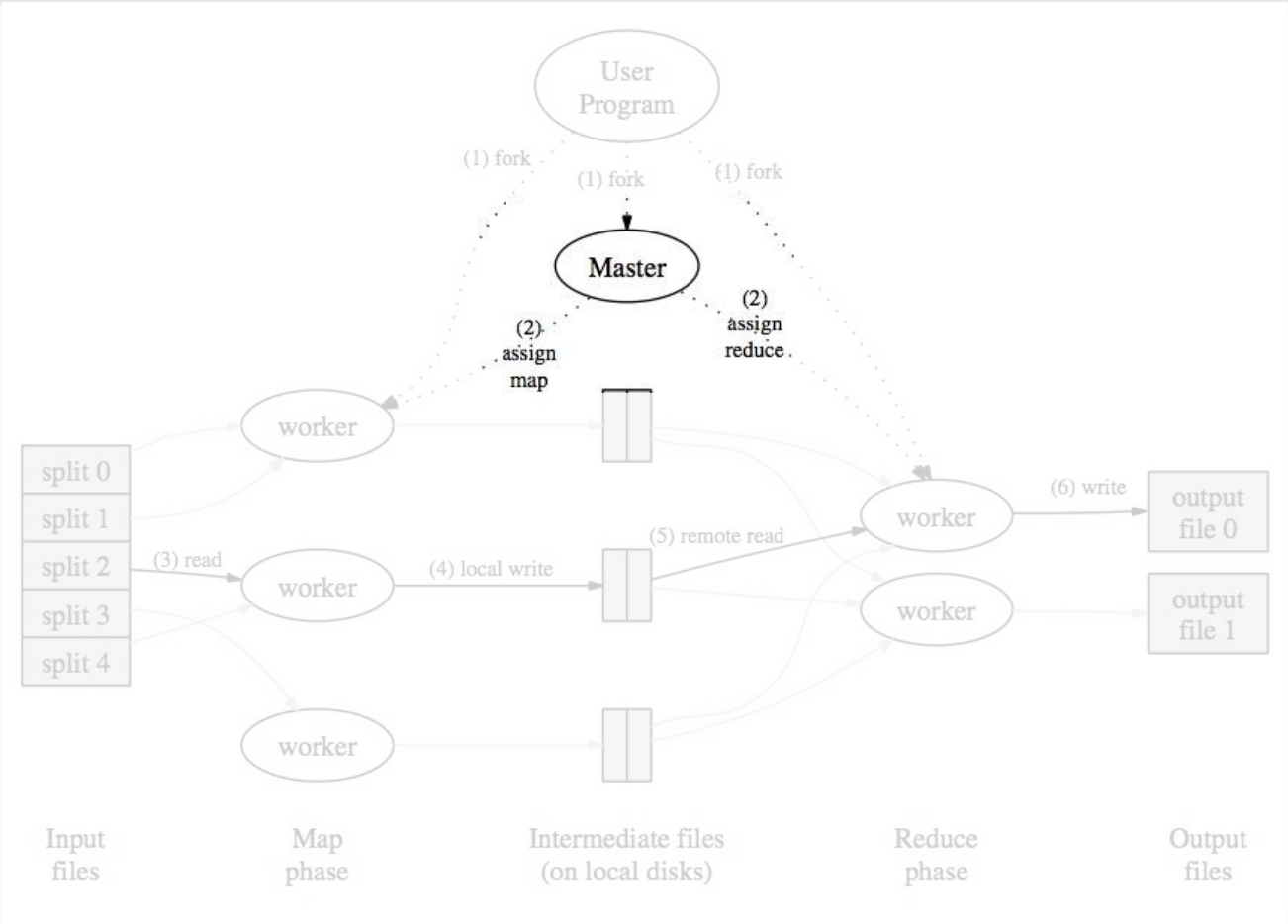
What does the framework do?



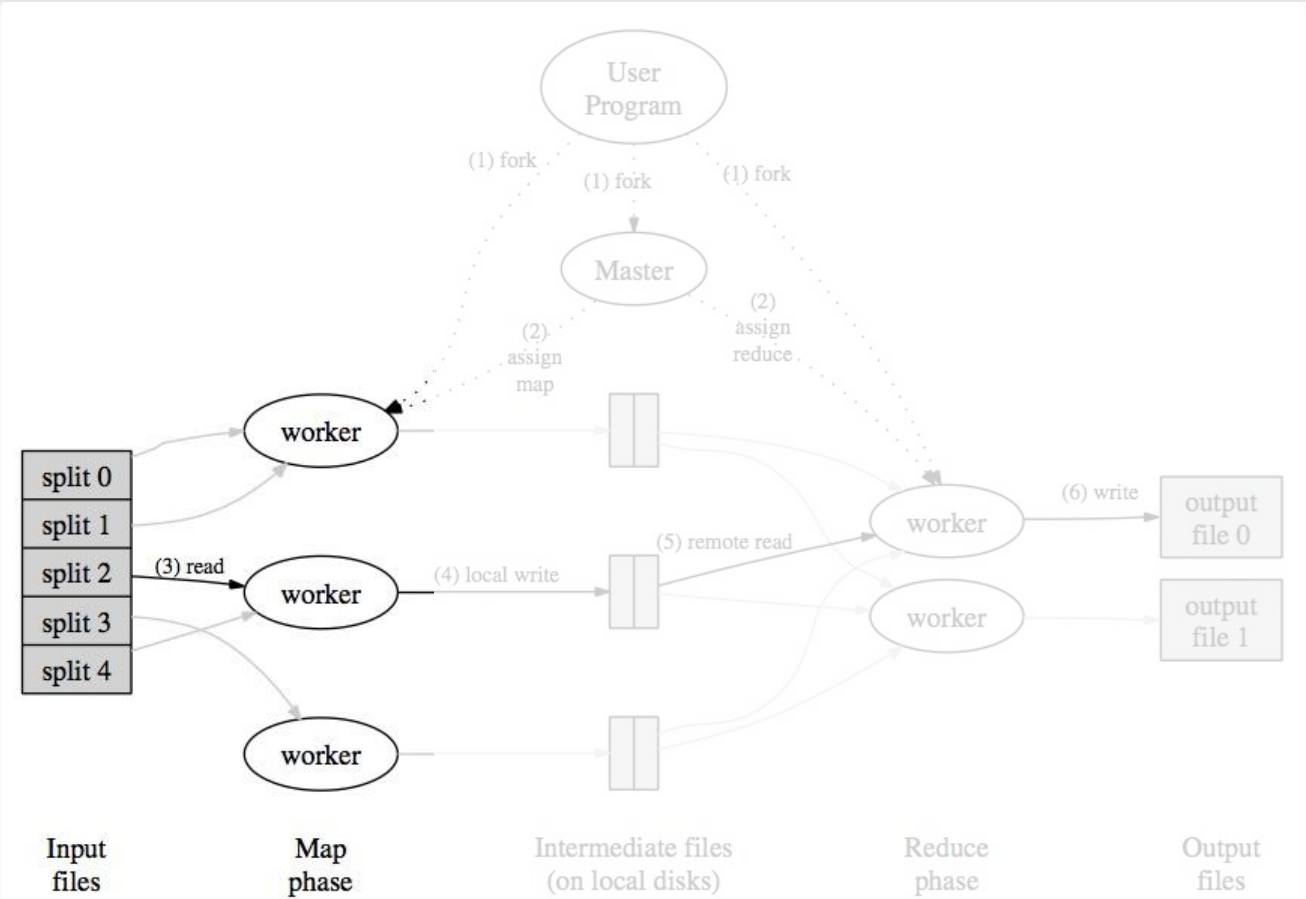
1. Shard the input.



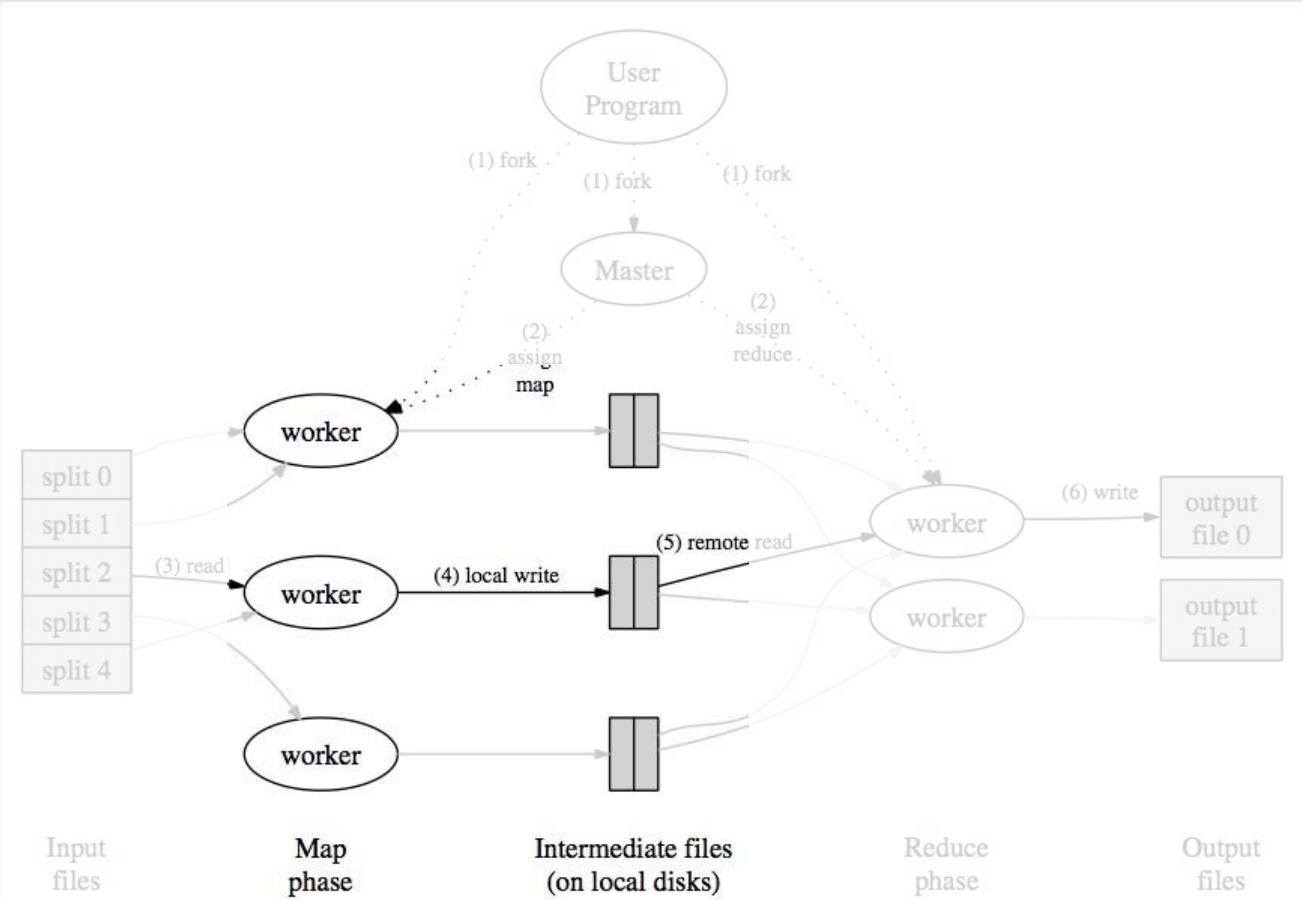
2. Start the Master, whom coordinates work among workers.



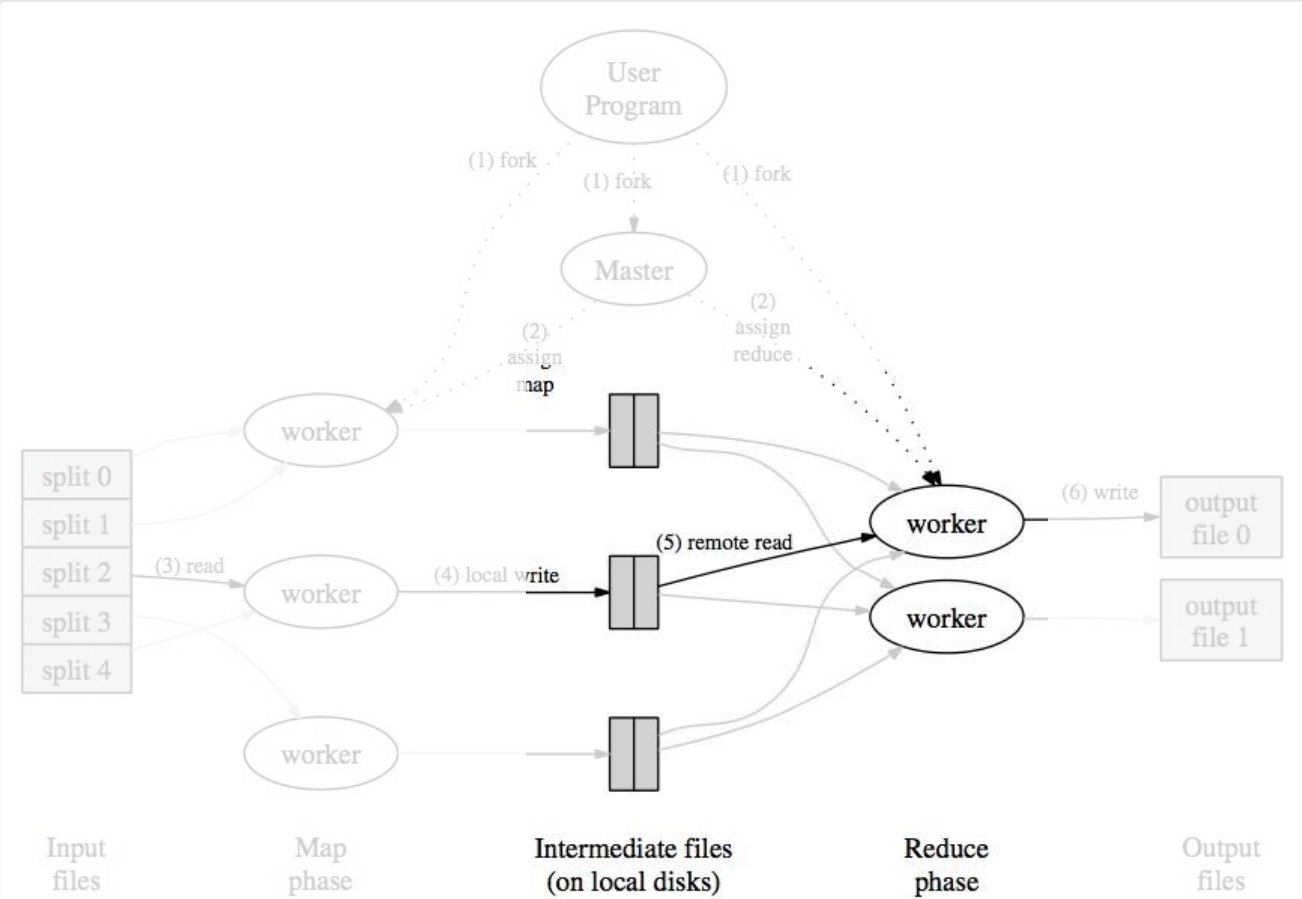
3. Workers read input and apply `map`.



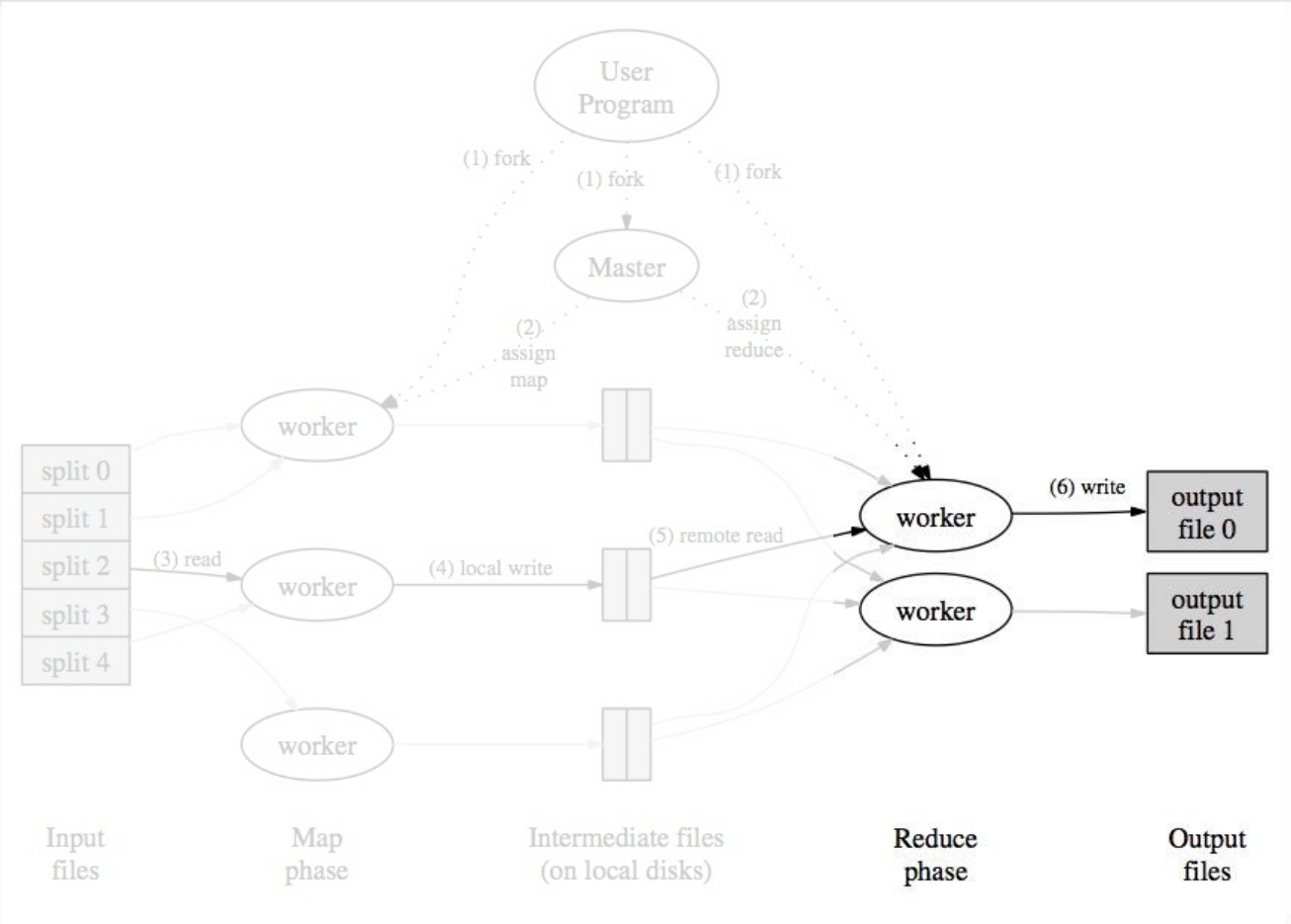
4. Periodically, buffered output is written to disk in partitions.



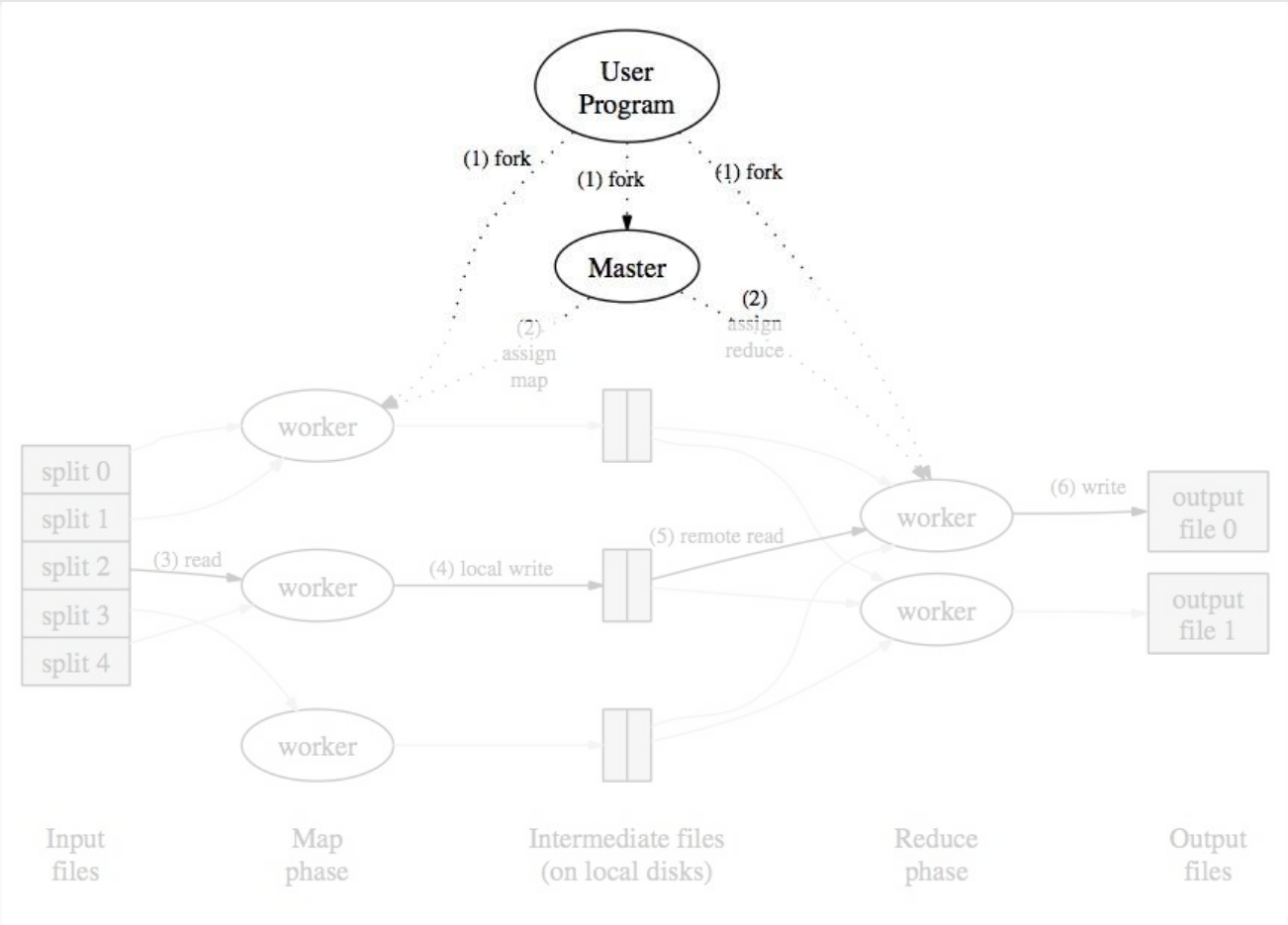
5. Reducers read from disk and shuffle the intermediate output.



6. When all data for a key is read, `reduce` is applied and the output is written.



7. When all map and reduce tasks are complete, Master returns to user code.



What does the Master do?

- Assigns tasks to map and reduce workers
  - Maintains the state of each task
  - Maintains the worker assigned to each task
- Maintains location of intermediate output for reducers

What happens when machines fail?

- Worker failures
  - Master checks worker health periodically
  - Tasks assigned to a failed worker are reset and eligible for another worker
- Master failures
  - Master checkpoints its own state so the job can be restarted from the last checkpoint

What happens when machines are slow?

- Google runs multiple workloads on the same cluster
  - Worker machine may be overloaded, becoming the long pole for the MapReduce
- Master schedules backup tasks to compensate for slow workers

Refinements

- Partitioning function
- Ordering guarantees
- Combiner function
- Input and output types
- Skipping bad records
- Local execution
- Status server
- Counters

App Engine MapReduce

How is App Engine MapReduce built?

MapReduce relies heavily on App Engine's:

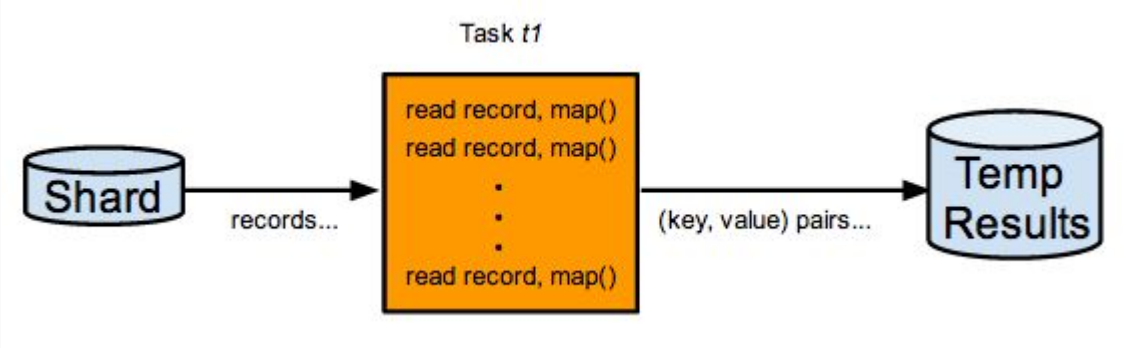
- Dynamic instance scheduling, and
- Task queues

Open sourced and available on Git:

http://github.com/GoogleCloudPlatform/appengine-mapreduce

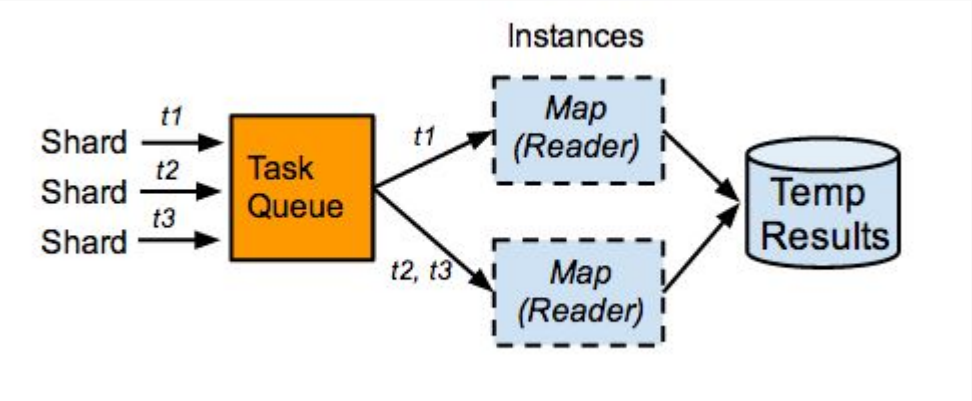
Each shard is modeled as a task.

Each task provides the data to read and the operation to perform (`map`, shuffle or `reduce`).



Task queues “assign” work to instances.

Tasks are executed dynamically via App Engine’s instance scheduling.



Failed and slow tasks are retried.

Task Queues support at least once execution semantics, naturally providing fault tolerance to App Engine MapReduce.

Dynamic instances have a 60 second deadline to handle requests, allowing slow tasks to be retried by another instance.

Demo: Word Count on App Engine

More in Google Cloud Platform

Google’s “Big Data” Products

Cloud Dataflow - http://cloud.google.com/dataflow

- Unified programming model and a managed service for developing and executing a wide range of data processing patterns.
- Uses collections and transforms instead of `map` and `reduce`.
- FlumeJava: Easy, Efficient Data-Parallel Pipelines (PLDI’2010) - https://goo.gl/fKjgoZ

Google’s “Big Data” Products

BigQuery - http://cloud.google.com/bigquery

- Fully managed, petabyte scale, low cost enterprise data warehouse for analytics.
- Allows you to write SQL-like queries over huge data sets.
- Dremel: Interactive Analysis of Web-Scale Datasets (VLDB’2010) - https://goo.gl/biRupw

Questions?