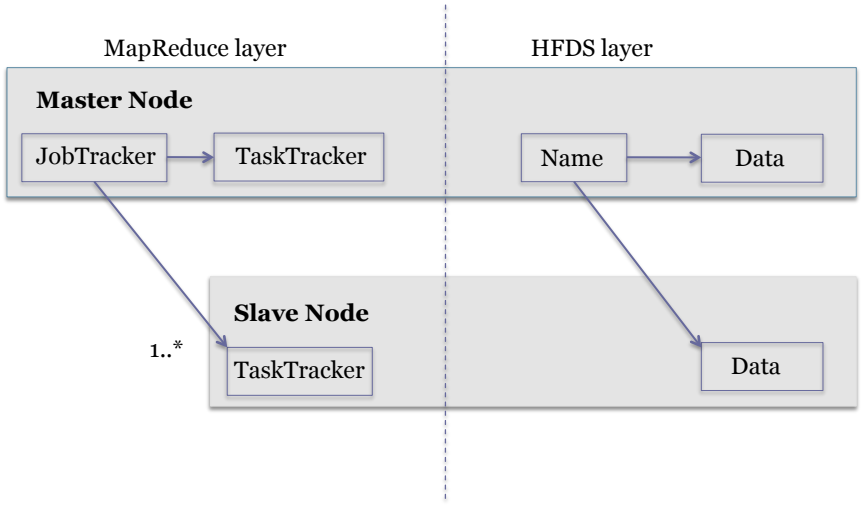
Hadoop MapReduce

- MapReduce is a programming model and software framework first developed by Google (Google's MapReduce paper submitted in 2004)
- Intended to facilitate and simplify the processing of vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner
  - Petabytes of data
  - Thousands of nodes
- Computational processing occurs on both:
  - Unstructured data : filesystem
  - Structured data : database

Hadoop Distributed File System (HFDS)

- Inspired by Google File System
- Scalable, distributed, portable filesystem written in Java for Hadoop framework
  - Primary distributed storage used by Hadoop applications
- HFDS can be part of a Hadoop cluster or can be a stand-alone general purpose distributed file system
- An HFDS cluster primarily consists of
  - NameNode that manages file system metadata
  - DataNode that stores actual data
- Stores very large files in blocks across machines in a large cluster
  - Reliability and fault tolerance ensured by replicating data across multiple hosts
- Has data awareness between nodes
- Designed to be deployed on low-cost hardware

Hadoop simple cluster graphic



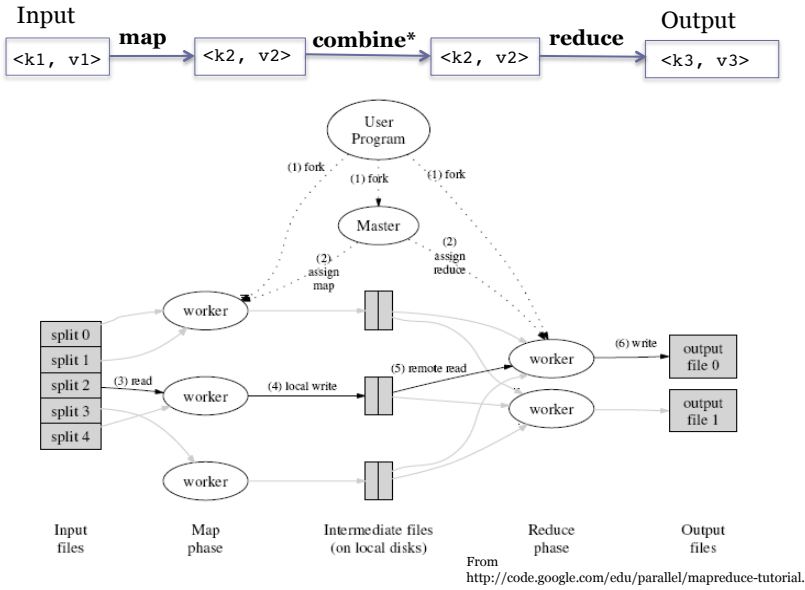
MapReduce framework

- Per cluster node:
  - Single JobTracker per master
    - Responsible for scheduling the jobs' component tasks on the slaves
    - Monitors slave progress
    - Re-executing failed tasks
  - Single TaskTracker per slave
    - Execute the tasks as directed by the master

MapReduce core functionality (II)

- Data flow beyond the two key pieces (map and reduce):
  - Input reader – divides input into appropriate size splits which get assigned to a Map function
  - Map function – maps file data to smaller, intermediate <key, value> pairs
  - Partition function – finds the correct reducer: given the key and number of reducers, returns the desired Reduce node
  - Compare function – input for Reduce is pulled from the Map intermediate output and sorted according to ths compare function
  - Reduce function – takes intermediate values and reduces to a smaller solution handed back to the framework
  - Output writer – writes file output

Input and Output (II)



From http://code.google.com/edu/parallel/mapreduce-tutorial.html

To explain in detail, we'll use a code example: WordCount
Count occurrences of each word across different files

Two input files:

file1: "hello world hello moon"

file2: "goodbye world goodnight moon"

Three operations:

map

combine

reduce

What is the output per step?

MAP

First map:

```
< hello, 1 >
< world, 1 >
< hello, 1 >
< moon, 1 >
```

Second map:

```
< goodbye, 1 >
< world, 1 >
< goodnight, 1 >
< moon, 1 >
```

COMBINE

First map:

```
< moon, 1 >
< world, 1 >
< hello, 2 >
```

Second map:

```
< goodbye, 1 >
< world, 1 >
< goodnight, 1 >
< moon, 1 >
```

REDUCE

```
< goodbye, 1 >
< goodnight, 1 >
< moon, 2 >
< world, 2 >
< hello, 2 >
```

Job details

- Job sets the overall MapReduce job configuration
- Job is specified client-side
- Primary interface for a user to describe a MapReduce job to the Hadoop framework for execution
- Used to specify
  - Mapper
  - Combiner (if any)
  - Partitioner (to partition key space)
  - Reducer
  - InputFormat
  - OutputFormat
  - Many user options; high customizability

Mapper

- Mapper maps input key/value pairs to a set of intermediate key/value pairs
- Implementing classes extend Mapper and override map()
  - Main Mapper engine: Mapper.run()
    - setup()
    - map() for each input record
    - cleanup()
- Mapper implementations are specified in the Job
- Mapper instantiated in the Job
- Output data is emitted from Mapper via the Context object
- Hadoop MapReduce framework spawns one map task for each logical representation of a unit of input work for a map task
  - E.g. a filename and a byte range within that file

Context object details

Recall Mapper code:

```
while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    context.write(word, one);
}
```

- Context object: allows the Mapper to interact with the rest of the Hadoop system
- Includes configuration data for the job as well as interfaces which allow it to emit output
- Applications can use the Context
  - to report progress
  - to set application-level status messages
  - update Counters
  - indicate they are alive

Details on Combiner class and intermediate outputs

- Framework groups all intermediate values associated with a given output key
- Passed to the Reducer class to get final output
- User-specified Comparator can be used to control grouping
- Combiner class can be user specified to perform local aggregation of the intermediate outputs
- Intermediate, sorted outputs always stored in a simple format
  - Applications can control if (and how) intermediate outputs are to be compressed (and the CompressionCode) in the Job

Reducer (III)

- Reduces a set of intermediate values which share a key to a (usually smaller) set of values
- Sorts and partitions Mapper outputs
- Number of reduces for the job set by user via Job.setNumReduceTasks(int)
- Reduce engine
  - receives a Context containing job's configuration as well as interfacing methods that return data back to the framework
  - Reducer.run()
    - setup()
    - reduce() per key associated with reduce task
    - cleanup()

Task Execution and Environment

- TaskTracker executes Mapper/Reducer task as a child process in a separate jvm
- Child task inherits the environment of the parent TaskTracker
- User can specify environmental variables controlling memory, parallel computation settings, segment size, and more

Scheduling

- By default, Hadoop uses FIFO to schedule jobs. Alternate scheduler options: capacity and fair
- Capacity scheduler
  - Developed by Yahoo
  - Jobs are submitted to queues
  - Jobs can be prioritized
  - Queues are allocated a fraction of the total resource capacity
  - Free resources are allocated to queues beyond their total capacity
  - No preemption once a job is running