MapReduce and Hadoop

Outline

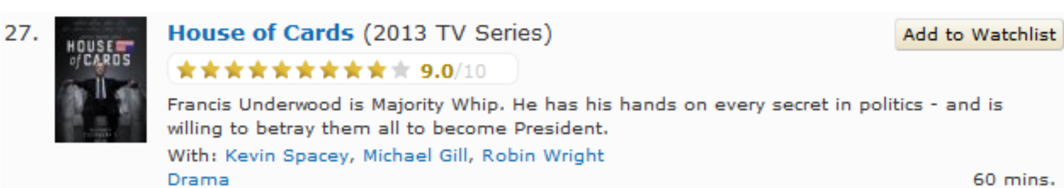
- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

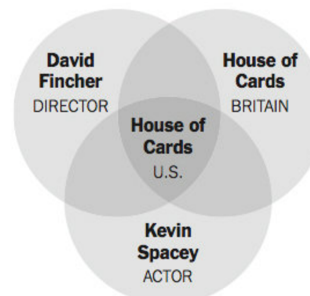
Outline

- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

Data! Big Data!

- We are in a knowledge economy
  - Data is an important asset to any organization
    - ◆2.5 quintillion (2.5×10^{18}) bytes of data were created every day (2012);
    - ◆Facebook has approximately 50 billion photos from its user base;
  - Discovery of knowledge
    - ◆Decoding the human genome: 10 years → one week;
    - ◆Big data analysis in Barack Obama's 2012 re-election campaign;[1]
    - ◆Netflix's original program "House of Cards";[2]





1. "How President Obama's campaign used big data to rally individual voters", http://www.technologyreview.com/featuredstory/509026/how-obamas-team-used-big-data-to-rally-voters/.
2. "How Netflix is turning viewers into puppets", http://www.salon.com/2013/02/01/how_netflix_is_turning_viewers_into_puppets/.

Data Storage

- Storage capacities of hard drives have increased massively, while access speeds have not kept up
  - 1 GB of data with transfer speed of 4.4 MB/s (1990)
    - You could read all the data from a full drive in around 5 min;
  - 1 TB drive with transfer speed around 100 MB/s (2010)
    - It takes more than 2.5 hour to read all the data off the disk; (write is even slower)
- Read from multiple disk? Say, 100 disks.
  - 100 drives working in parallel, each holding one hundredth of the data → less than 2 minutes;
  - Wasteful? → store other datasets and provide shared access;

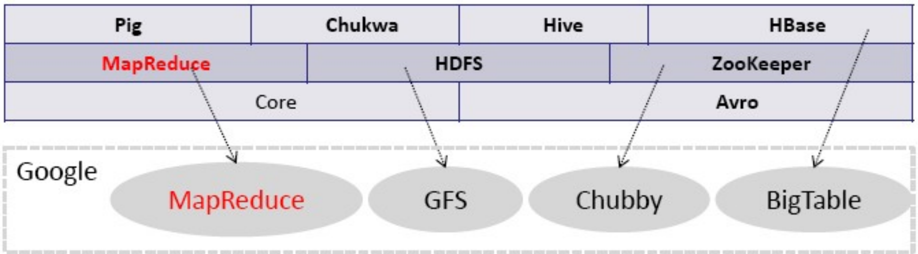
What we are look for?

- Problems?
  - Storage: Hardware failure
    - Law of truly large numbers;
    - Storage problem;
  - Analysis: Combine the data
    - Bandwidth/IO bottleneck;
    - Analysis problem;
  - In summary, how to store and analyze the data
- We are looking at newer
  - File systems and programming models;
  - Supporting algorithms and data structures;

Breakthroughs

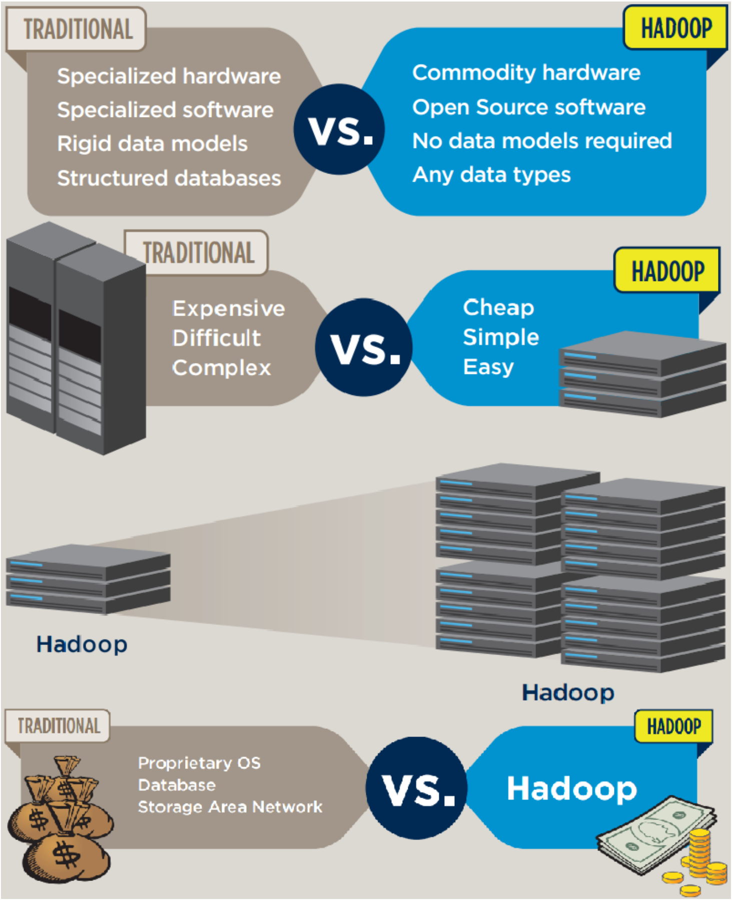
proprietary!!

- The Google File System
  - Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, 19th ACM Symposium on Operating Systems Principles, 2003;
  - http://research.google.com/archive/gfs.html;
- MapReduce: Simplified Data Processing on Large Clusters
  - Jeffrey Dean and Sanjay Ghemawat, 6th Usenix Symposium on Operating System Design and Implementation, 2004;
  - http://research.google.com/archive/mapreduce.html;
- Hadoop Distributed File System
  - Doug Cutting and Yahoo! Inc., 2005;
  - http://hadoop.apache.org/;



Benefits

- Overcomes the traditional limitations of storage and computation
- Leverage inexpensive, commodity hardware as the platform
- Provides linear scalability from 1 to 5000 servers
- Low cost, open source software



With MapReduce Google does…

- Large-scale web search indexing
- Clustering problems for Google News
- Produce reports for popular queries
  - E.g. Google Trend
- Processing of satellite imagery data
- Language model processing for statistical machine translation
- Large-scale machine learning problems
- Plain tool to reliably spawn large number of tasks
  - E.g. parallel data backup and restore

Who are using Hadoop…

- Adobe
- Amazon (EC2, S3)
- Facebook
- Foursquare
- IBM
- New York Times
- Twitter
- Yahoo!
- More on:
  - http://wiki.apache.org/hadoop/PoweredBy

Hadoop Ecosystem

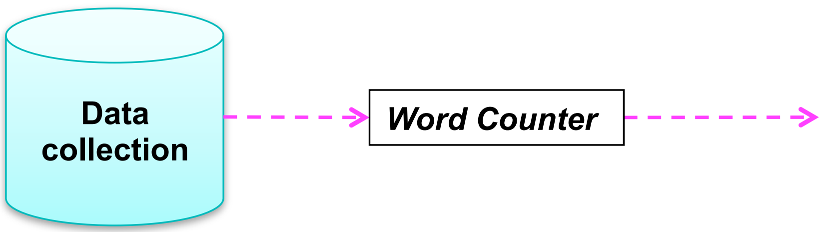
- **Apache Hadoop:**
  - Hadoop Common A set of components and interfaces for distributed file systems and general I/O;
  - Hadoop Distributed File System (HDFS) A distributed file system that runs on large clusters of commodity machines;
  - Hadoop MapReduce A distributed data processing model and execution environment that runs on large clusters of commodity machines;
- **Hadoop-related projects at Apache:**
  - Ambari A tool for provisioning, managing, and monitoring Hadoop clusters.
  - Avro A serialization system for cross-language RPC, and persistent data storage;
  - Cassandra A scalable multi-master database with no single points of failure;
  - Chukwa A data collection system for managing large distributed systems.
  - HBase A distributed, column-oriented database.
  - Hive A distributed data warehouse provides a query language based on SQL.
  - Mahout A Scalable machine learning and data mining library.
  - Pig A data flow language and execution environment for exploring huge datasets.
  - ZooKeeper A distributed, highly available coordination service.

Outline

- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

Word Count

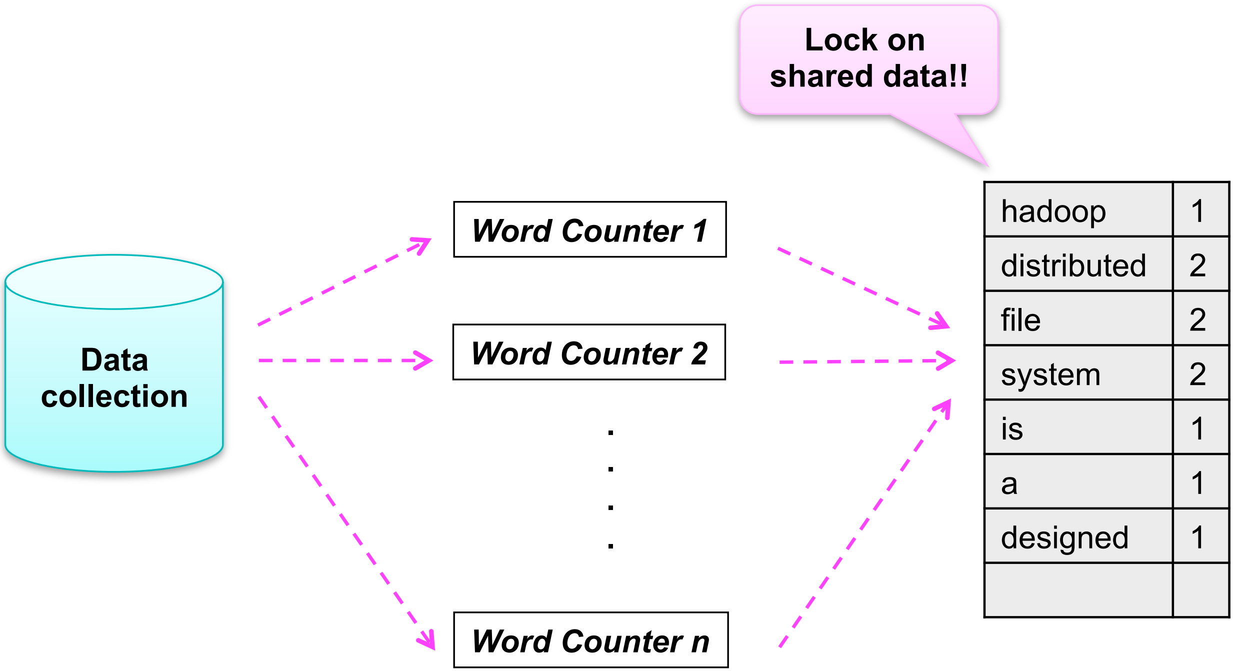
- Consider a large data collection
  – {hadoop, distributed, file, system, is, a, distributed, file, system, designed, …};
- Problem
  – Count the occurrences of the different words in the collection;
- Naïve solution



hadoop	1
distributed	2
file	2
system	2
is	1
a	1
designed	1

Word Count (Cont’d)

- Multiple Threading



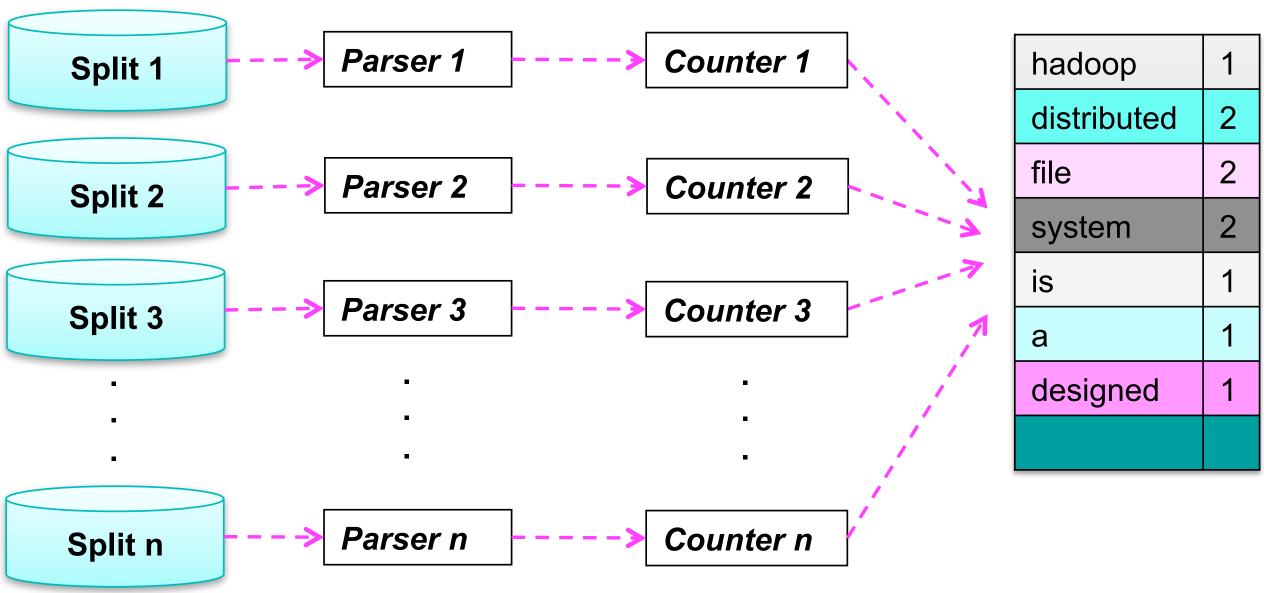
Word Count (Cont’d)

- Peta-scale Data: a single machine cannot handle

hadoop	1
distributed	2
file	2
system	2
is	1
a	1
designed	1

Word Count (Cont’d)

- Divide and Conquer?
  - Separated data
  - Separated counters



Word Count (Cont’d)

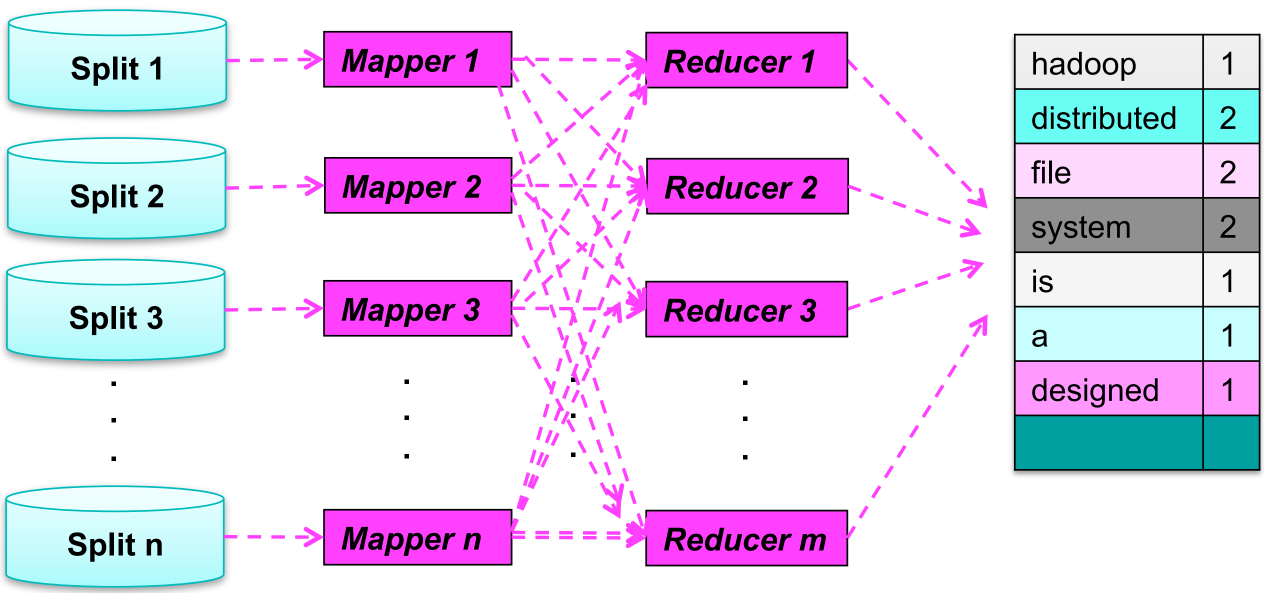
- Peta-scale Data
  - Distributed system;
  - Large number of commodity hardware disks
    - Issue: fault-tolerant; data communication (e.g., monitoring); …
- Problem properties
  - Iterate over a large number of records;
  - Extract something of interest from each;
  - Shuffle and sort intermediate results;
  - Aggregate intermediate results;
  - Generate final output;

Exploit parallelism by splitting parsing and counting

Typical properties of Large-data Problem!!

Word Count (Cont’d)

- MapReduce
  - Mapping operation (input → <key, value> pairs);
  - Reduce operation (<key, value> pairs reduced);

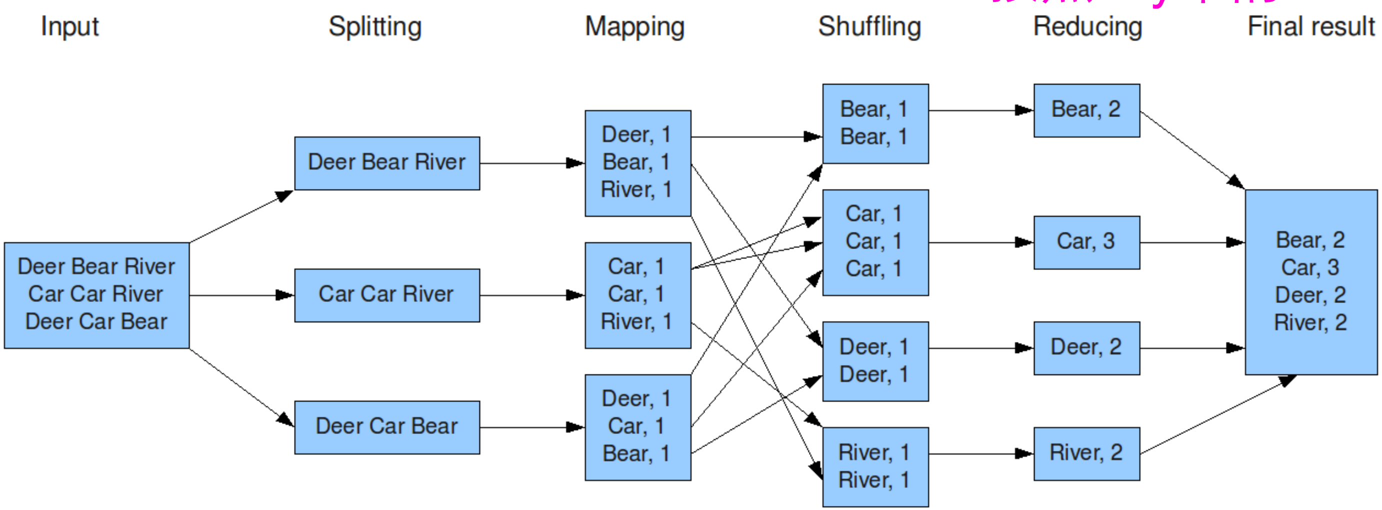


Word Count (Cont’d)

- Determine how many times different words appear in a set of files
  - foo.txt: Deer Bear River
  - bar.txt: Car Car River Deer Car Bear

The overall MapReduce word count process

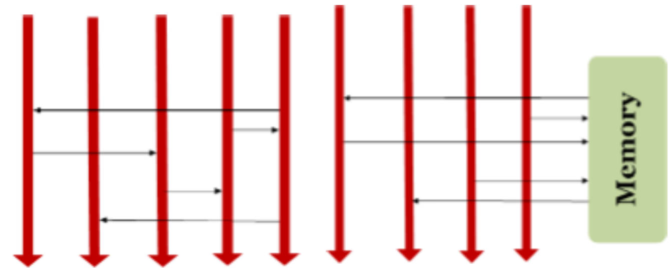
按照key来的



It is not easy to parallel….

Different programming models

Message Passing   Shared Memory



Fundamental issues

Scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …

Architectural issues

Flynn's taxonomy (SIMD, MIMD, etc.), network topology, bisection bandwidth, cache coherence, …

Different programming constructs

Mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues,. …

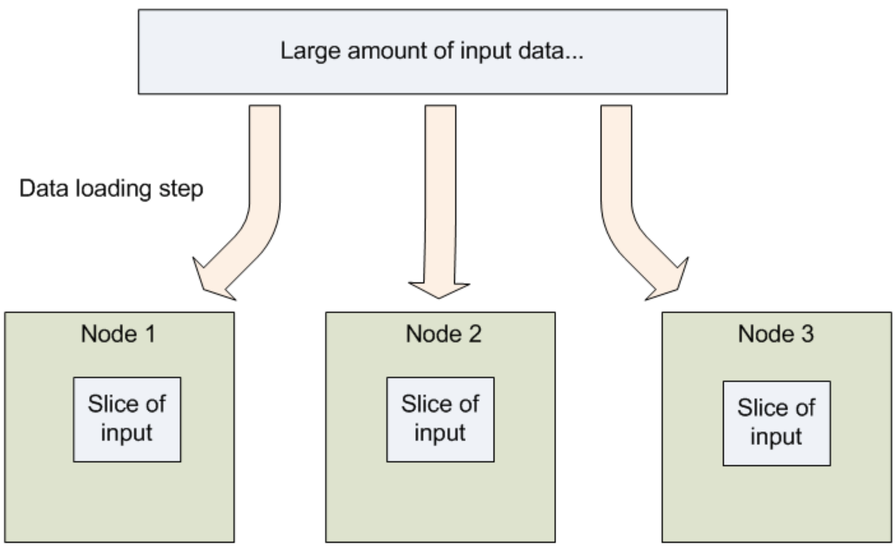
Common problems

Livelock, deadlock, data starvation, priority inversion, …dining philosophers, sleeping barbers, cigarette smokers, …

- **MapReduce/Hadoop: Automate for you**

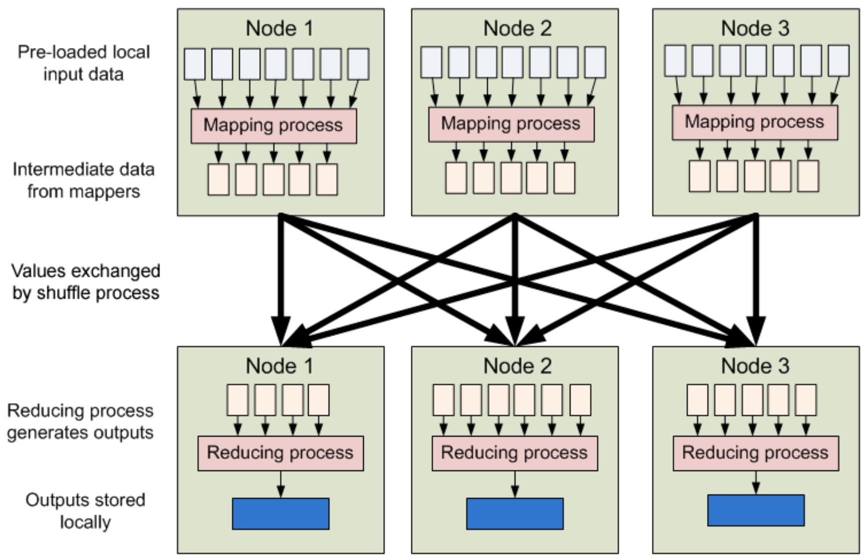
Word Count (Cont’d)

- MapReduce on Hadoop
  - Data is distributed to all the nodes of the cluster as it is being loaded in.
    - Split large data files into chunks which are managed by different nodes;
    - Each chunk is replicated across several machines;
    - Which data operated on by a node is chosen based on its locality to the node;



Word Count (Cont’d)

- MapReduce on Hadoop (Cont’d)
  - Limits the amount of communication can be performed by the processes
    - Each individual record is processed by a task in isolation from one another;
    - Records are processed in isolation by tasks called ***Mappers***;
    - Results from different ***Mappers*** can be merged together by tasks called ***Reducers***.



Outline

- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

Hadoop Distributed File System (HDFS)

- Google File System (GFS)

  proprietary!!

  - Developed by Google Inc.;
  - MapReduce operations run on GFS.;
  - A new version of the GFS is codenamed ***Colossus***;
- Hadoop Distributed File System (HDFS)
  - Created by Doug Cutting and Yahoo! Inc.;
  - Derived from Google's MapReduce and GFS papers;
  - Licensed under the Apache v2 License;

  open-source!!

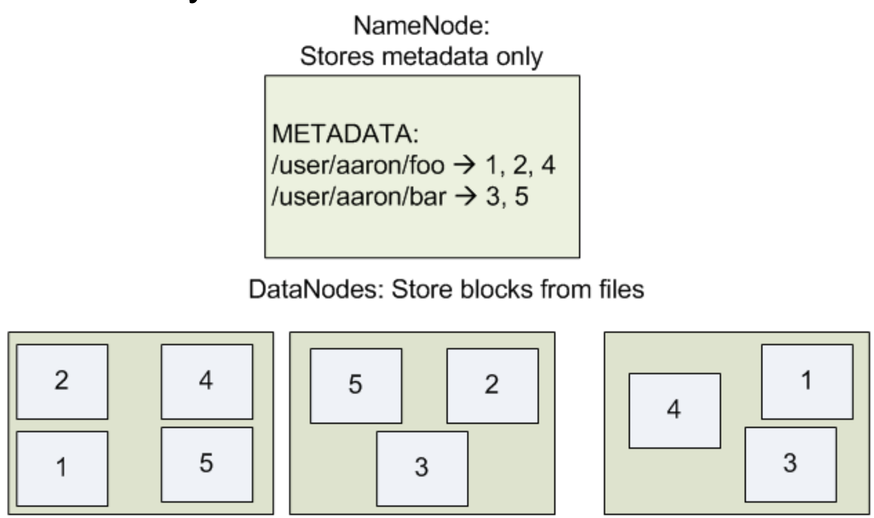
- HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.

Design of HDFS

- Don’t move data to workers… move workers to the data!
- Commodity hardware over “exotic” hardware;
- High component failure rates;
- “Modest” number of HUGE files;
- Files are read-many, write-once (mostly appended to);
- Large streaming reads over random access;
- High sustained throughput over low latency;

DataNodes and NameNode

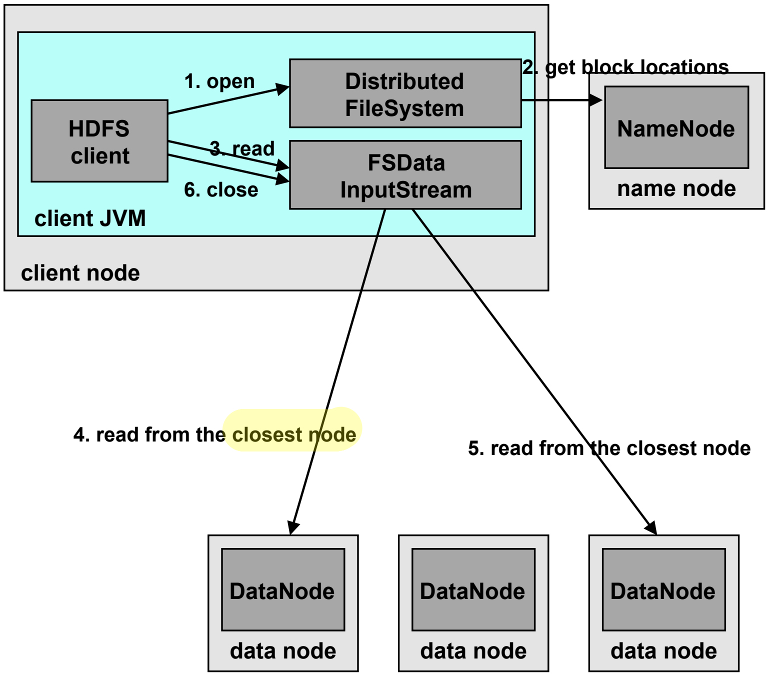
- HDFS is a block-structured file system
  - Individual files are broken into blocks of a fixed size (64MB by default);
  - Blocks are stored across a cluster of one or more machines;
- DataNodes (workers/slaves) v.s. NameNode (the master)
  - DataNodes: Individual machines in the cluster;
  - NameNode: A single machine stores all the metadata for the file system; small meta data and stored in memory

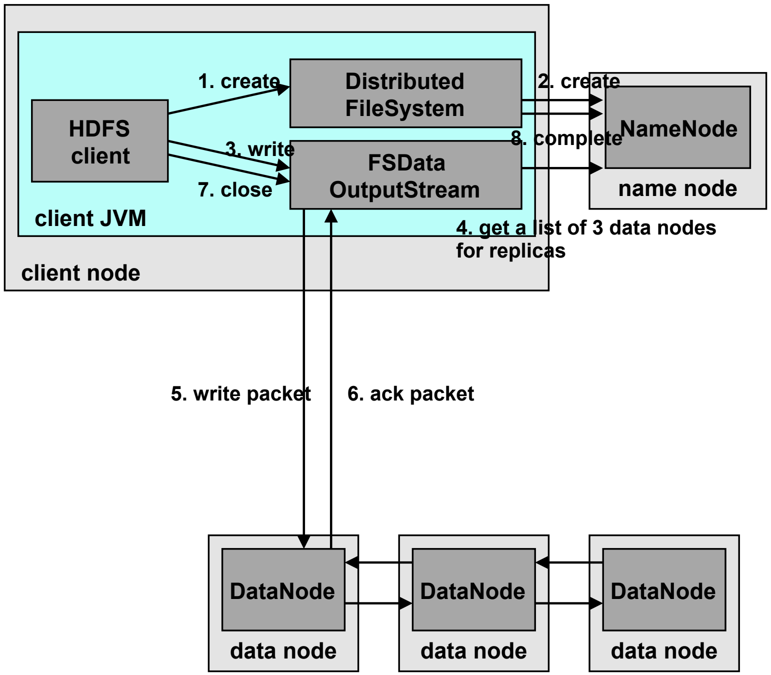


File Read/Write

- The namenode (master) is responsible for maintaining the file namespace and directing clients to datanodes;
- Datanodes (slaves) hold data blocks containing user data;

- File Read
- File Write





Outline

- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

Functional Programming: LISP

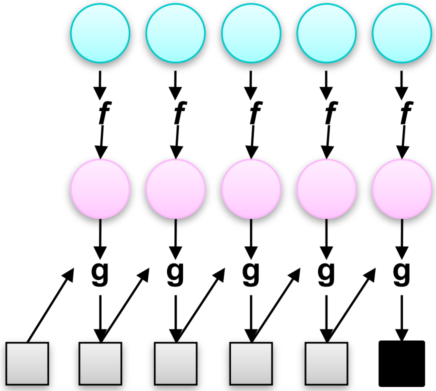
- LISP (LISt Processing)
  - Lists are primitive data types;
  - Functions written in prefix notation;
  - Two high-level functions: ***map*** and ***fold***;
    - Map: do something to every element in a list

```
(map (lambda (x) (* x x))
     '(1 2 3 4 5))
 → '(1 4 9 16 25)
```

    - Fold: combine results of a list in some way

```
(fold + 0 '(1 2 3 4 5)) → 15
```

Sum of Square

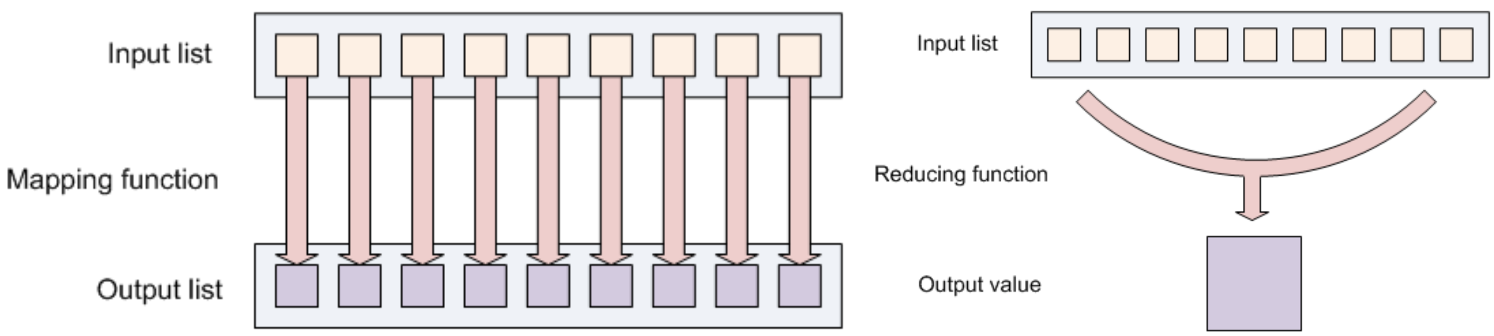


From LISP to MapReduce

- Consider a long list of records: imagine if...
  - We can distribute the execution of map operations to multiple nodes;
  - We have a mechanism for bringing map results back together in the fold operation;
- That's MapReduce! (and Hadoop)
- Implicit parallelism:
  - We can parallelize execution of map operations since they are isolated;
  - We can reorder folding if the fold function is commutative and associative;

Map and Reduce

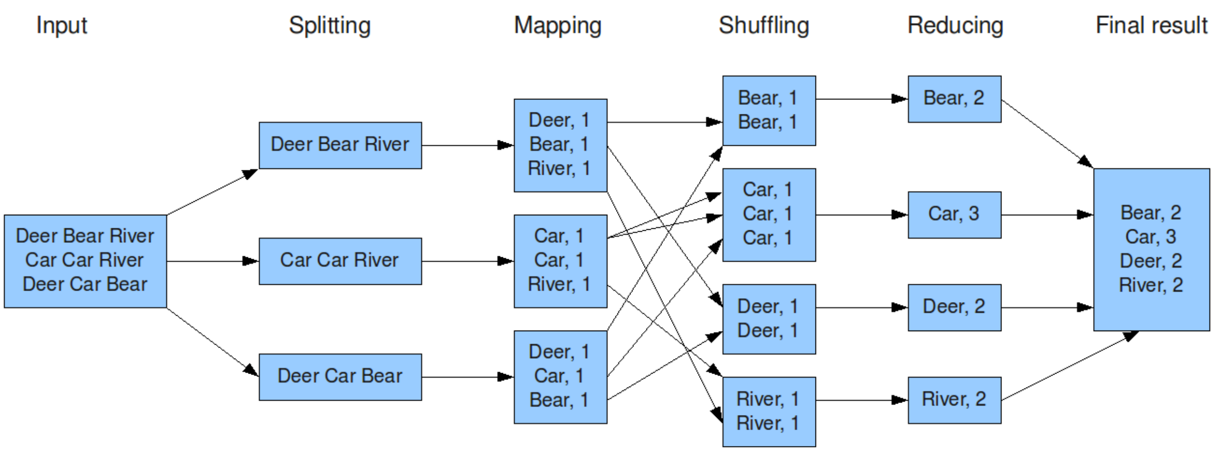
- The first phase of MapReduce program is called Mapping.
- Reducing lets you aggregate values together.



Word Count (Cont' d)

- Determine how many times different words appear in a set of files
  - foo.txt: Deer Bear River
  - bar.txt: Car Car River Deer Car Bear
- Application:
  - Analyze web server logs to find popular URLs;

The overall MapReduce word count process



Word Count (Cont’d)

- Several instances of the mapper function are created on the different machines of the cluster
  - Each instance receives a different input file;
- Several instances of the reducer method are also instantiated on the different machines
  - Each reducer is responsible for processing the list of values associated with a different word;

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

Word Count (Cont’d)

- Implementation of Map

four formal parameters: (input key, input value, output key and output value)
Input key is LongWritable: the byte offset within the file of the beginning of the line.

// org.apache.hadoop.iopackage
LongWritable ←→ Long;
Text ←→ String
IntWritable ←→ Integer

```
public static class WordCountMap extends Mapper<LongWritable, Text, Text, IntWritable>
{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
        {
                String line = value.toString();
                StringTokenizer tokenizer = new StringTokenizer(line);
                while(tokenizer.hasMoreTokens())
                {
                        word.set(tokenizer.nextToken());
                        context.write(word, one);
                }
        }
}
```

context: to write the output to. Context can allow user code to communicate with the system.

Word Count (Cont’d)

- Implementation of Reduce

four formal parameters: (input key, input value, output key and output value)
// input key and input value type must match with Mapper output, output key

```
public static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
        {
                int sum = 0;
                for(IntWritable value: values)
                {
                        sum += value.get();
                }
                context.write(key, new IntWritable(sum));
        }

}
```

Word Count (Cont’d)

- Skeleton

```
public class WordCount extends Configured implements Tool {

        public static class WordCountMap extends Mapper<LongWritable, Text, Text, IntWritable>


        public static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>


        public int run(String[] args) throws Exception


        /**
         * @param args
         * @throws Exception
         */
        public static void main(String[] args) throws Exception {
                // TODO Auto-generated method stub
                int result = ToolRunner.run(new WordCount(), args);
                System.exit(result);
        }
}
```

Word Count (Cont’d)

- Gluing Map and Reduce together

```
public int run(String[] args) throws Exception  {
        // TODO Auto-generated method stub
        Job job = new Job(getConf());
        job.setJarByClass(WordCount.class);
        job.setJobName("wordcount");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(WordCountMap.class);
        job.setCombinerClass(WordCountReducer.class);
        job.setReducerClass(WordCountReducer.class);


        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);


        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        boolean success = job.waitForCompletion(true);
        return success ? 0: 1;
}
```

The setOutputKeyClass() and setOutputValueClass() methods control the output types for the map and the reduce functions

Job object forms the specification of the job.

setInputFormatClass() method defines how the input files are split up and read, and setOutputFormatClass() defines how the output is written to disk.

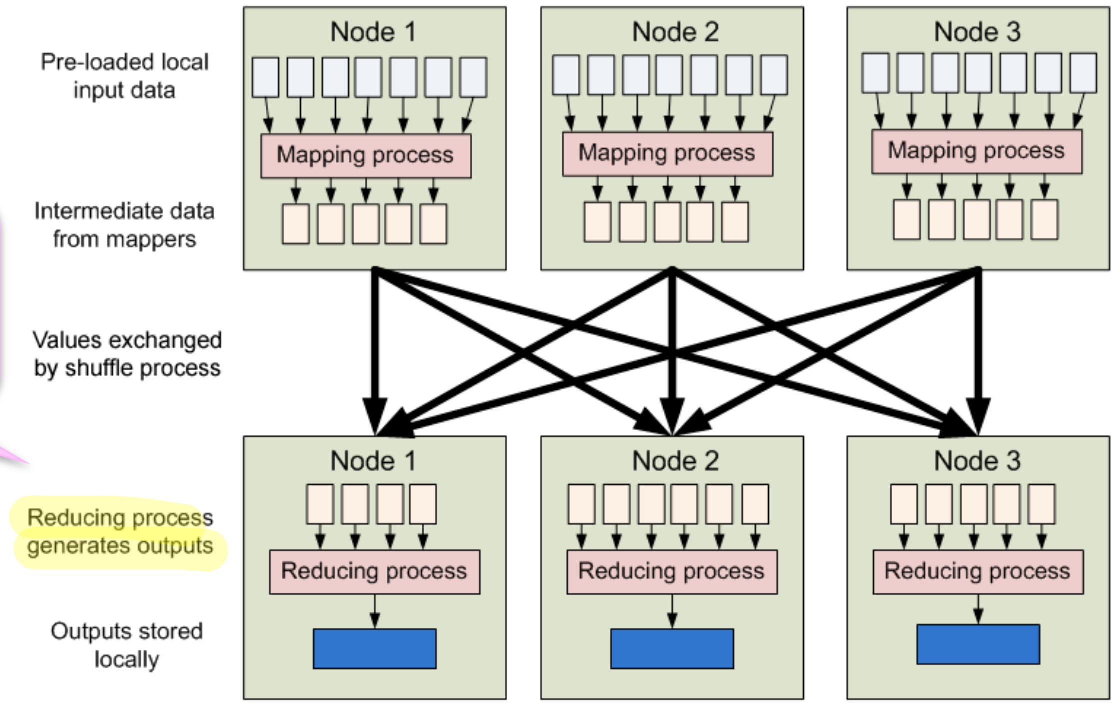
specify the input and output paths

waitForCompletion() method is a boolean indicating success (true) or failure (false)

MapReduce Data Flow

- High-level MapReduce pipeline

the only communication step in MapReduce
No side effect. Can restart any tasks/nodes

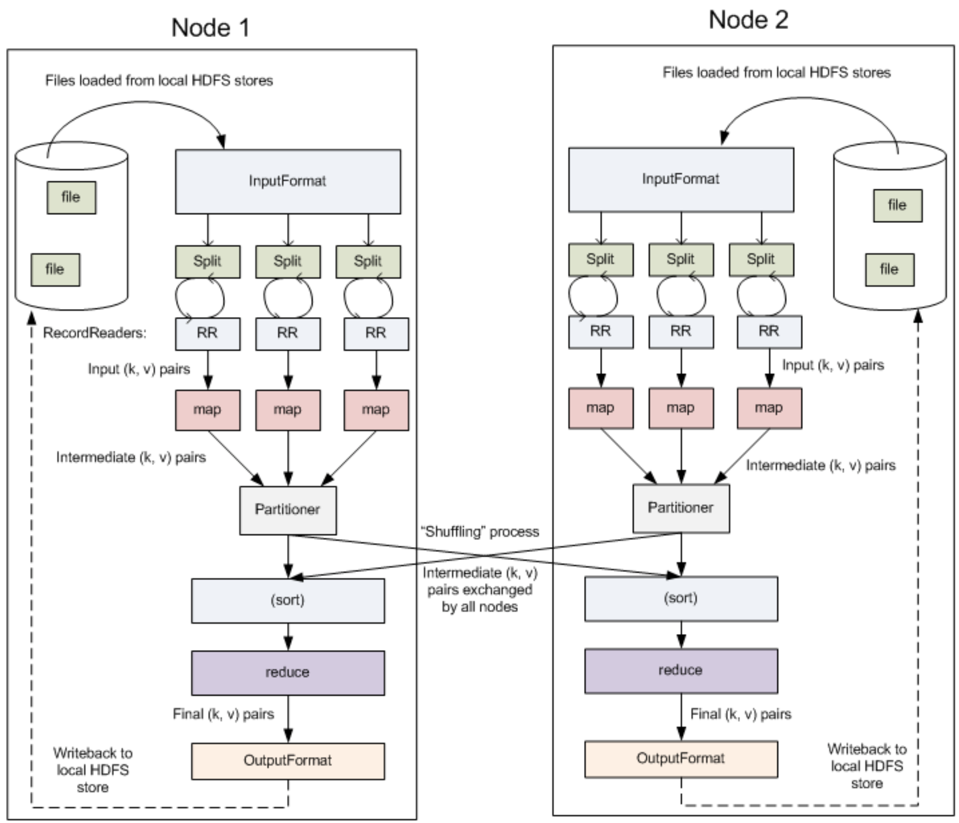


MapReduce Data Flow (Cont'd)

- Detailed Hadoop MapReduce data flow

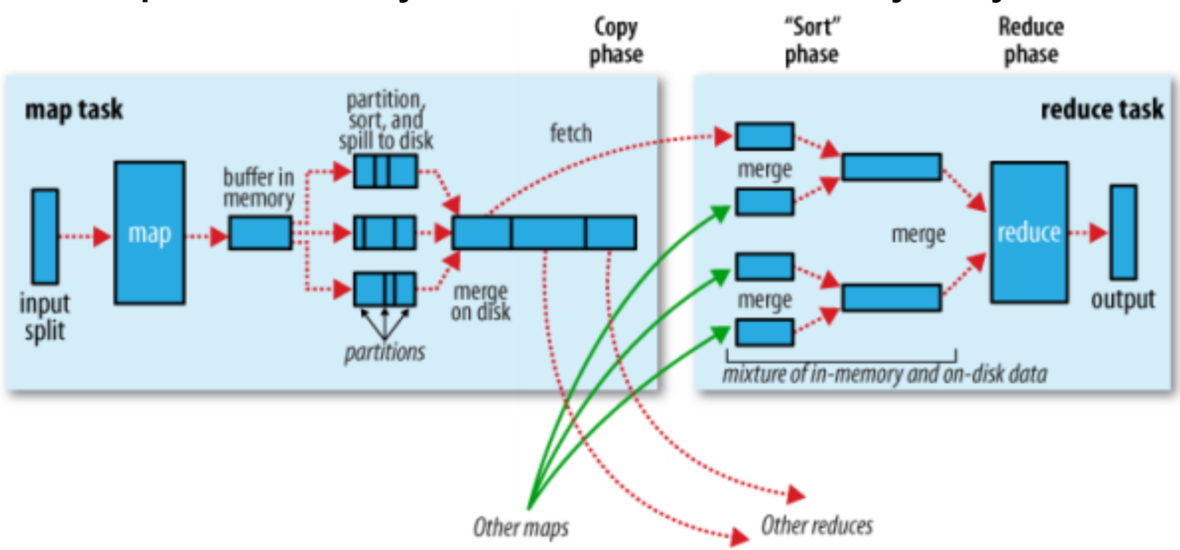
The default *InputFormat* is the TextInputFormat. This treats each line of each input file as a separate record, and performs no parsing.

RecordReader (RR) class actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper.



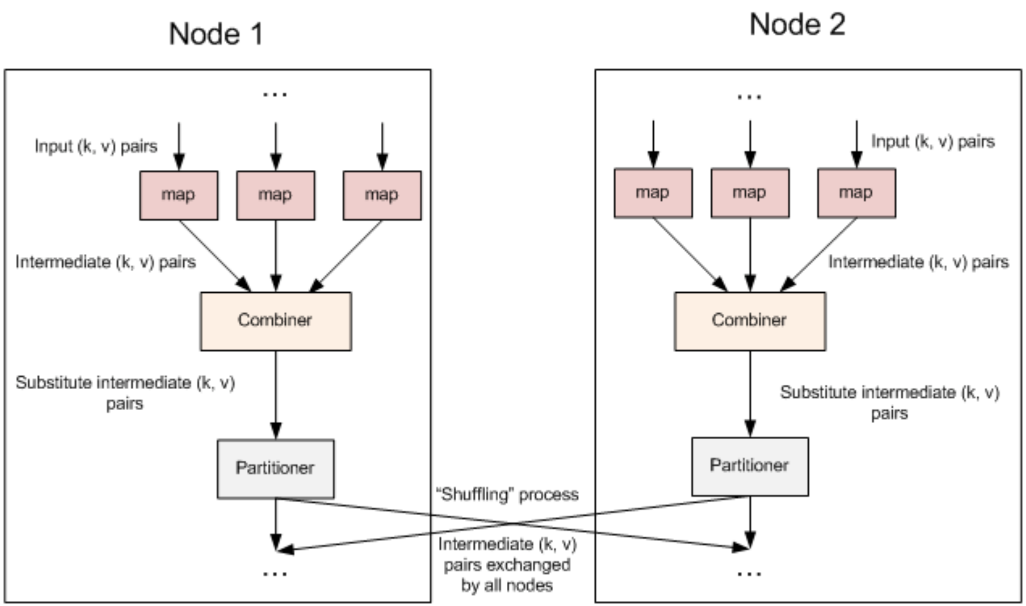
MapReduce Data Flow (Cont’d)

- Partition and Sort
  - Where the “magic” happens;
  - Decide which reducer is responsible for a particular key.
  - The input to every reducer is sorted by key;



MapReduce Data Flow (Cont’d)

- Further improvement → Combiner
  - Optimizing bandwidth usage;
  - After the Mapper and before the Reducer; (mini-reducers before reduce phase).
  - Usage of the Combiner is optional;



Combiner

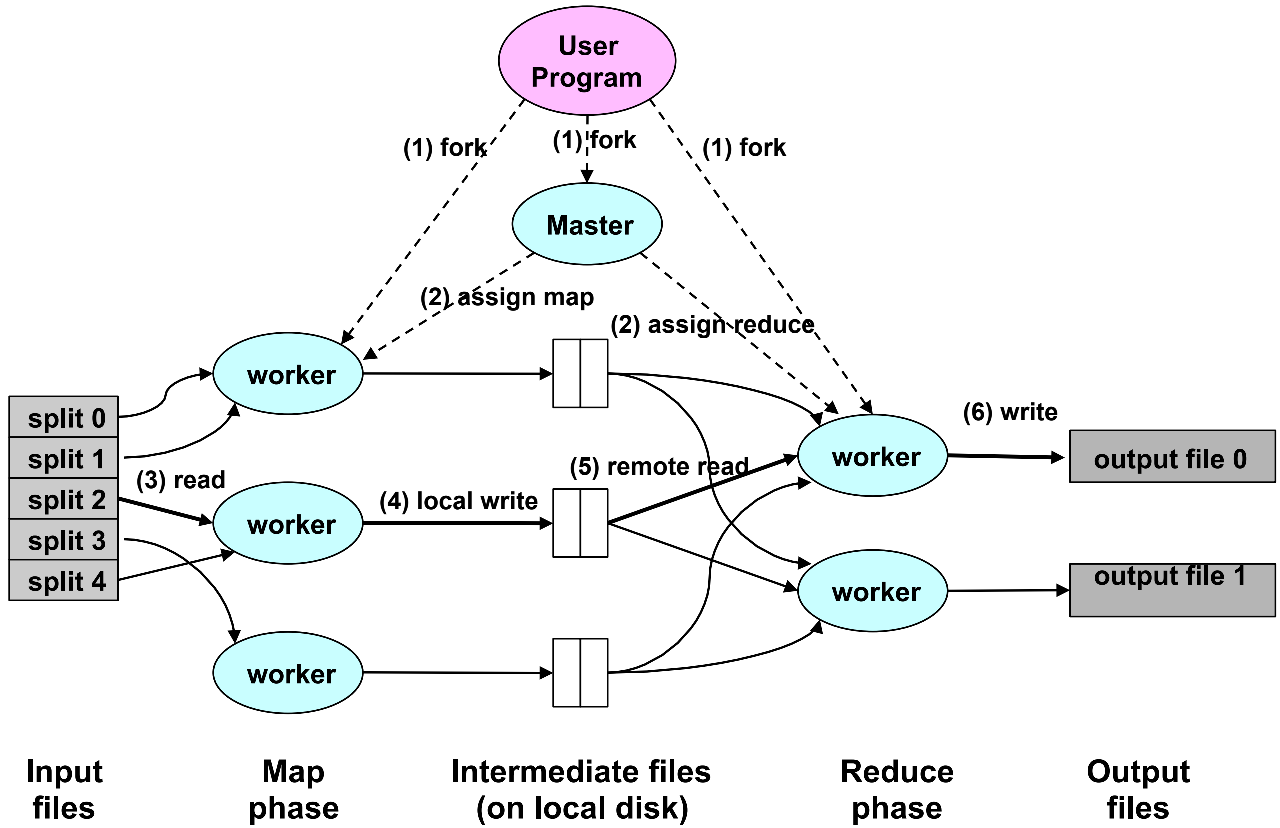
- The best part of all is that we do not need to write any additional code to take advantage of this!
- If a reduce function is both *commutative* and *associative*, then it can be used as a Combiner as well.
- conf.setCombinerClass(Reduce.class);

- If your Reducer itself cannot be used directly as a Combiner because of commutativity or associativity, you might still be able to write a third class to use as a Combiner for your job.

Outline

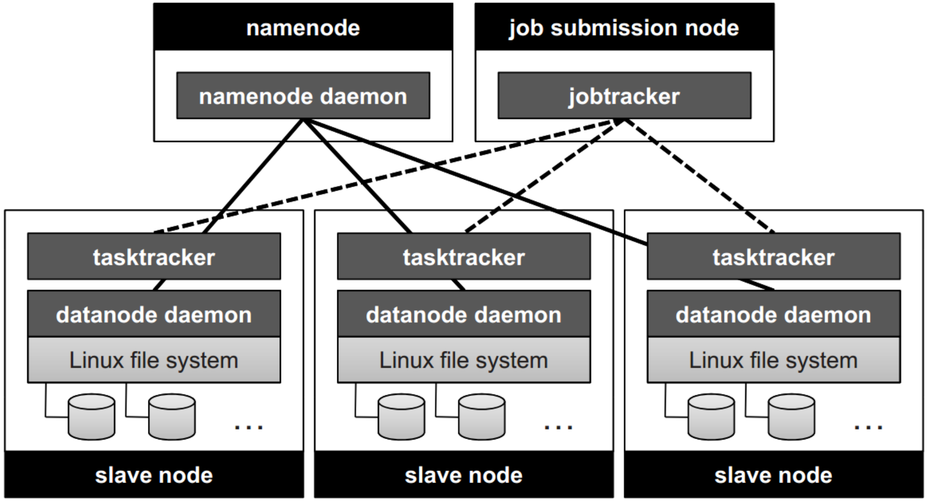
- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

Execution Overview of MapReduce Job



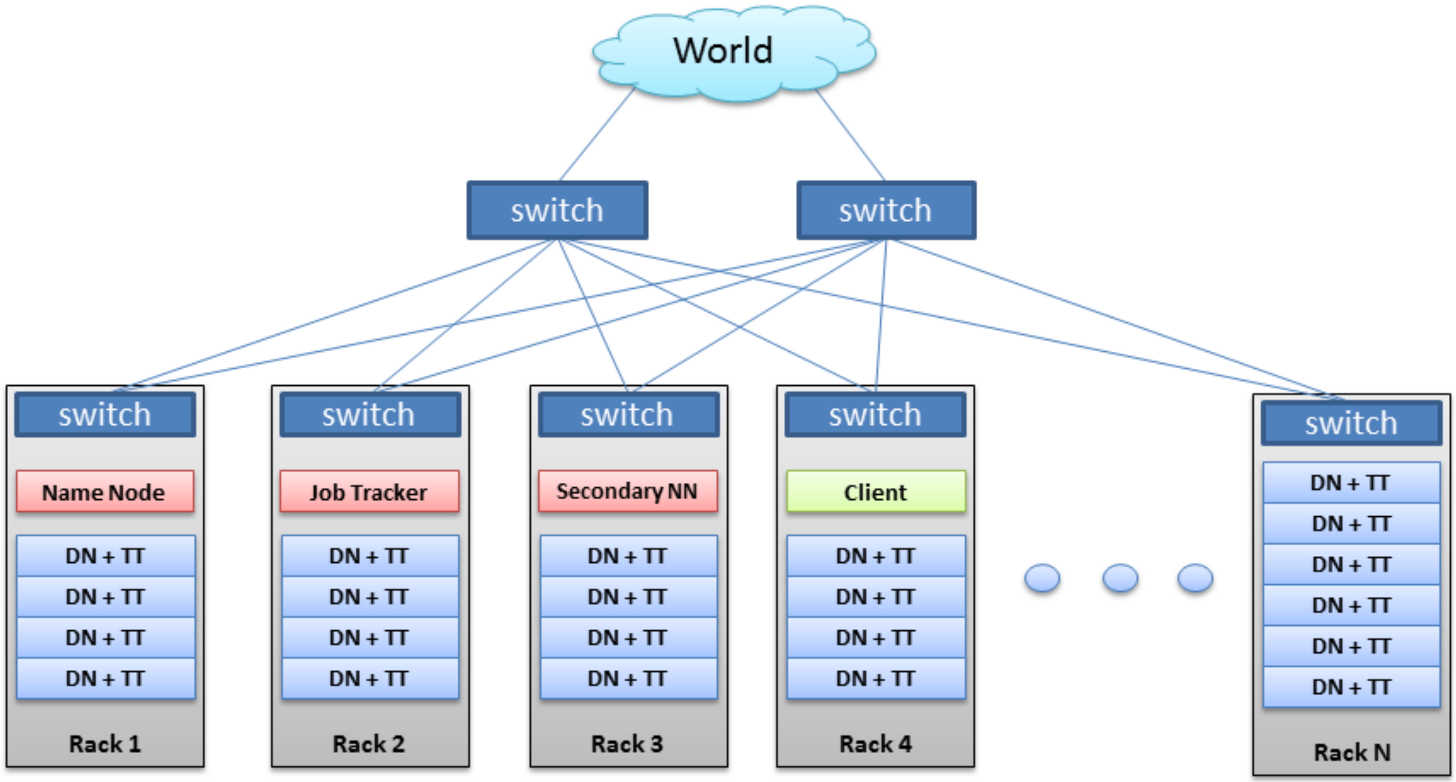
Master/Slaver Structure

- Master data structures
  - Task status: (*idle/in-progress/completed*);
  - Idle tasks get scheduled as workers become available;
  - When a map task completes, it sends the master the location and sizes of its *R*; intermediate files, one for each reducer;
  - Master pushes this info to reducers;
- Master pings workers periodically to detect failures



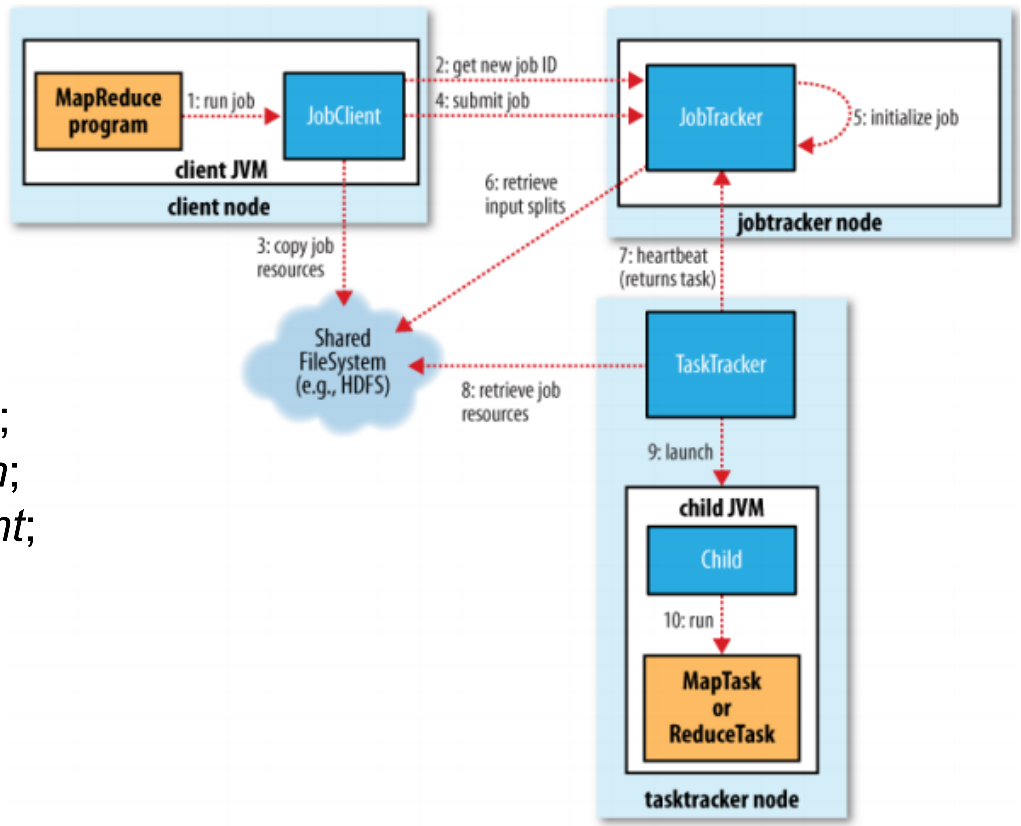
Hadoop Work Flow

- Hadoop cluster



Hadoop Work Flow (Cont'd)

- How Hadoop runs a MapReduce job



- *Job submission*;
- *Job Initialization*;
- *Task assignment*;

Hadoop Job Scheduling

- FIFO scheduler
- Fair Scheduler
- Capacity Scheduler

Fault Tolerance

- Hadoop achieves high fault tolerance
  - If the job is still in the mapping phase, other TaskTrackers will re-execute all map tasks previously run by the failed TaskTracker.
    - Even if a node has completed ten map tasks, the reducers may not have all copied their inputs from the output of those map tasks.
  - If the job is in the reducing phase, other TaskTrackers will re-execute all reduce tasks that were in progress on the failed TaskTracker.
    - Since finished reduce tasks will be written back to HDFS.
  - Master failure
    - MapReduce task is aborted and client is notified;
    - Master writes checkpoints periodically as failsafe;

Speculative Execution

- MapReduce abstraction accounts for "stragglers”
  - nodes that do not fail but take an unusually long time to complete the assigned work unit;
  - Possible Causes
    - Other tasks may be scheduled on machine;
    - Contention on network;
    - Degraded components;
  - Solution
    - Force tasks to run in isolation from one another;
    - The same input can be processed multiple times in parallel;
    - Occurs when most of the tasks in a job are coming to a close;
    - Whichever copy of a task finishes first becomes the definitive copy;
    - Other tasks executing the same copy speculatively will be abandoned and the output of them are discarded;

How many Map and Reduce tasks?

- *M* map tasks, *R* reduce tasks;
- Rule of thumb
  - Make *M* and *R* much larger than the number of nodes in cluster;
  - One DFS block per map is common;
  - Improve dynamic load balancing;
  - speed recovery from worker failure;
- Usually *R* is smaller than *M*
  - output is spread across *R* files;

Hadoop Programming

- What we need to do is writing map and reduce function
  - According to specific application;
  - Combiner function is optional;
- Hadoop setup & configuration
  - Local (Standalone) Mode;
  - Pseudo-Distributed Mode;
  - Fully-Distributed Mode;

Hadoop Programming (Cont’d)

- Supported Platforms
  - GNU/Linux (development platform + production platform)
  - Win32 (development platform)
- Required Software (for Linux)
  - Java 1.6.x
  - ssh (manage remote Hadoop daemons)
  - *Cygwin (for shell support in windows)*
- Hadoop download
  - http://apache.communilink.net/hadoop/common/
  - **1.2.X -** current stable version, 1.2 release

Outline

- Why Hadoop and MapReduce
- Case study
- Hadoop Distributed File System
- MapReduce
- MapReduce on Hadoop
- Conclusion and Reference

Beyond Word Count

- distributed pattern-based searching
- distributed sorting
- web link-graph reversal
- term-vector per host
- web access log stats
- inverted index construction
- document clustering
- machine learning
- statistical machine translation
- …

Comparison with others…

- RDBMS
  - Seek time is improving more slowly than transfer rate
    - Traditional B-Tree only works well when updating a small proportion of records;
    - MapReduce uses Sort/Merge to rebuild the database;

	Traditional	RDBMS MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

- Grid Computing
  - distribute the work across a cluster of machines and access a shared file system hosted by a Storage Area Network (SAN)
    - works well for predominantly compute-intensive jobs, but becomes a problem when nodes need to access larger data volumes;
    - MapReduce tries to collocate the data with the compute node, and is a shared-nothing architecture;

MapReduce2

- YARN (MapReduce2)
  - Yet Another Resource Negotiator (YARN)
  - Scalability bottlenecks for MapReduce
    - Very large clusters in the region of 4000 nodes and higher
  - splitting the re-sponsibilities of the jobtracker into separate entities.
    - a resource manager to manage the use of resources across the cluster;
    - an application master to manage the lifecycle of applications running on the cluster;
  - More on:
    - The Next Generation of Apache Hadoop MapReduce, Arun C. Murthy.[1]

1. "The Next Generation of Apache Hadoop MapReduce", http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen/

Controversy

- D. J. DeWitt, M. Stonebraker, "MapReduce: A major step backwards", http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/, (no longer available)
- D. J. DeWitt, M. Stonebraker, "MapReduce II", http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/, (no longer available)
- M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and Parallel DBMSs: Friends or Foes?," Communications of the ACM, vol. 53, iss. 1, pp. 64-71, 2010.
- A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data, New York, NY, USA, 2009, pp. 165-178.

Reference

- Hadoop official site
  - http://hadoop.apache.org/
- Yahoo! Hadoop Tutorial
  - http://developer.yahoo.com/hadoop/tutorial/
- Jimmy Lin and Chris Dyer, “Data-Intensive Text Processing with MapReduce”, Morgan & Claypool Publishers, 2010.
- Tom White, “Hadoop: The Definitive Guide”, O'Reilly Media, 2012.