CS 423
Operating System Design:

The Apache™ Hadoop® project

Why we talk about Hadoop?

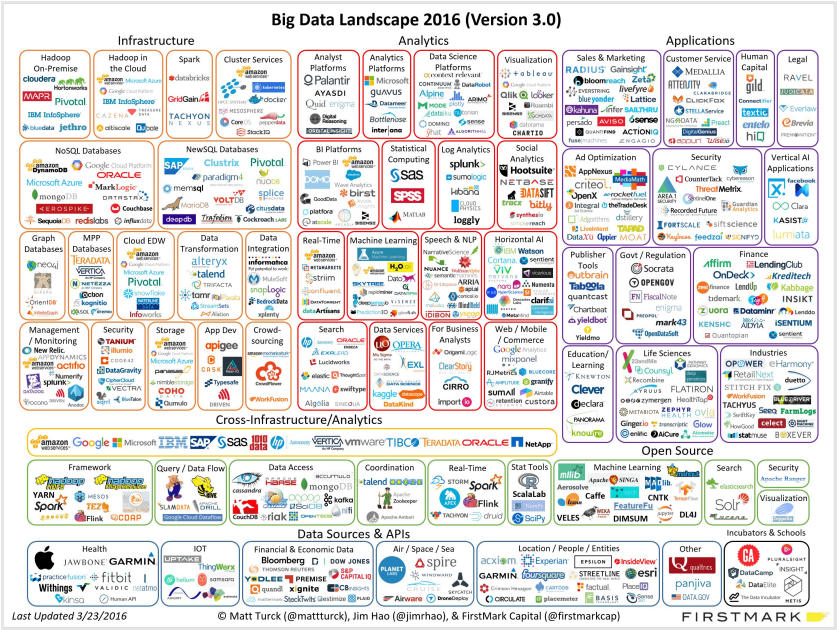
- A widely-used distributed system
  - Almost every IT company in the bay area uses some form of Hadoop
- Resemble many of today's distributed systems
  - HDFS – Distributed File System
  - MapReduce – Distributed Data Processing
  - YARN – Cluster Management
- Mature code base
  - Initial release: 2006 (13 years ago)
- Free and open source!
  - You can download and try it out!

What is Hadoop

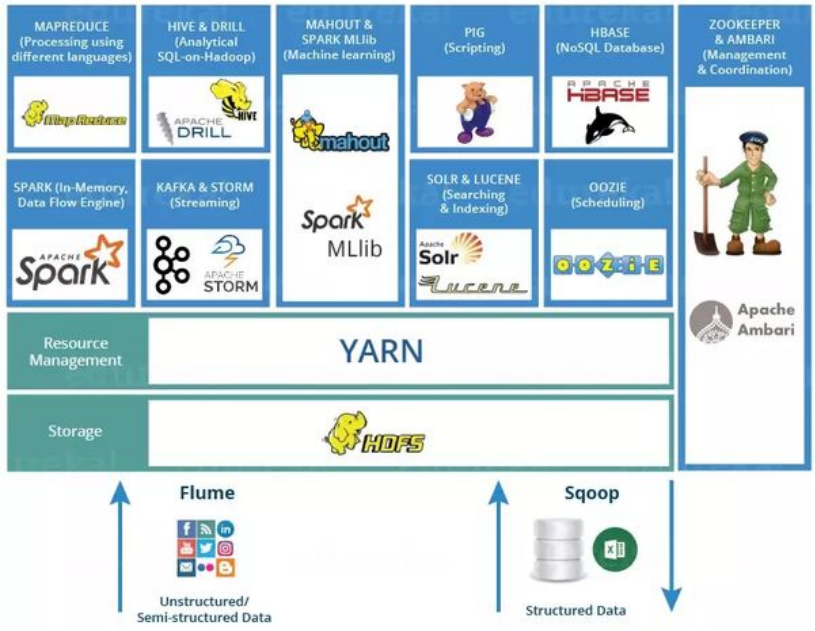
- HDFS (Hadoop Distributed File System)
- YARN (Hadoop's Cluster Resource Management)
  - Yet Another resource Negotiator
- MapReduce (Distributed Data Processing)

- New component
  - Ozone (Hadoop's Object Store)
  - Submarine (Distributed Machine Learning Engine)

A key component of the big data ecosystem



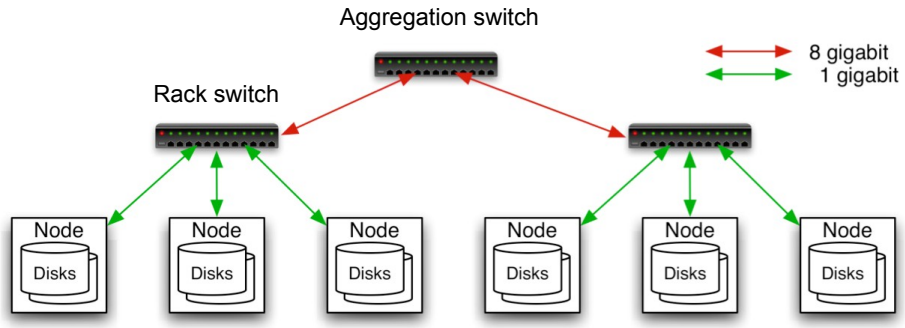
A key component of the big data ecosystem



History

- Co-founders, Doug Cutting and Mike Cafarella,
  - The genesis of Hadoop was the Google File System paper that was published in October 2003.
  - This paper spawned another one from Google – "MapReduce: Simplified Data Processing on Large Clusters
  - Doug Cutting was working at Yahoo! at the time,
  - Hadoop is Doug's son's toy elephant.

Typical Hadoop Architecture



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 30-40 nodes/rack
  - Uplink from rack is 8 gigabit
  - Rack-internal is 1 gigabit

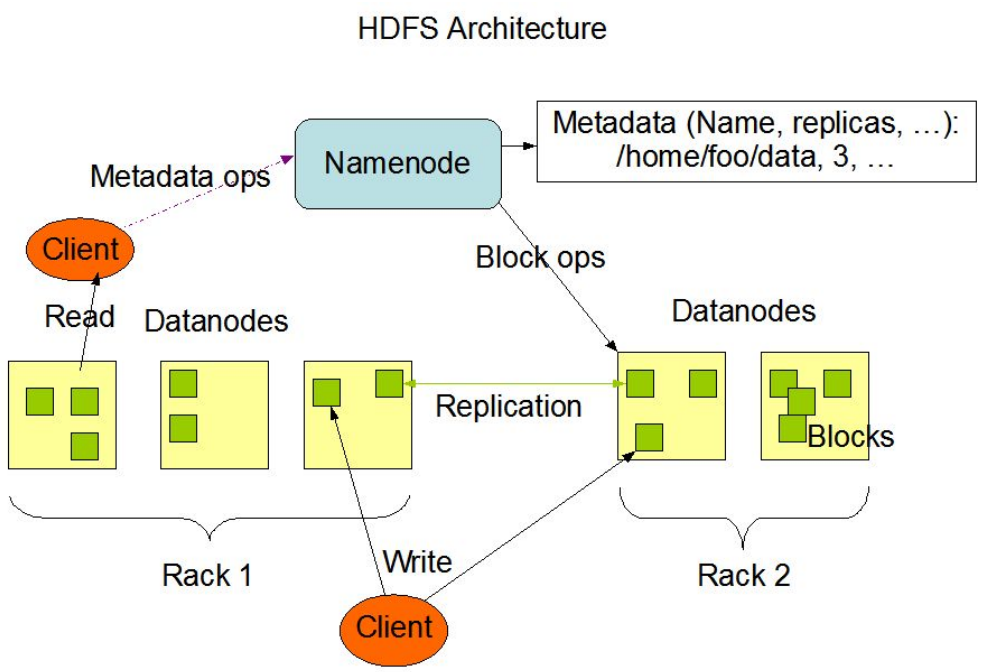
HDFS (Hadoop Distributed File System)

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

HDFS (Hadoop Distributed File System)

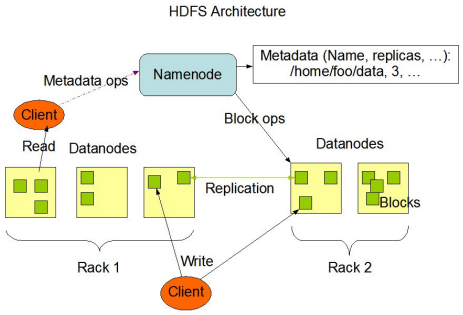
- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

HDFS Architecture



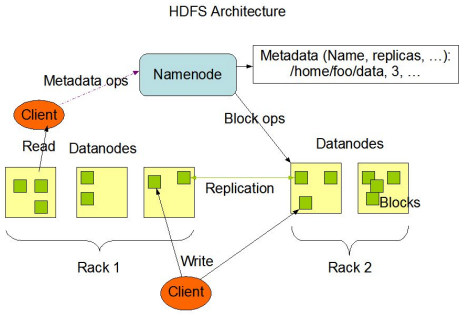
- Master/slave architecture
- A single NameNode
- A number of DataNodes
- Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes

NameNode



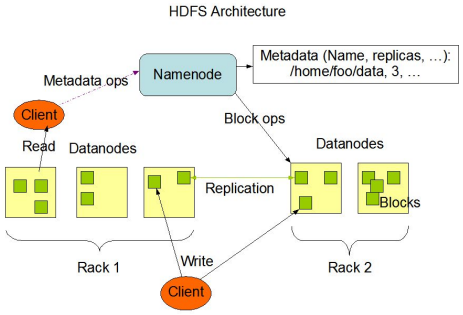
- Manages File System Namespace
  - Executes file system namespace operations like opening, closing, and renaming files and directories
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- The existence of a single NameNode in a cluster greatly simplifies the architecture of the system.
- The NameNode is the arbitrator and repository for all HDFS metadata.

NameNode Metadata



- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

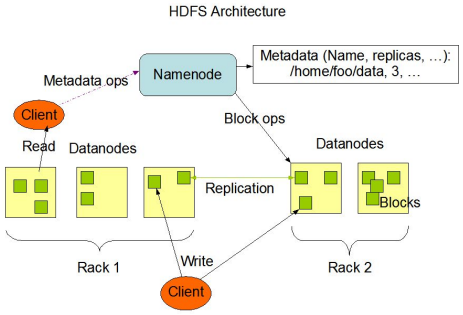
DataNode



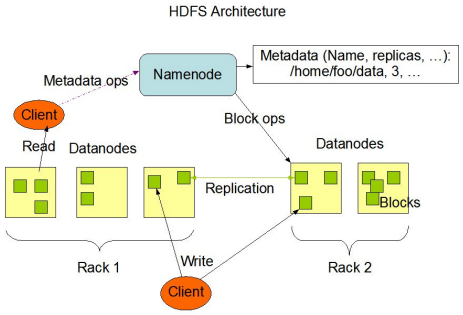
- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas
- Would like to make this policy pluggable



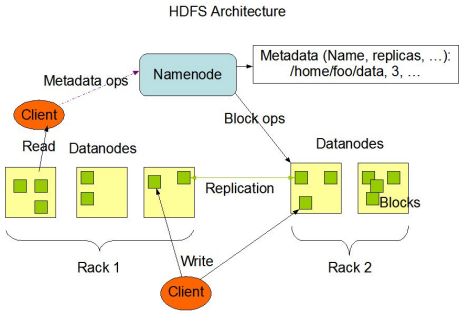
Heartbeats



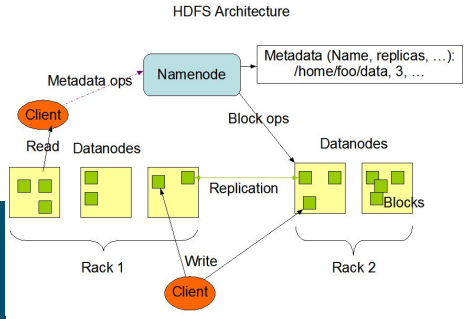
- DataNodes send hearbeat to the NameNode
- NameNode uses heartbeats to detect DataNode failure
- A network partition can cause a subset of DataNodes to lose connectivity with the NameNode.
- The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them.

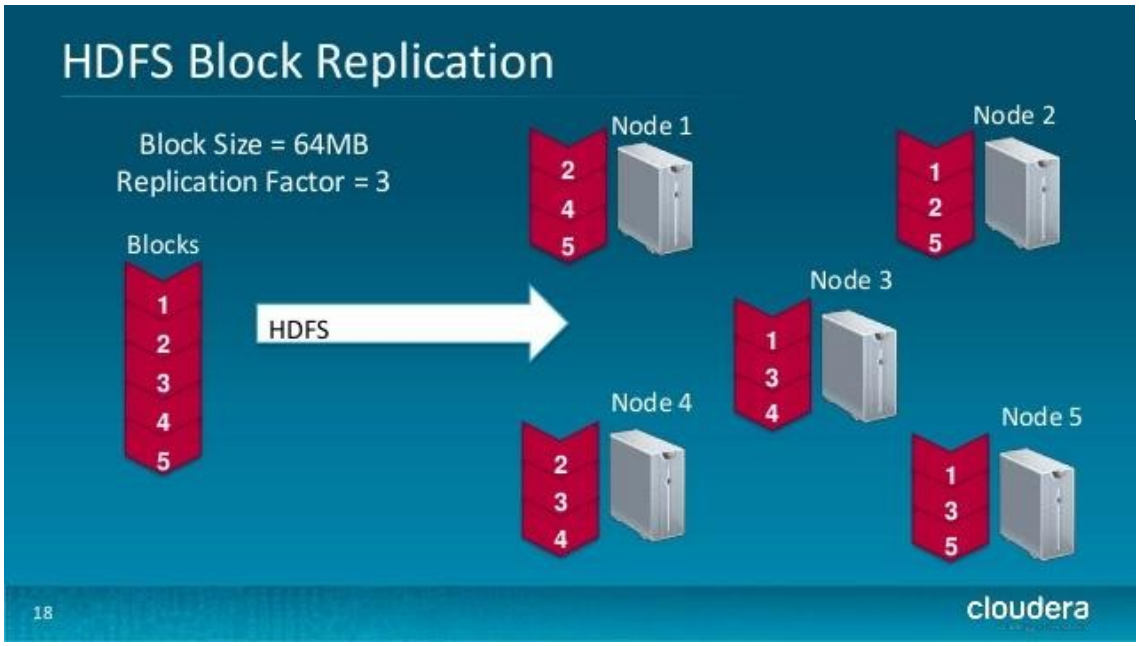
Data Correctness

- Use Checksums to validate data
  - Use CRC32
- File Creation
  - Client computes checksum per 512 bytes
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas



Block Replication





Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next node in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

Rebalancer

- Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool

Secondary NameNode

- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode
  - Transaction Log on NameNode is purged

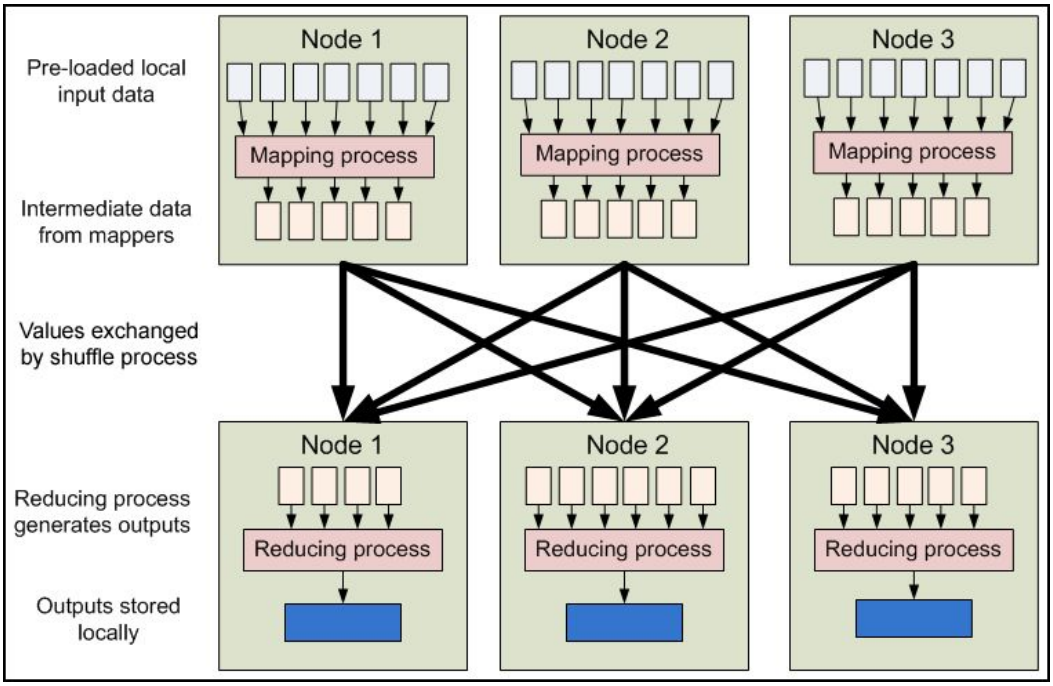
User Interface

- Commads for HDFS User:
  - hadoop dfs -mkdir /foodir
  - hadoop dfs -cat /foodir/myfile.txt
  - hadoop dfs -rm /foodir/myfile.txt
- Commands for HDFS Administrator
  - hadoop dfsadmin -report
  - hadoop dfsadmin -decommision datanodename
- Web Interface
  - http://host:port/dfshealth.jsp

MapReduce

- MapReduce is a programming model for efficient distributed computing
- It works like a Unix pipeline
  - cat input | grep | sort | uniq -c | cat > output
  - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building

MapReduce - Dataflow



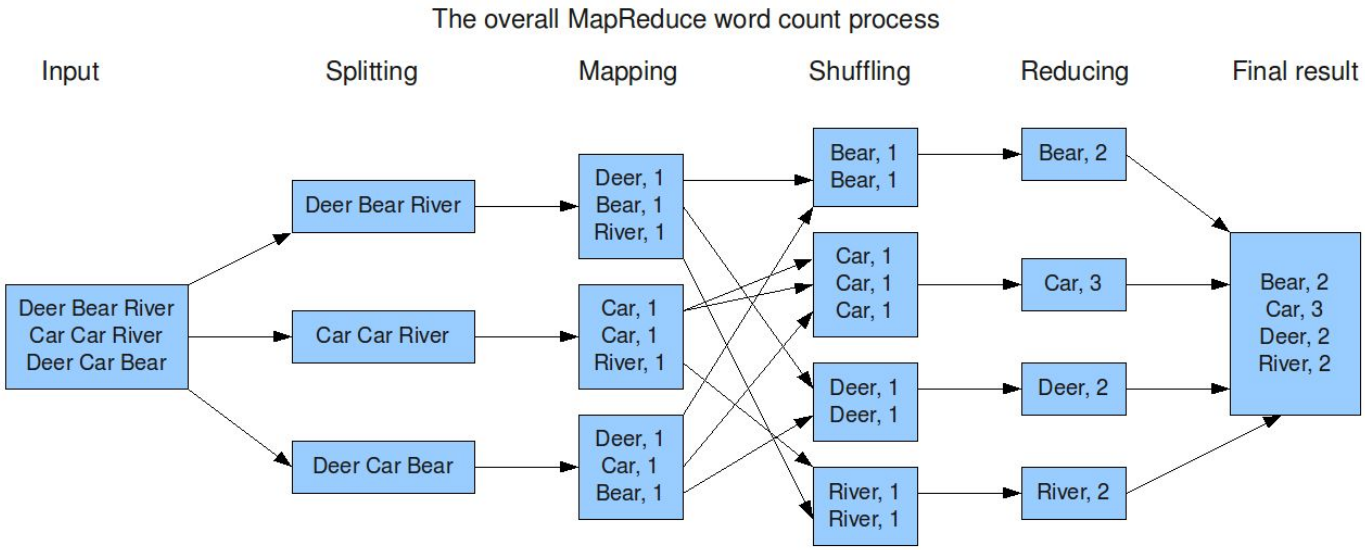
MapReduce - Features

- Fine grained Map and Reduce tasks
  - Improved load balancing
  - Faster recovery from failed tasks
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - With large data, bandwidth to data is a problem
  - Map-Reduce + HDFS is a very effective solution
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

Word Count Example

- Mapper
  - Input: value: lines of text of input
  - Output: key: word, value: 1
- Reducer
  - Input: key: word, value: set of counts
  - Output: key: word, value: sum
- Launching program
  - Defines this job
  - Submits job to cluster

Word Count Dataflow



Word Count Mapper

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

  private Text word  = new Text();


  public static void map(LongWritable key, Text value, OutputCollector<Text,IntWritable> output, Reporter reporter) throws
     IOException {  String line = value.toString();

     StringTokenizer = new StringTokenizer(line);
     while(tokenizer.hasNext()) {
        word.set(tokenizer.nextToken(

        ));  output.collect(word,one);

        }
     }
 }
```

Word Count Reducer

```
public static class Reduce extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable> {

public static void map(Text key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable>
output, Reporter reporter) throws IOException {
        int sum = 0;

        while(values.hasNext()) {

            sum += values.next().get();

        }

        output.collect(key, new IntWritable(sum));

    }

  }
```

Word Count Example

- Jobs are controlled by configuring *JobConfs*
- JobConfs are maps from attribute names to string values
- The framework defines attributes to control how the job is executed
  - conf.set(“mapred.job.name”, “MyApp”);
- Applications can add arbitrary values to the JobConf
  - conf.set(“my.string”, “foo”);
  - conf.set(“my.integer”, 12);
- JobConf is available to all tasks

Putting it all together

- Create a launching program for your application
- The launching program configures:
  - The *Mapper* and *Reducer* to use
  - The output key and value types (input types are inferred from the *InputFormat*)
  - The locations for your input and output
- The launching program then submits the job and typically waits for it to complete

Putting it all together

```
JobConf conf = new JobConf(WordCount.class);
conf.setJobName("wordcount");

conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.cla
ss);

conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducer(Reduce.class);

conf.setInputFormat(TextInputFormat.class);
Conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new
Path(args[0]));
```

How many Maps and Reducers

- Maps
  - Usually as many as the number of HDFS blocks being processed, this is the default
  - Else the number of maps can be specified as a hint
  - The number of maps can also be controlled by specifying the *minimum split size*

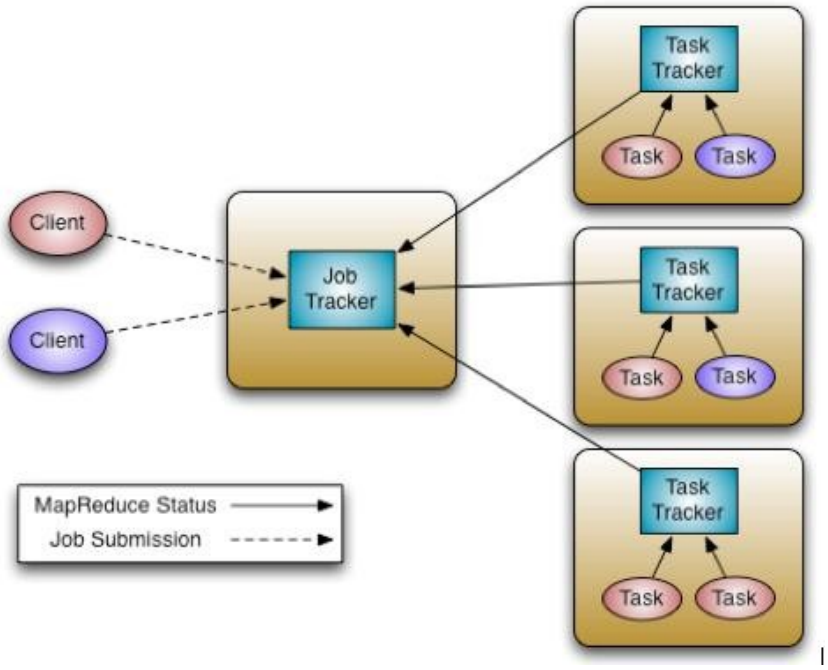
- Reducers
  - Also specifiable
  - Sometimes, a specific reducer number is needed to run a certain reducer algorithm.

YARN

- Yet Another Resource Negotiator
- Remedies the scalability issues of “classic” MapReduce
- Is more of a general purpose framework of which classic MapReduce is one application.

Classic MapReduce

- Job Tracker
  - Manages cluster resources and
    job scheduling
- Task Tracker
  - Per-node agent
  - Manage tasks

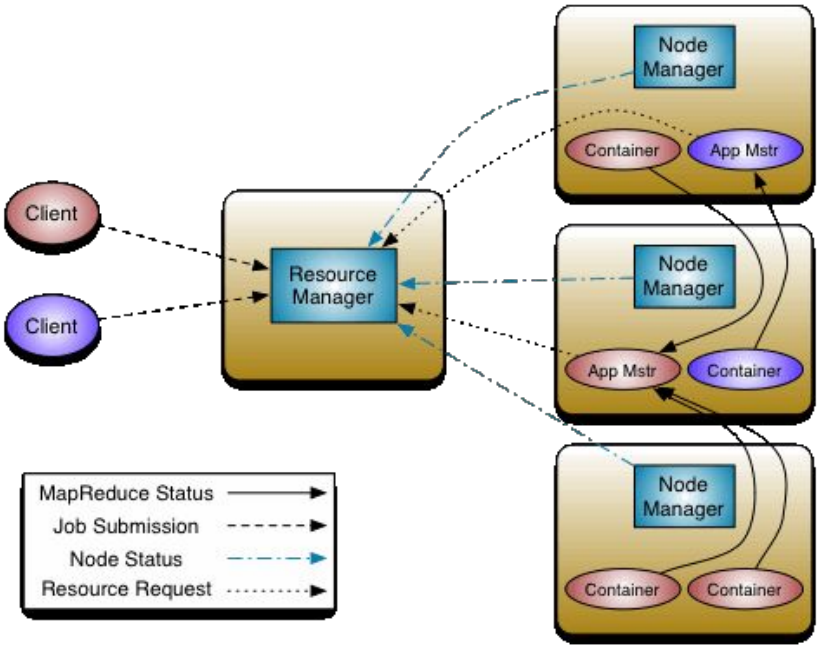


Classic MapReduce Limitations

- Scability
  - Maximum cluster size ~4000 nodes
  - Maximum concurrent tasks ~40,000
  - Coarse synchronization in JobTracker
- Availability
  - Failures kills all queued and running tasks
- Hard partition of resources into map and reduce slots
  - Low resource utilization

YARN Architecture

- Scability
  - Cluster 6,000-10,000 machines
  - 100,000 concurrent tasks
  - 10,000 concurrent jobs



YARN

- Splits up the two major functions of JobTracker
  - Global Resource Manager - Cluster resource management
  - Application Master - Job scheduling and monitoring (one per application). The Application Master negotiates resource containers from the Scheduler, tracking their status and monitoring for progress. Application Master itself runs as a normal container.
- Tasktracker
  - NodeManager (NM) - A new per-node slave is responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting to the Resource Manager.
- YARN maintains compatibility with existing MapReduce applications and users.

Classic MapReduce vs YARN

- Fault Tolerance and Availability
  - Resource Manager
    - No single point of failure – state saved in ZooKeeper
    - Application Masters are restarted automatically on RM restart
  - Application Master
    - Optional failover via application-specific checkpoint
    - MapReduce applications pick up where they left off via state saved in HDFS
- Compatibility
  - Protocols are wire-compatible
  - Old clients can talk to new servers
  - Rolling upgrades

Classic MapReduce vs YARN

- Support for programming paradigms other than MapReduce
  - Tez – Generic framework to run a complex DAG
  - HBase on YARN(HOYA)
  - Machine Learning: Spark
  - Graph processing: Giraph
  - Real-time processing: Storm
  - Enabled by allowing the use of paradigm-specific application master
  - Run all on the same Hadoop cluster!