Data management in the cloud using Hadoop

Murat Kantarcioglu

Outline

- Hadoop - Basics
- HDFS
  - Goals
  - Architecture
  - Other functions
- MapReduce
  - Basics
  - Word Count Example
  - Handy tools
  - Finding shortest path example
- Related Apache sub-projects (Pig, Hbase,Hive)

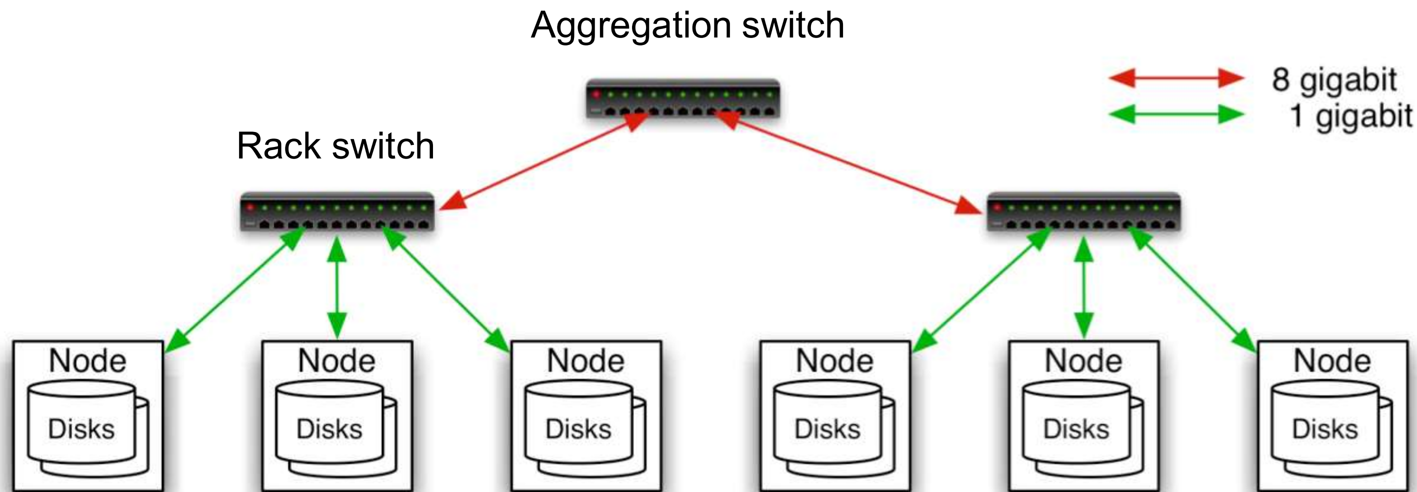
Hadoop - Why ?

- Need to process huge datasets on large clusters of computers
- Very expensive to build reliability into each application
- Nodes fail every day
  - Failure is expected, rather than exceptional
  - The number of nodes in a cluster is not constant
- Need a common infrastructure
  - Efficient, reliable, easy to use
  - Open Source, Apache Licence version of Google File System

Who uses Hadoop?

- Amazon/A9
- Facebook
- Google
  - It has GFS
- New York Times
- Veoh
- Yahoo! was the first big company to use Hadoop.
- …. many more
- Cloudera
  - Similar to Redhat business model.
  - Added services on Hadoop

Commodity Hardware



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 30-40 nodes/rack
  - Uplink from rack is 3-4 gigabit
  - Rack-internal is 1 gigabit

Hadoop Distributed File System (HDFS)

Original Slides by
Dhruba Borthakur
Apache Hadoop Project Management Committee

Goals of HDFS

- Very Large Distributed File System
  - Initial 4 goals (Surpassed): 10K nodes, 100M 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
    - Remember moving large data is an important bottleneck.
  - Provides very high aggregate bandwidth

Distributed File System

- Single Namespace for entire cluster: bottleneck
  – HDFS Federation; Multiple namenodes, namespaces

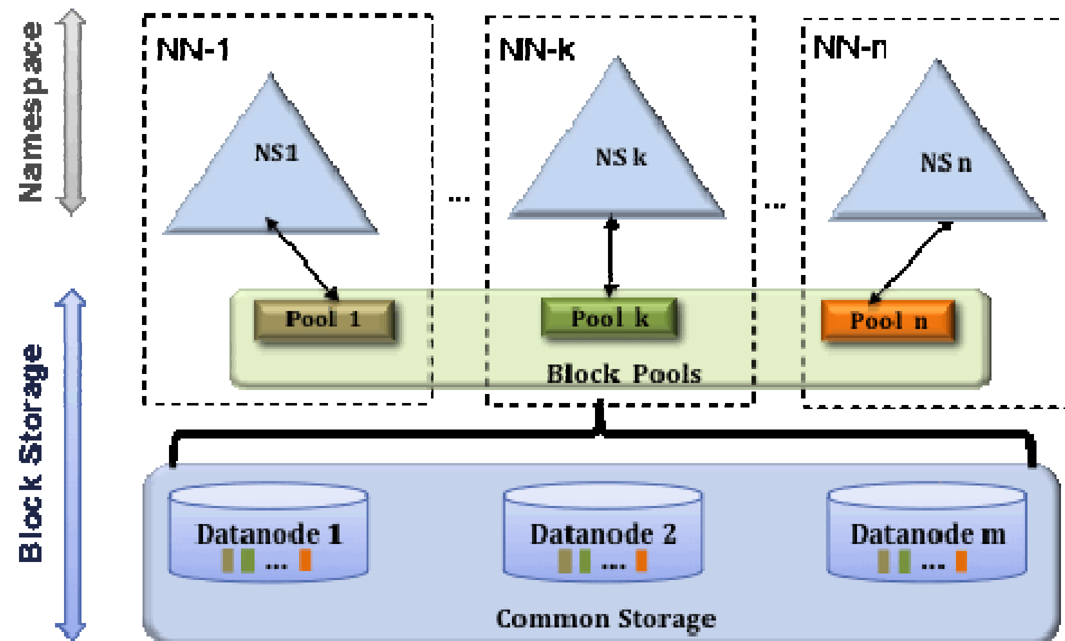


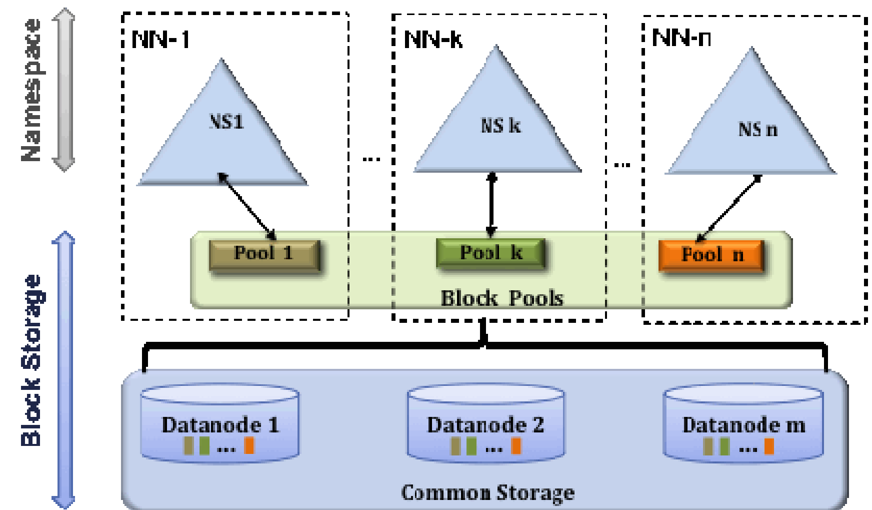
Figure: https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/Federation.html

Distributed File System

- Block Pool: a set of blocks belong to a single namespace.
- Datanodes store blocks for all the block pools in the cluster. Each Block Pool is managed independently.

- Namespace generates Block IDs for new blocks without the need for coordination with the other namespaces.

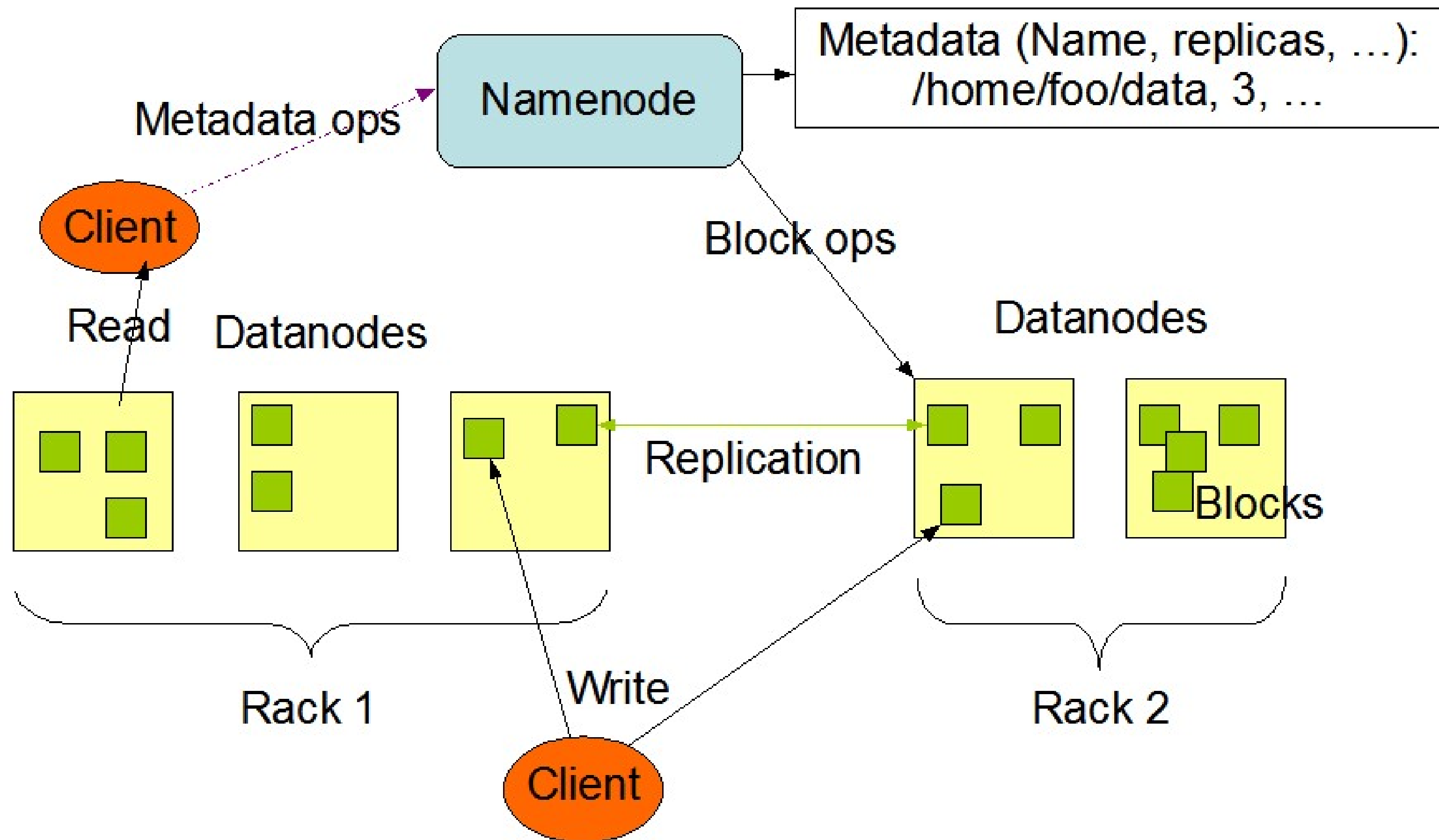
- Namenode/space is deleted, the corresponding block pool at the Datanodes is deleted.



Distributed File System

- 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



Functions of a NameNode (Master)

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
  - Keeps a directory tree of all files in the system
  - Usually has more RAM than all other nodes
  - Single point of failure
- Cluster Configuration Management
- Replication Engine for Blocks

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

Heartbeats

- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

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

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Federated HDFS is a solution to the single namenode issues

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/removed
  - In an unbalanced cluster, data read/write requests become very busy on some data nodes and some data nodes are under utilized.
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool – not triggered automatically

Source: http://hadooptutorial.info/hdfs-rebalance/

Secondary NameNode

- FsImage: snapshot of the filesystem when namenode started
- Tlog: sequence of changes made to the filesystem after namenode started
- Logs become too large in time.
- Secondary Namenode is a checkpoint in HDFS.
- A helper node for namenode. Also known as checkpoint node in the community.

Secondary NameNode

- Regularly updated with logs from the NameNode
- Updates the FsImage

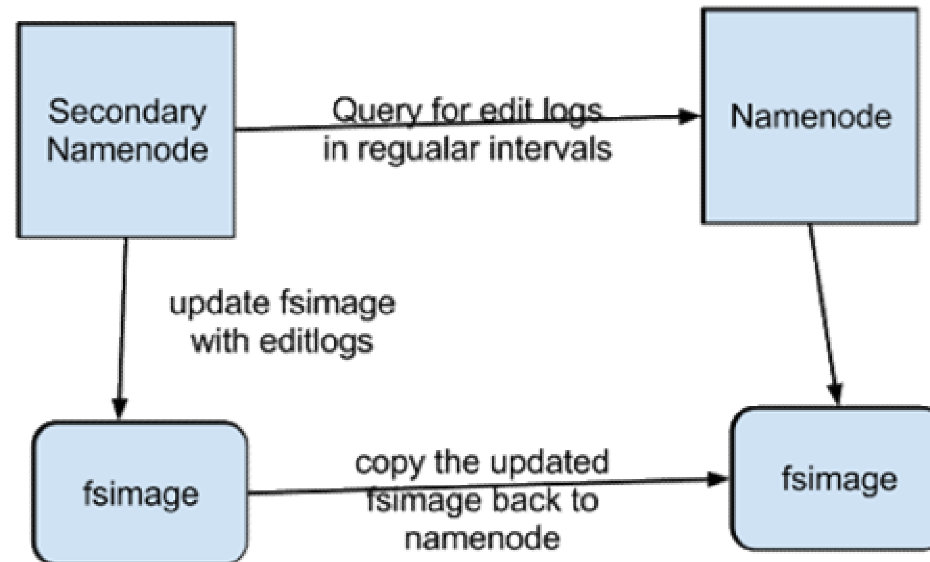


Figure: http://blog.madhukaraphatak.com/secondary-namenode---what-it-really-do/

User Interface

- Commands 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

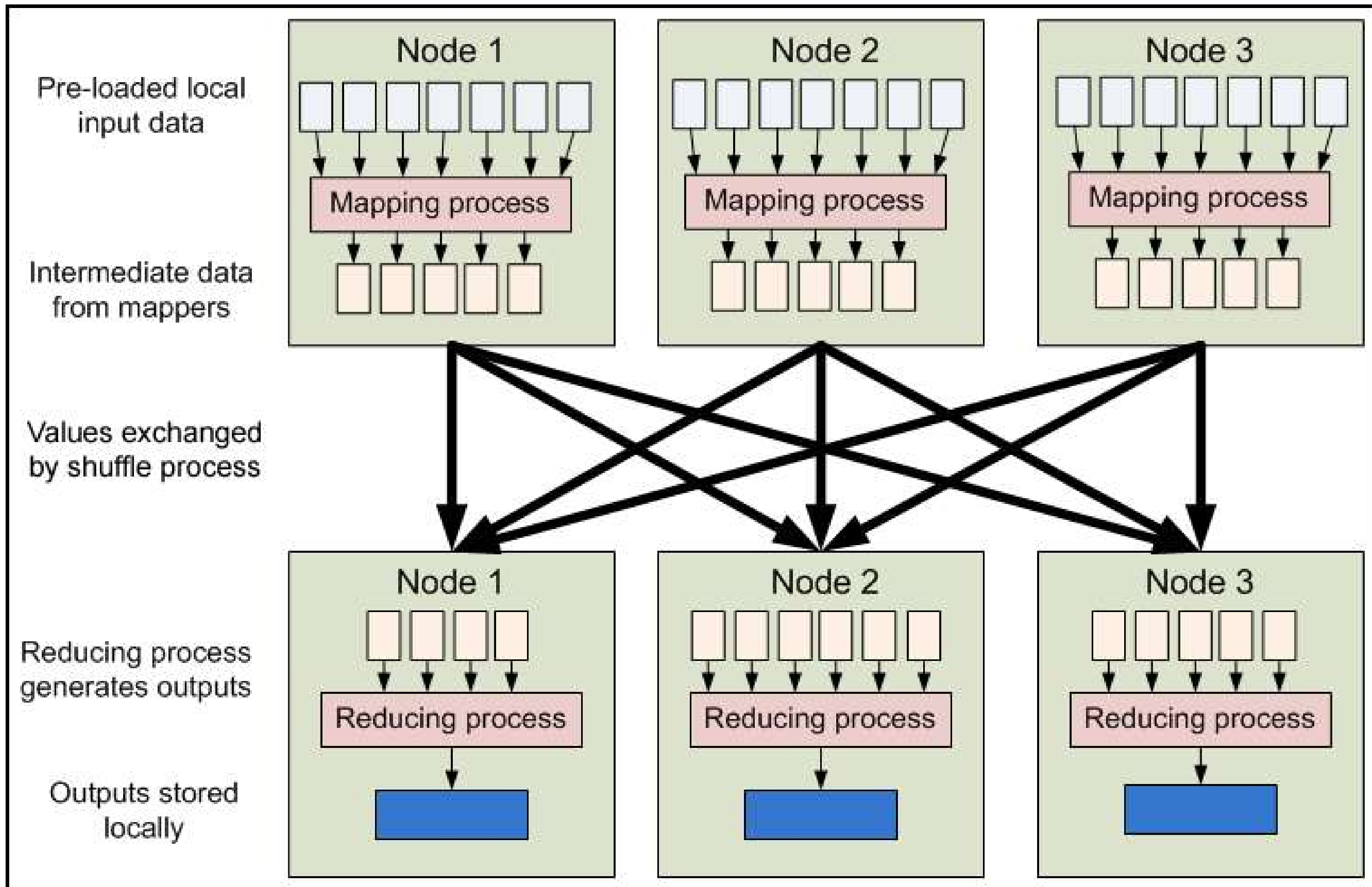
MapReduce

Original Slides by
Owen O’Malley (Yahoo!)
&
Christophe Bisciglia, Aaron Kimball & Sierra Michells-Slettvet

MapReduce - What?

- 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



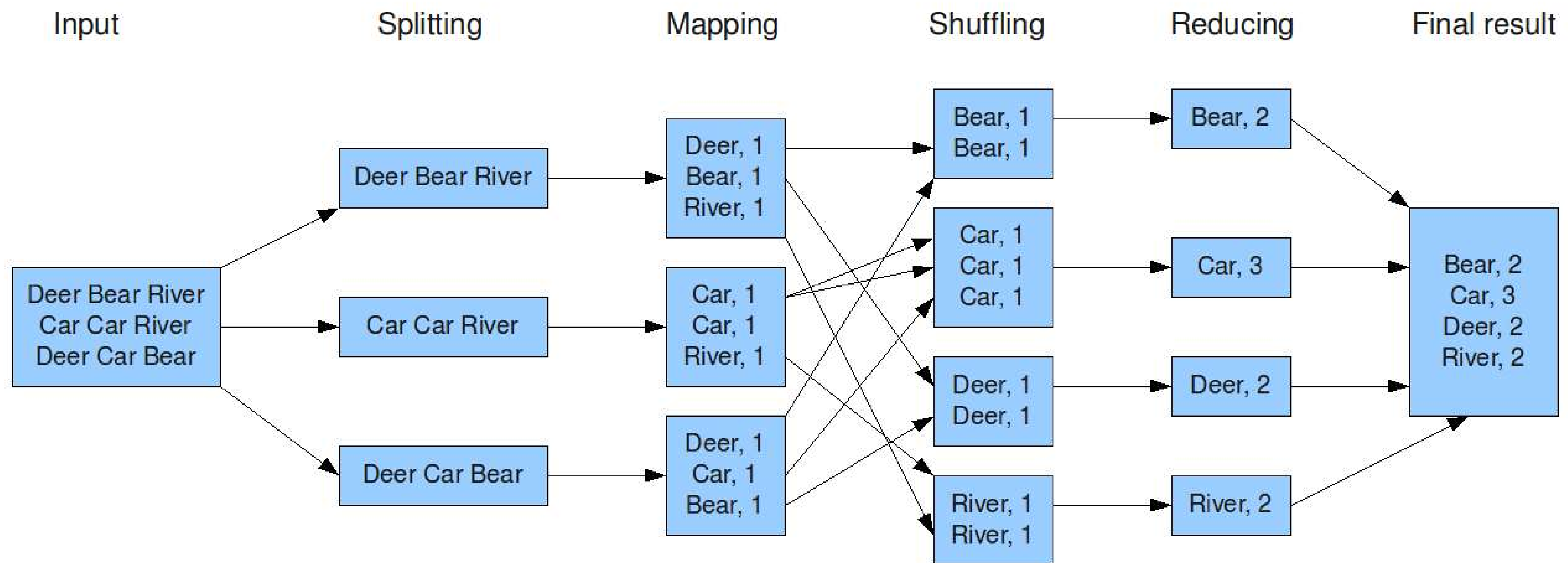
MapReduce - Features

- 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 reduce(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.class);

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]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);
```

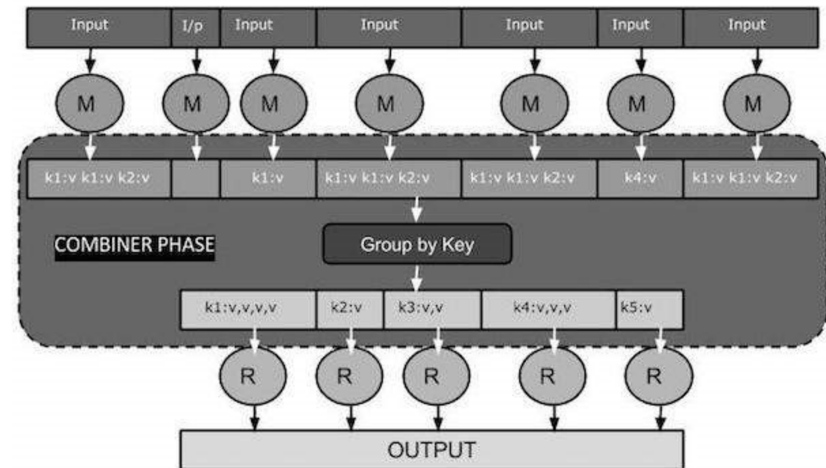


Image: https://www.tutorialspoint.com/map_reduce/map_reduce_combiners.htm

Input and Output Formats

- For outputs, <key, value> pairs are used exclusively.
- A Map/Reduce may specify how it's input is to be read by specifying an *InputFormat* to be used
- A Map/Reduce may specify how it's output is to be written by specifying an *OutputFormat* to be used
- These default to *TextInputFormat* and *TextOutputFormat*, which process line-based text data
- Another common choice is *SequenceFileInputFormat* and *SequenceFileOutputFormat* for binary data

How many Maps and Reduces

- 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*
  - Split is a logical division of the input data while block is a physical division of data. Each split is gets a mapper.
  - The actual sizes of the map inputs are computed by:
    - *max(min(block_size,data/#maps), min_split_size)*
- Reduces
  - Unless the amount of data being processed is small
    - *0.95*num_nodes*mapred.tasktracker.tasks.maximum*

Some handy tools

- Partitioners
- Combiners
- Compression
- Counters
- Zero Reduces
- Distributed File Cache
- Tool

Finding the Shortest Path: Intuition

- We can define the solution to this problem inductively
  - DistanceTo(startNode) = 0
  - For all nodes *n* directly reachable from startNode, DistanceTo(*n*) = 1
  - For all nodes *n* reachable from some other set of nodes *S*,

    DistanceTo(*n*) = 1 + min(DistanceTo(*m*), $m \in S$)

From Intuition to Algorithm

- A map task receives a node *n* as a key, and (*D*, *points-to*) as its value
  - *D* is the distance to the node from the start
  - *points-to* is a list of nodes reachable from *n*
- $\forall p \in$ points-to, emit (p, D+1)
- Reduces task gathers possible distances to a given *p* and selects the minimum one

What This Gives Us

- This MapReduce task can advance the known frontier by one hop
- To perform the whole BFS, a non-MapReduce component then feeds the output of this step back into the MapReduce task for another iteration
  - Problem: Where'd the *points-to* list go?
  - Solution: Mapper emits (*n*, *points-to*) as well

Blow-up and Termination

- This algorithm starts from one node
- Subsequent iterations include many more nodes of the graph as the frontier advances
- Does this ever terminate?
  - Yes! Eventually, routes between nodes will stop being discovered and no better distances will be found. When distance is the same, we stop
  - Mapper should emit (n,D) to ensure that “current distance” is carried into the reducer

Extensions to Map-Reduce

- Main applications for Workflow systems
  - Representing a cascade of multiple-map reduce jobs
  - Complex distributed tasks
- Generally more efficient than running multiple map-reduce sequentially.
  - Writing results to hard disks could be problematic.
  - Potential pipelining optimizations.