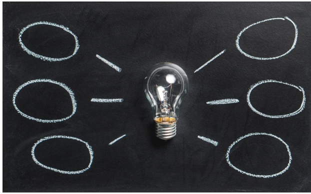
Julian Kunkel

Distributed Storage and Processing with Hadoop



Learning Objectives

- Describe the architecture and features of Apache Hadoop
- Formulate simple algorithms using the MapReduce programming model
- Justify architectural decisions made in Apache Hadoop
- Sketch the execution phases of MapReduce and describe their behavior
- Describe limitations of Hadoop1 and the benefits of Hadoop2 with TEZ
- Sketch the parallel file access performed by MapReduce jobs

Outline

Hadoop Version 1

- Apache Hadoop: Framework for scalable processing of data
  - Based on Google's MapReduce paper
  - Still used in (big data) industry
  - Good example of a distributed system
- Consists of:
  - Hadoop distributed file system (HDFS)
  - MapReduce execution engine: schedules tasks on HDFS
- Why should we combine storage and execution paradigms?
  - Execution exploits data locality to avoid network data transfer
  - Ship compute to data and not (big) data to compute
- A complete ecosystem has been layered on top of MapReduce

Hadoop Distributed File System (HDFS)

- Goal: Reliable storage on commodity-of-the-shelf hardware
- Implemented in Java
- Provides single-writer, multiple-reader concurrency model
- Has demonstrated scalability to 200 PB of storage and 4500 servers [12]

Features

- Hiearchical namespace (with UNIX/ACL permissions)
- High availability and automatic recovery
- Replication of data (pipelined write)
- Rack-awareness (for performance and high availability)
- Parallel file access

Hadoop File System Shell

Overview

- Invoke via: `hadoop fs <command> <args>`
  - Example: `hadoop fs -ls hdfs://serverName/`

HDFS command overview

- Read files: `cat`, `tail`, `get`, `getmerge` (useful!)
- Write files: `put`, `appendToFile`, `moveFromLocal`
- Permissions: `chmod`, `chgrp`, ..., `getfacl`
- Management: `ls`, `rm`, `rmdir`, `mkdir`, `df`, `du`, `find`, `cp`, `mv`, `stat`, `touchz`

Special commands

- `distcp`: map-reduce parallelized copy command between clusters
- `checksum`
- `expunge` (clear trash)
- `setrep` (change replication factor)

Architectural Components

- Namenode: Central manager for the file system namespace
  - Filenames, permissions
  - Information about file block (location)
  - For HA, a secondary NameNode backups data
- DataNode: Provide storage for objects (data)
  - Directly communicates with other DataNodes for replication
- TaskTracker: accept and runs map, reduce and shuffle
  - Provides a number of **slots** for tasks (logical CPUs)
  - A **task** is tried to be scheduled on a slot of the machine hosting data
  - If all slots are occupied, run the task on the same rack
- JobTracker: Central manager for running MapReduce jobs
  - For HA, a secondary JobTracker backups data
- Tools to access and manage the file system (e.g., rebalancing)

High-Level Perspective

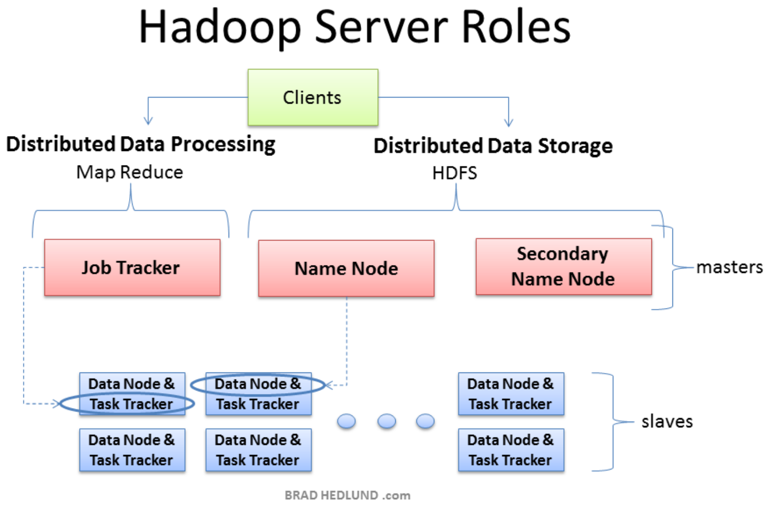


Figure: Source: B. Hedlund. [15]

System-Level Perspective of Hadoop Clusters

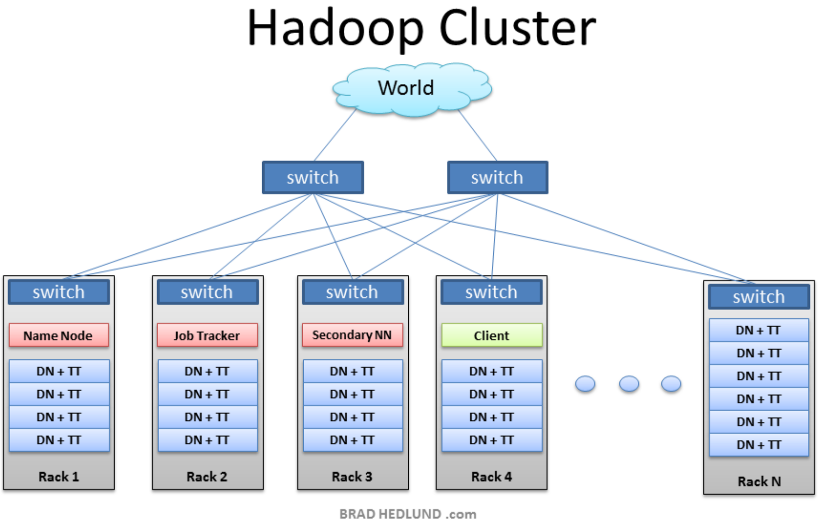


Figure: Source: B. Hedlund. [15]

Mapping of In-Memory Data Structures to Streams

(De)Serialization

- Data structure (in memory) ⇒ byte stream (on storage) ⇒ data structure
- Serialization is the process of creating a byte stream from a data structure
- De-serialization creates a data structure in memory from the byte stream
- Byte streams can be transferred via network or stored on block storage

Serialization frameworks

- Provide serialization code for basic types
- Support writing of datatype-specific serializers
- Examples:
  - Java: Apache Avro[1], Kryo [40]
  - Python: Pickle
  - R: `serialize()`, `unserialize()` (functions for objects)
  - Apache Thrift supports multiple languages
- Requirements: Performance, platform independence

[1] https://avro.apache.org/

Excerpt of File Formats Supported by MapReduce

Mapping to Storage: Files are split into **blocks**

- A typical block size is 64 MiB
- Blocks are distributed across nodes
- Blocks may be compressed individually
- Hadoop provides record readers for various file formats

Text files

- Delimiters can be choosen
- Splittable at newlines (only decompressed files)

```
This is a simple file.\n
With three lines – \n
this is the end.
```

Comma-separated values (CSV)

- No header supported but JSON records are supported
- Does not support block compression

```
'max', 12.4, 10 \n
'john', 10.0, 2 \n
```

File Formats (2)

Sequence files

- Flat binary file for key/value pairs
- Supports splitting in HDFS using a synchronization marker
- Optional block compression for values (and keys)
- Widely used within Hadoop as internal structure

MapFile [21]

- Extends the sequence file
- Provides an index for keys

Avro

- Apache Avro's serialization system format
- Self-describing data format[2], allows inference of schema
  - Schema can also be changed upon read
- Enables exchange of data types between tools

⇒ Popular file format for Hadoop ecosystem

[2] A self-describing format contains information (metadata) needed to understand its contained data, e.g., variable/field names, data types

The HDFS I/O Path

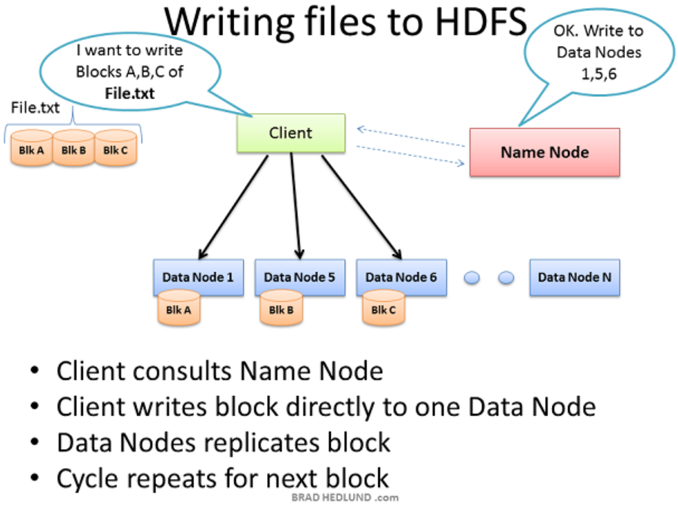


Figure: Source: B. Hedlund. [15]

The HDFS Write Path

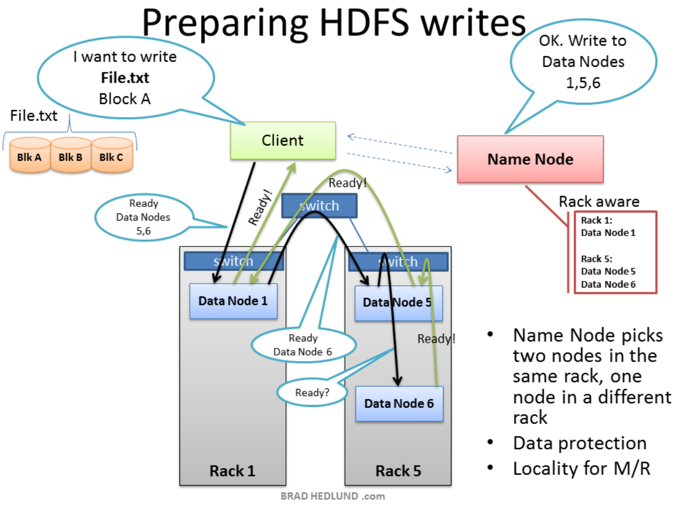


Figure: Source: B. Hedlund [15]

The HDFS Write Path

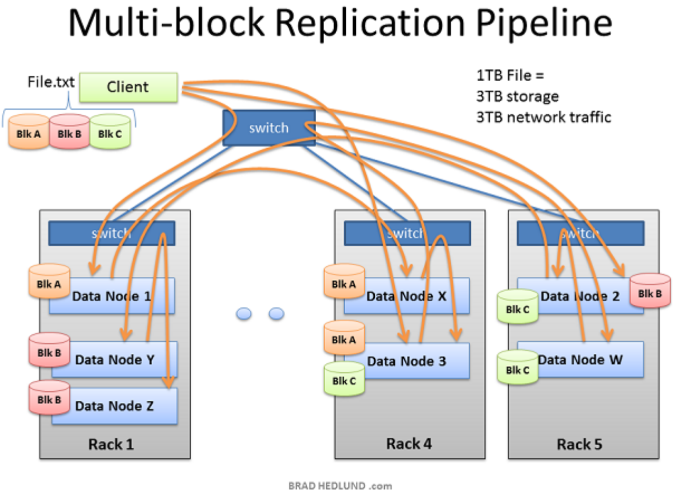


Figure: Source: B. Hedlund [15]

The HDFS Read Path

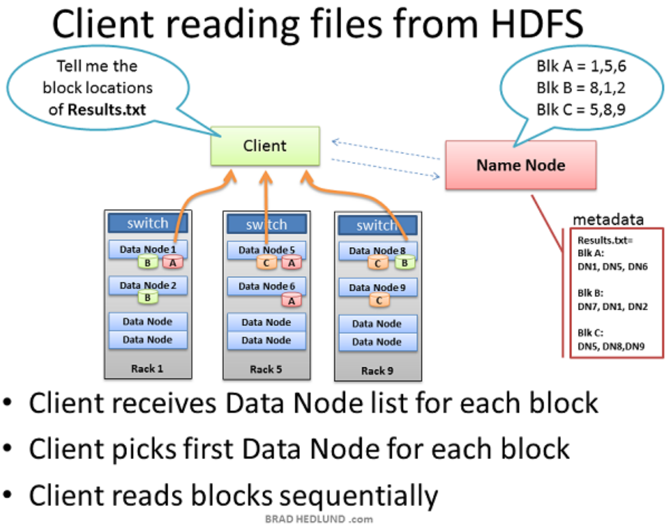


Figure: Source: B. Hedlund [15]

Name Node and High Availability

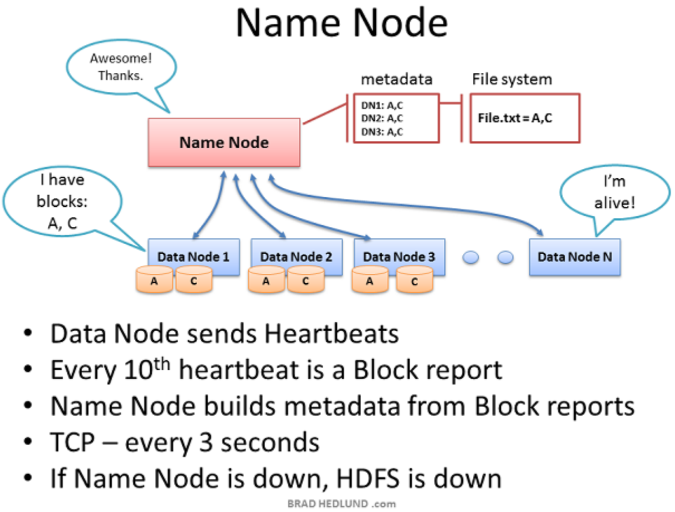


Figure: Source: B. Hedlund. [15]

Name Node and High Availability

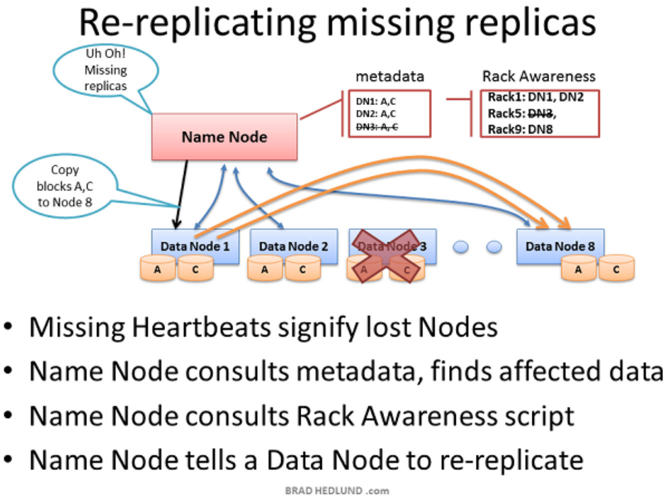


Figure: Source: B. Hedlund. [15]

Outline

Map Reduce Execution Paradigm

Idea: Apply a processing pipeline consisting of map and reduce operations

1. Map: filter and convert input records (pos, data) to tuples (key, value)
2. Reduce: receives all tuples with the same key (key, list<value>)

- Hadoop takes care of reading input, distributing (key,value) to reduce
- Types for key, value & format, records depend on the configuration

Example: WordCount [10]: Count word frequency in large texts

```
map(key, text): # input: key=position, text=line
  for each word in text:
    Emit(word,1) # outputs: key/value

reduce(key, list of values): # input: key == word, our mapper output
  count = 0
  for each v in values:
    count += v
  Emit(key, count) # it is possible to emit multiple (key, value) pairs here
```

Map Reduce Execution: Aggregation of Tables

Example from [17]

Goal: aggregate a CSV file by grouping certain entries

Country	State	City	Population
USA,	CA,	Su,	12
USA,	CA,	SA,	42
USA,	MO,	XY,	23
USA,	MO,	AB,	10

$\Rightarrow$

Country	State	Population
USA	CA	54
USA	MO	33

Algorithm

```
map(key, line):
  (county, state, city, population) = line.split(',')
  Emit( (country, state), population )

reduce(key, values): # key=(country,state) values=list of populations
  count = 0
  for each v in values:
    count += v
  Emit(key, count)
```

Group Work

- Sketch the MapReduce algorithm for aggregating at the same time: Country+State, Country, and summing all !
- Time: 10 min
- Organization: breakout groups - please use your mic or chat

Example

```
Country State  City Population          Country State   Population
USA,    CA,    Su,  12                  USA     CA      54
USA,    CA,    SA,  42                  USA     MO      33
USA,    MO,    XY,  23          ⇒       USA     ?       87
USA,    MO,    AB,  10                  GER     ?       20
GER,    BW,    HD,  20                   ?      ?      107
```

- Think about what the "?" should be, if anything

Phases of MapReduce Execution

Phases of MapReduce Execution

1. Distribute code (JAR files)
2. Determine files to read, blocks and file splits, assign mappers to splits and slots
3. Map: Invoke (local) map functions
4. Combine: Perform a local reduction by the key
5. Shuffle: Sort by the key, exchange data
6. Partition: Partition key space among reducers (typically via hashing)
7. Reduce: Invoke reducers
8. Write output, each reducer writes to its own file[3]

[3] Use `hadoop fs -getmerge <HDFS DIR> file.txt` to retrieve merged output

Parallel Access to Files

- A MapReduce job processes all files in a directory
  - Provides **parallelism on the file level**, each file is read independently
- MapReduce jobs process records that are grouped in **input splits**
  - Input splits == logical organization of blocks
  - Each input split is processed by one **mapper** (local processing preferred)
  - Processing for records spanning blocks
    - Skip partial records at the beginning of a split
    - For truncated records, read data from a remote
  - Input splitting (intelligence) depends on the file format
- File formats that are not splittable must be avoided
  - e.g., XML, JSON Files, compressed text files
  - They enforce sequential read by one mapper
- Usage of file formats depends on the tools to query data

Mapping of Data Blocks to Input Splits [23]

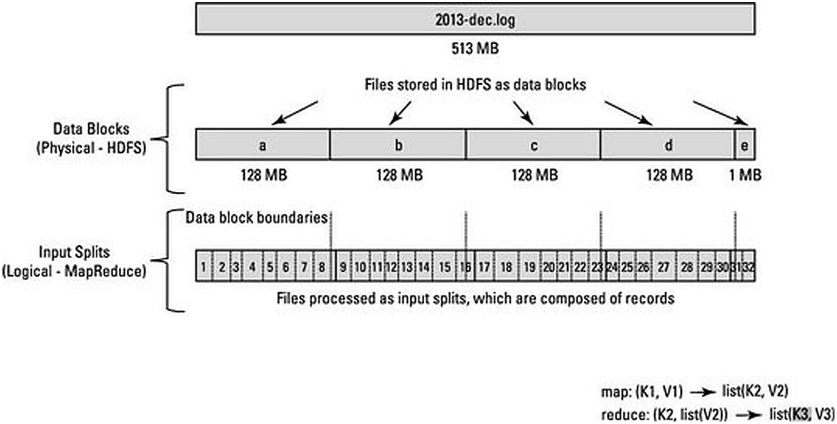


Figure: Source: [23]

Execution of MapReduce – the Big Picture

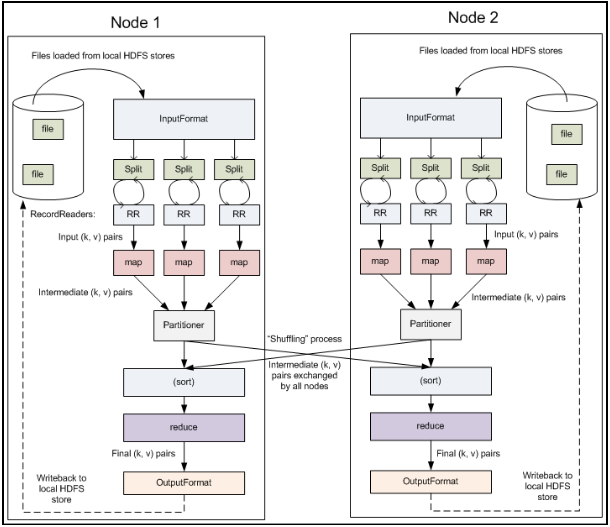


Figure: Source: jcdenton. [16]

Execution of MapReduce on HDFS – the Combiner

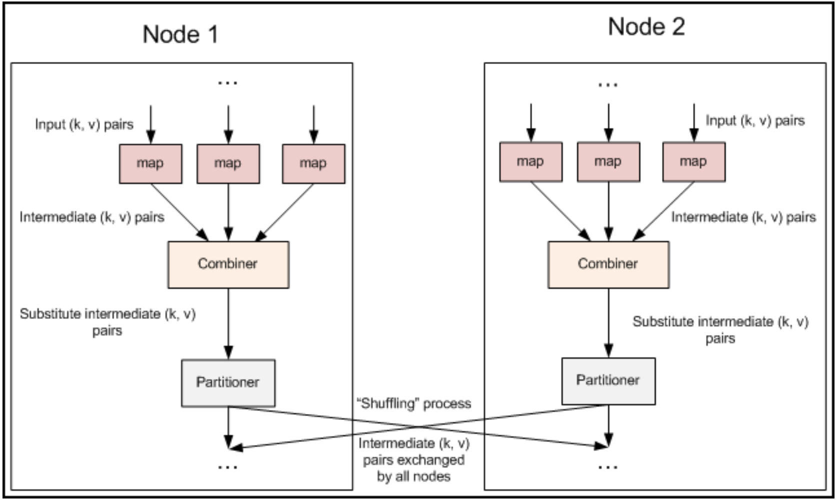


Figure: Source: jcdenton. [16]

Execution of MapReduce Tasks on Hadoop [14]

Steps in the execution of tasks

1. Client submits a job to the JobTracker
2. JobTracker identifies the location of data via the NameNode
3. JobTracker locates TaskTracker nodes with free slots close to the data
4. JobTracker starts tasks on the TaskTracker nodes
5. Monitoring of TaskTrack nodes
   - If heartbeat signals are missed, work is rescheduled on another TaskTracker
   - A TaskTracker will notify the JobTracker when a task fails
6. The JobTracker constantly updates its status
   - Clients can query this information

Execution of MapReduce

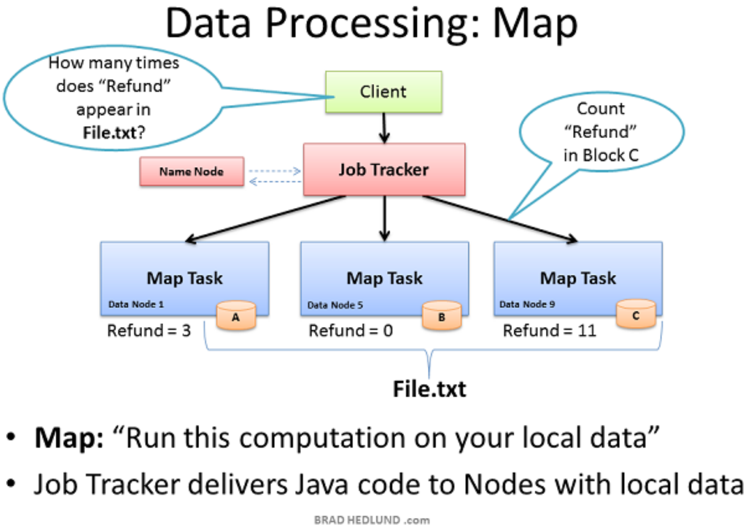


Figure: Source: B. Hedlund. [15]

Execution of MapReduce

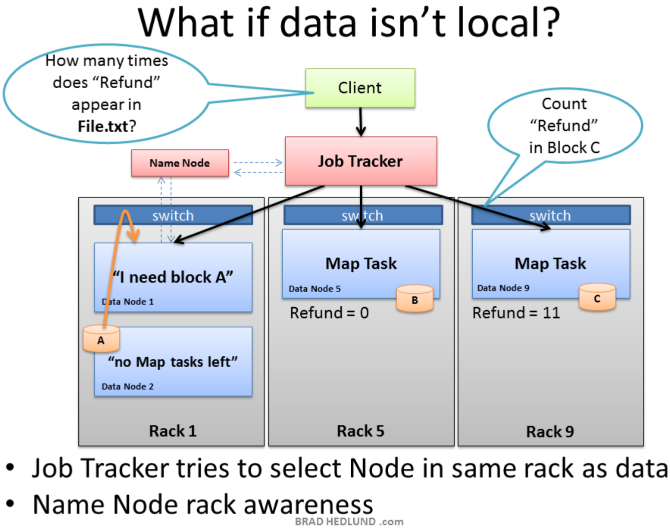


Figure: Source: B. Hedlund. [15]

Execution of MapReduce

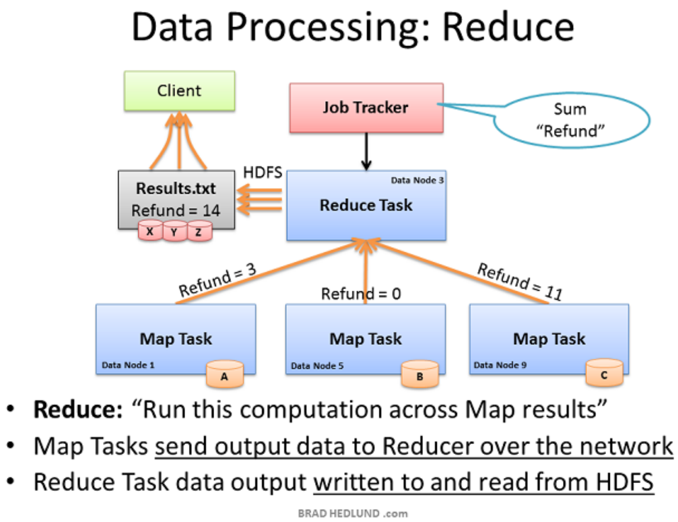


Figure: Source: B. Hedlund. [15]

Outline

Hadoop 2, the Next Generation [12]

- Goal: real-time and interactive processing of events
- Introduction of YARN: Yet Another Resource Negotiator
- Supports of classical MapReduce and, via TEZ, DAG of tasks
- Support for NFS access to HDFS data
- Compatability to Hadoop v1
- High-availability, federation and snapshots for HDFS

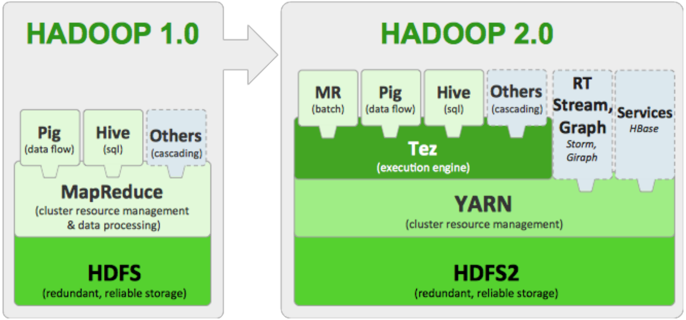


Figure: Source: Apache Hadoop 2 is now GA. Hortonworks. [12]

System Architecture

Yarn modularizes JobTracker functionality

1. Resource management
2. Job scheduling/execution inclusive monitoring

Data computation framework

- Applications are executed in containers
- ResourceManager component (global daemon)
  - Partitiones resources and schedules applications
  - Scheduler: distributes resources among applications
  - ApplicationsManager: accepts jobs, negotiates execution of AppMaster
- Per-node NodeManager: manages and monitors local resources
- Per-application ApplicationMaster
  - Framework-specific library
  - Negotiates container resources with ResourceManager
  - Works with Scheduler/NodeManager to execute and monitor tasks

YARN System Architecture

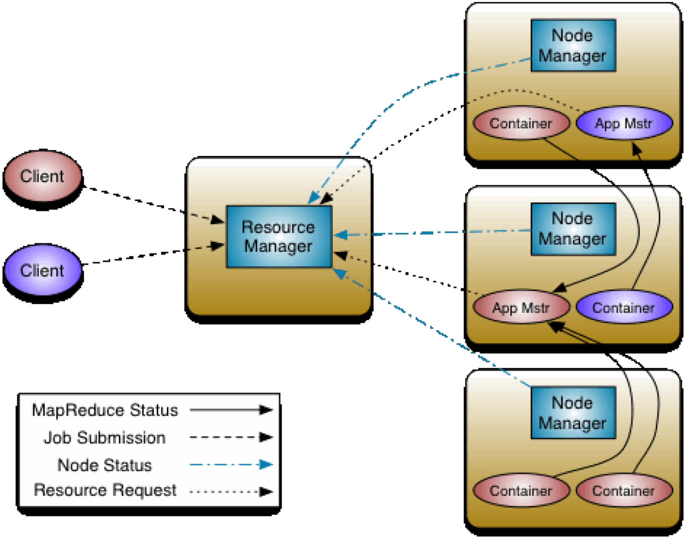


Figure: Source: Apache Hadoop NextGen [18]

Outline

TEZ Execution Engine

- TEZ: Hindi for “speed”
- Allow modelling and execution of data processing logic
  - Directed acyclic graph (DAG) of tasks
  - Vertex with input (dependencies) and output edges
- VertexManager defines parallelism and resolves dependencies

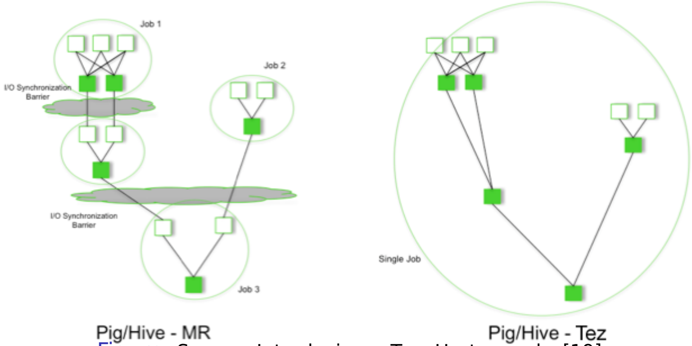


Figure: Source: Introducing... Tez. Hortonworks [19]

TEZ Example DAG [20]

```
// Define DAG
DAG dag = new DAG();
// Define Vertex, which class to execute
Vertex Map1 = new Vertex(Processor.class);
// Define Edge
Edge edge = Edge(Map1, Reduce2,
  SCATTER_GATHER, // Distribution of data from source to target(s)
  PERSISTED,  // Persistency of data
  SEQUENTIAL, // Scheduling: either concurrent or sequential execution
  Output.class, Input.class);
// Connect edges with vertex
dag.addVertex(Map1).addEdge(edge)...
```

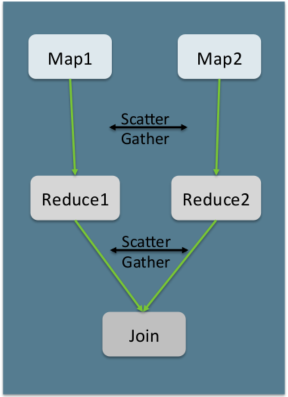


Figure: Source: Apache Tez. H. Shah [20]

TEZ DAG API

Edge properties define the connection between producer and consumer tasks in the DAG

- **Data movement – Defines routing of data between tasks**
  - **One-To-One** : Data from the i^{th} producer task routes to the i^{th} consumer task.
  - **Broadcast** : Data from a producer task routes to all consumer tasks.
  - **Scatter-Gather** : Producer tasks scatter data into shards and consumer tasks gather the data. The i^{th} shard from all producer tasks routes to the i^{th} consumer task.
- **Scheduling – Defines when a consumer task is scheduled**
  - **Sequential** : Consumer task may be scheduled after a producer task completes.
  - **Concurrent** : Consumer task must be co-scheduled with a producer task.
- **Data source – Defines the lifetime/reliability of a task output**
  - **Persisted** : Output will be available after the task exits. Output may be lost later on.
  - **Persisted-Reliable** : Output is reliably stored and will always be available
  - **Ephemeral** : Output is available only while the producer task is running

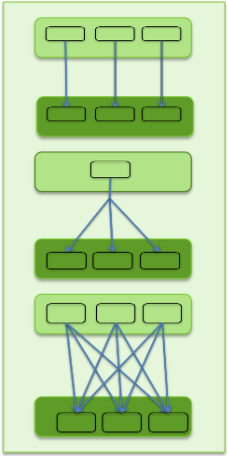


Figure: Source: Apache Tez. H. Shah [20]

TEZ Dynamic Graph Reconfiguration

- Reconfigure dataflow graph based on data sizes and target load
- Controlled by vertex management modules
  - State changes of the DAG invoke plugins on the vertices
  - Plugins monitor runtime information and provide hints to TEZ

Example: Adaption of the number of reducers

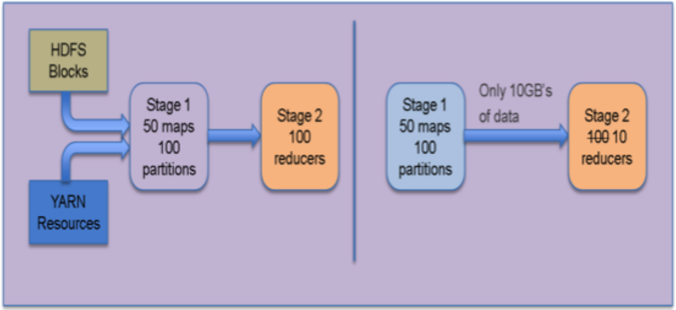


Figure: Source: Introducing... Tez. Hortonworks [19]

TEZ Resource Management

- Task and resource aware scheduling
- Pre-launch and re-use containers and intermediate results (caching)

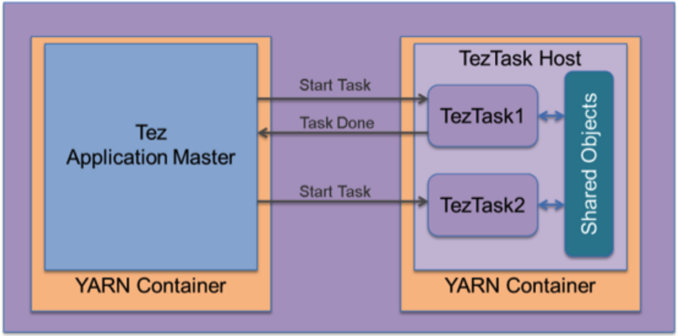


Figure: Source: Introducing... Tez. Hortonworks [19]

Outline

1. Hadoop
2. Map Reduce
3. Hadoop 2
4. TEZ Execution Engine
5. Development
   - Coding
   - Compilation
   - Execution
   - Debugging
   - Job Information via Web Interface
6. Summary

Coding

- Programming Map-Reduce can be done in various languages
  - Java (beware, it's very low-level)
  - Python
  - ... basically any language nowadays!
- Process:
  - Implement map/reduce functions
  - Main method controls process:
    - Define mapper/reducer/combiner
    - Define input/output formats and files
    - Run the job
- Programming of TEZ in Java
- Command line tools to run the "application"
- There are some tools for debugging / performance analysis

Coding: Wordcount, Mapper & Reducer

Goal: Count the frequency of each word in a text

```java
package org.myorg;
import java.io.IOException; import java.util.*; import org.apache.hadoop.fs.Path; import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*; import org.apache.hadoop.mapred.*; import org.apache.hadoop.util.*;

public class WordCount {
  public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1); // for small optimization of object cleaning, reuse object

    // Mapper splits sentence and creates the tuple (word, 1) for each word
    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      String line = value.toString();
      Text word = new Text();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
      }
    }}

  // Reducer accumulates tuples with the same key by summing their frequency
  public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public 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));
    }
  } // Continued => see the next slide
```

Coding: Wordcount, Main Method

The main method configures the Hadoop Job[4]

```
  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    // Set data types of output
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    // Set classes for map, reduce and combiner
    conf.setMapperClass(Map.class);
    conf.setReducerClass(Reduce.class);
    conf.setCombinerClass(Reduce.class);

    // Set file input and output format
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    // Configure input and output paths
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }
}
```

See `https://github.com/apache/tez/tree/master/tez-examples/src/main/java/org/apache/tez/examples` for examples with TEZ

4 There are more modern interfaces available, you'll see in the excercise.

Compilation

Here we compile manually and are not using ant or maven:

1. Prepare the class path for dependencies (may be complex)
2. Compile each Java file
3. Create a JAR package

Example

```
# Java classpath with all required JAR files
CP=/usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core.jar:
    ↪ /usr/hdp/current/hadoop-hdfs-client/hadoop-hdfs.jar
    ↪ :/usr/hdp/current/hadoop/hadoop-common.jar

# Compile a Java file and output all artifacts to the classes directory
# Repeat this step until all required sources are compiled to byte code
javac -classpath $CP -d classes AveragePerformance.java

# Create a JAR package from the classes directory
jar -cvf averagePerformance.jar -C classes .

# Now we are ready to submit the job to HADOOP
```

Execution

Syntax: `[hadoop|yarn] jar FILE.jar ClassWithMain Arguments`

Example

```
> hadoop jar averagePerformance.jar de.wr.AveragePerformance data-energy-efficiency.csv summary
STARTUP: Computing average  ## NOTE: This is output of the main() method
15/10/15 13:49:24 INFO impl.TimelineClientImpl: Timeline service address: http://abu3.cluster:8188/ws/v1/timeline/
15/10/15 13:49:25 INFO client.RMProxy: Connecting to ResourceManager at abu3.cluster/10.0.0.65:8050
15/10/15 13:49:25 INFO impl.TimelineClientImpl: Timeline service address: http://abu3.cluster:8188/ws/v1/timeline/
15/10/15 13:49:25 INFO client.RMProxy: Connecting to ResourceManager at abu3.cluster/10.0.0.65:8050
15/10/15 13:49:26 INFO mapred.FileInputFormat: Total input paths to process : 1
15/10/15 13:49:26 INFO mapreduce.JobSubmitter: number of splits:8
15/10/15 13:49:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1444759114226_0016
15/10/15 13:49:27 INFO impl.YarnClientImpl: Submitted application application_1444759114226_0016
15/10/15 13:49:27 INFO mapreduce.Job: The url to track the job: http://abu3.cluster:8088/proxy/application_1444759114226_0016/
15/10/15 13:49:27 INFO mapreduce.Job: Running job: job_1444759114226_0016
15/10/15 13:49:37 INFO mapreduce.Job: Job job_1444759114226_0016 running in uber mode : false
15/10/15 13:49:37 INFO mapreduce.Job:  map 0% reduce 0%
15/10/15 13:49:54 INFO mapreduce.Job:  map 11% reduce 0%
15/10/15 13:50:02 INFO mapreduce.Job:  map 100% reduce 100%
15/10/15 13:50:02 INFO mapreduce.Job: Job job_1444759114226_0016 completed successfully
15/10/15 13:50:02 INFO mapreduce.Job: Counters: 50
  File System Counters
    FILE: Number of bytes read=768338
    FILE: Number of bytes written=2679321
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=1007776309
    HDFS: Number of bytes written=1483856
    HDFS: Number of read operations=27
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=8
```

Retrieving Output Data

The output is a directory containing one file per reducer

```
# Retrieve the summary directory
$ hadoop fs -get summary
$ ls -lah summary/
-rw-r--r-- 1 kunkel wr 1,5M Okt 15 14:45 part-00000
-rw-r--r-- 1 kunkel wr    0 Okt 15 14:45 _SUCCESS
$ head summary/part-00000
ESM_example_ESM_example_ESM_example_ESM_example 4397 112.69512266727315
     ↪ 186388.93997432772 ...
EXX_example_EXX_example_EXX_example_EXX_example 4511 118.44219725094219
     ↪ 251865.2199417397 ...
...

# A merged file can be retrieved via getmerge
hadoop fs -getmerge summary summary.csv
```

Using Arbitrary Tools/Languages via Streaming

- Hadoop Streaming [22] allows to pipe data through arbitrary tools
- This allows easy integration of Python code, e.g.,

```
yarn jar /usr/hdp/current/hadoop-mapreduce/hadoop-streaming.jar \
  -Dmapred.map.tasks=11 -mapper $PWD/mein-map.py \
  -Dmapred.reduce.tasks=1 -reducer $PWD/mein-reduce.py \
  -input <input> -output <output-directory>
```

- Map/reduce apps receive lines with key value pairs and emit them
  - ANY other (disturbing) output must be avoided to avoid errors
- Trivial mapper:

```
#!/usr/bin/python3
import sys

for line in sys.stdin:
   print("\t".join(line.split(","))) # Split CSV into key (first word) and values
```

- Easy testing on the shell:

```
cat Input.csv | ./mein-map.py | sort | ./mein-reduce.py
```

Using Arbitrary Tools/Languages via Streaming

- We can use the streaming also to integrate Rscripts

```
#!/usr/bin/env Rscript

# WordCount Example
# Discard error messages for loading libraries (if needed) as this would be seen as a "tuple"
sink(file=NULL, type="message")
library('stringi')
# Remove redirection
sink(type="message")

stdin=file('stdin', open='r')

# Batch processing of multiple lines, here 100 elements
while(length( lines=readLines(con=stdin, n=100L) ) > 0){
  # paste concatenates all lines (the array) together
  # stri_extract_all_words() returns an 2D array of lines with words
  # Instead of paste, we could use unlist() to take care of multiple lines and returns a single array
  # table() counts number of occurences of factor levels (that are strings)
  tblWithCounts = table(stri_extract_all_words(paste(lines, collapse=" ")))
  words = names(tblWithCounts)
  counts = as.vector(tblWithCounts)
  cat(stri_paste(words, counts, sep="\t"), sep="\n")
}
```

- Still: easy testing on the shell, similar execution with streaming

```
cat Input.csv | ./mein-map.R | sort | ./mein-reduce.py
```

Debugging of MapReduce and YARN Applications

Runtime information

- Call: `yarn logs -applicationId` $< ID >$
  - The ID is provided upon startup of the job
- Provides for each phase of the execution
  - Log4j output
  - Node information (logfiles)
  - Container information
  - Stdout, stderr of your application
- Increase log verbosity

```
export YARN_ROOT_LOGGER=DEBUG,console
or
run yarn --loglevel DEBUG ...
```

  - Properties: mapreduce.map.log.level, mapreduce.reduce.log.level
- Dump the current configuration of (X) by adding the argument:
  - Parent class: `hadoop org.apache.hadoop.conf.Configuration`
  - Yarn: `hadoop org.apache.hadoop.yarn.conf.YarnConfiguration`
  - MapReduce: `hadoop org.apache.hadoop.mapred.JobConf`

Example Logfile Output

```
> yarn logs -applicationId application_1444759114226_0016

Container: container_1444759114226_0016_01_000005 on abu3.cluster_45454
========================================================================
LogType:stderr
Log Upload Time:Thu Oct 15 13:50:09 +0200 2015
LogLength:243
Log Contents:
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.impl.MetricsSystemImpl).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
End of LogType:stderr

LogType:stdout
Log Upload Time:Thu Oct 15 13:50:09 +0200 2015
LogLength:751944
Log Contents:
...
KEY: 134195662 word cpu_idl_idl_idl
ACCEPTING LINE
KEY: 134204510 word cpu_idl_idl_idl
ACCEPTING LINE
KEY: 134213460 word cpu_idl_idl_idl
ACCEPTING LINE
End of LogType:stdout
...
```

Job Information via Web Interface

- The task tracker keeps detailed information about job execution
- Access via an internal web-server on Port 8088 and 19888
- An internal web-server on each node provides node information
- On a firewalled cluster, SSH forwards are required
  - `ssh -L 8080:NODE:8088 -L 19888:NODE:19888 USERNAME@HOST`

Example

```
# Output when submitting the job:
16/10/21 12:50:27 INFO mapreduce.Job: The url to track the job: http://gwu101:8088/proxy/application_1444759114226_0016/

# After SSH forward visit localhost:8088, you may need to change the hostname from abu3.cluster to localhost again
```

Job Status

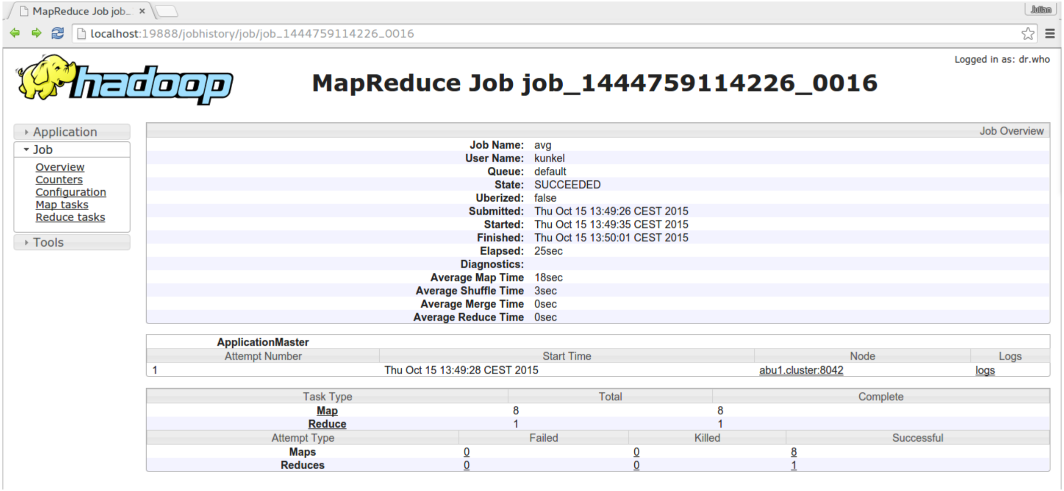
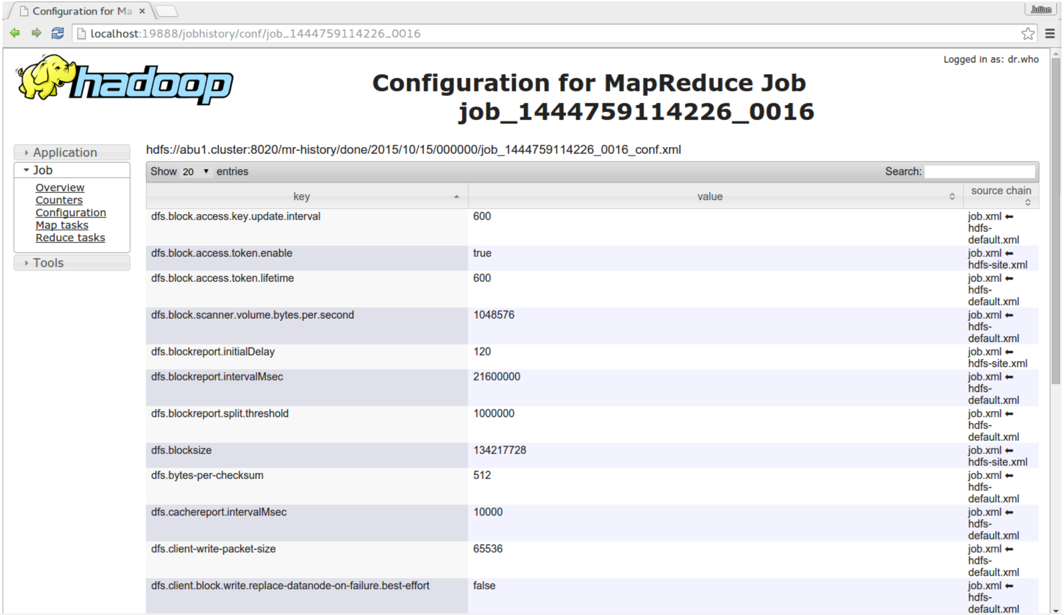


Figure: Overview, when using the tracking url

Job Configuration



Configuration for Ma ×

localhost:19888/jobhistory/conf/job_1444759114226_0016

Logged in as: dr.who

Configuration for MapReduce Job job_1444759114226_0016

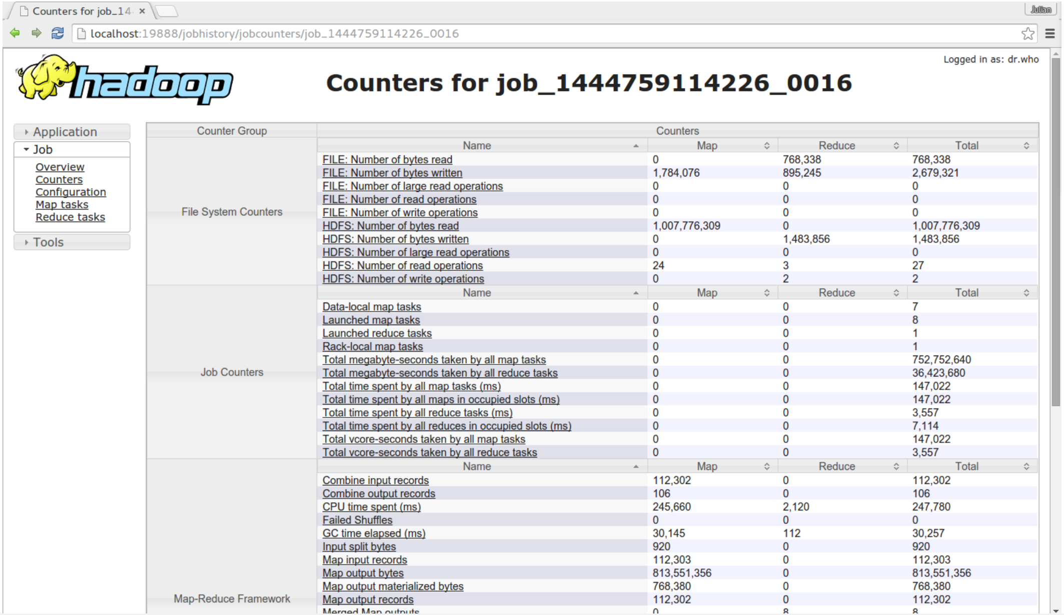
- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

hdfs://abu1.cluster:8020/mr-history/done/2015/10/15/000000/job_1444759114226_0016_conf.xml

Show 20 entries Search:

key	value	source chain
dfs.block.access.key.update.interval	600	job.xml ⟵ hdfs-default.xml
dfs.block.access.token.enable	true	job.xml ⟵ hdfs-site.xml
dfs.block.access.token.lifetime	600	job.xml ⟵ hdfs-default.xml
dfs.block.scanner.volume.bytes.per.second	1048576	job.xml ⟵ hdfs-default.xml
dfs.blockreport.initialDelay	120	job.xml ⟵ hdfs-site.xml
dfs.blockreport.intervalMsec	21600000	job.xml ⟵ hdfs-default.xml
dfs.blockreport.split.threshold	1000000	job.xml ⟵ hdfs-default.xml
dfs.blocksize	134217728	job.xml ⟵ hdfs-site.xml
dfs.bytes-per-checksum	512	job.xml ⟵ hdfs-default.xml
dfs.cachereport.intervalMsec	10000	job.xml ⟵ hdfs-default.xml
dfs.client-write-packet-size	65536	job.xml ⟵ hdfs-default.xml
dfs.client.block.write.replace-datanode-on-failure.best-effort	false	job.xml ⟵ hdfs-default.xml

Performance Counters


Counters for job_14 ×

localhost:19888/jobhistory/jobcounters/job_1444759114226_0016

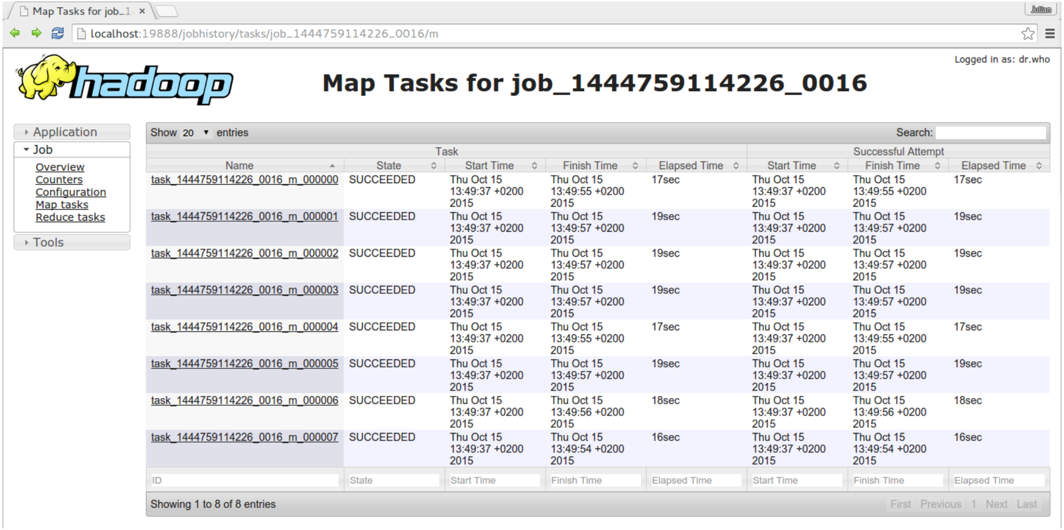
Logged in as: dr.who

Counters for job_1444759114226_0016

- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	768,338	768,338
	FILE: Number of bytes written	1,784,076	895,245	2,679,321
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1,007,776,309	0	1,007,776,309
	HDFS: Number of bytes written	0	1,483,856	1,483,856
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	24	3	27
	HDFS: Number of write operations	0	2	2
Job Counters	Data-local map tasks	0	0	7
	Launched map tasks	0	0	8
	Launched reduce tasks	0	0	1
	Rack-local map tasks	0	0	1
	Total megabyte-seconds taken by all map tasks	0	0	752,752,640
	Total megabyte-seconds taken by all reduce tasks	0	0	36,423,680
	Total time spent by all map tasks (ms)	0	0	147,022
	Total time spent by all maps in occupied slots (ms)	0	0	147,022
	Total time spent by all reduce tasks (ms)	0	0	3,557
	Total time spent by all reduces in occupied slots (ms)	0	0	7,114
	Total vcore-seconds taken by all map tasks	0	0	147,022
	Total vcore-seconds taken by all reduce tasks	0	0	3,557
Map-Reduce Framework	Combine input records	112,302	0	112,302
	Combine output records	106	0	106
	CPU time spent (ms)	245,660	2,120	247,780
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	30,145	112	30,257
	Input split bytes	920	0	920
	Map input records	112,303	0	112,303
	Map output bytes	813,551,356	0	813,551,356
	Map output materialized bytes	768,380	0	768,380
	Map output records	112,302	0	112,302
	Merged Map outputs	0	8	8

Information About Map Tasks



Map Tasks for job_1... ×

localhost:19888/jobhistory/tasks/job_1444759114226_0016/m

Logged in as: dr.who

Map Tasks for job_1444759114226_0016

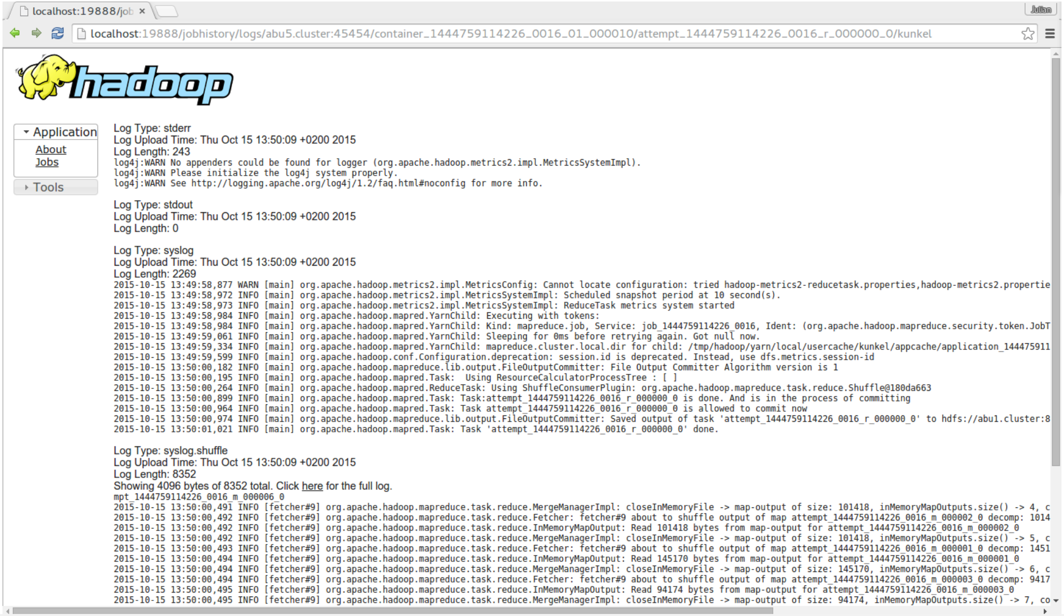
- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

Show 20 entries — Search:

Task					Successful Attempt		
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Finish Time	Elapsed Time
task_1444759114226_0016_m_000000	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:55 +0200 2015	17sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:55 +0200 2015	17sec
task_1444759114226_0016_m_000001	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec
task_1444759114226_0016_m_000002	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec
task_1444759114226_0016_m_000003	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec
task_1444759114226_0016_m_000004	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:55 +0200 2015	17sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:55 +0200 2015	17sec
task_1444759114226_0016_m_000005	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:57 +0200 2015	19sec
task_1444759114226_0016_m_000006	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:56 +0200 2015	18sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:56 +0200 2015	18sec
task_1444759114226_0016_m_000007	SUCCEEDED	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:54 +0200 2015	16sec	Thu Oct 15 13:49:37 +0200 2015	Thu Oct 15 13:49:54 +0200 2015	16sec
ID	State	Start Time	Finish Time	Elapsed Time	Start Time	Finish Time	Elapsed Time

Showing 1 to 8 of 8 entries — First Previous 1 Next Last

Logfile


localhost:19888/jobhistory/logs/abu5.cluster:45454/container_1444759114226_0016_01_000010/attempt_1444759114226_0016_r_000000_0/kunkel

Application
About
Jobs
Tools

Log Type: stderr
Log Upload Time: Thu Oct 15 13:50:09 +0200 2015
Log Length: 243

```
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.impl.MetricsSystemImpl).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Log Type: stdout
Log Upload Time: Thu Oct 15 13:50:09 +0200 2015
Log Length: 0

Log Type: syslog
Log Upload Time: Thu Oct 15 13:50:09 +0200 2015
Log Length: 2269

```
2015-10-15 13:49:58,877 WARN [main] org.apache.hadoop.metrics2.impl.MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-reducetask.properties,hadoop-metrics2.propertie
2015-10-15 13:49:58,972 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: Scheduled snapshot period at 10 second(s).
2015-10-15 13:49:58,973 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: ReduceTask metrics system started
2015-10-15 13:49:58,984 INFO [main] org.apache.hadoop.mapred.YarnChild: Executing with tokens:
2015-10-15 13:49:58,984 INFO [main] org.apache.hadoop.mapred.YarnChild: Kind: mapreduce.job, Service: job_1444759114226_0016, Ident: (org.apache.hadoop.mapreduce.security.token.JobT
2015-10-15 13:49:59,061 INFO [main] org.apache.hadoop.mapred.YarnChild: Sleeping for 0ms before retrying again. Got null now.
2015-10-15 13:49:59,334 INFO [main] org.apache.hadoop.mapred.YarnChild: mapreduce.cluster.local.dir for child: /tmp/hadoop/yarn/local/usercache/kunkel/appcache/application_144475911
2015-10-15 13:49:59,599 INFO [main] org.apache.hadoop.conf.Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
2015-10-15 13:50:00,182 INFO [main] org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter: File Output Committer Algorithm version is 1
2015-10-15 13:50:00,195 INFO [main] org.apache.hadoop.mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
2015-10-15 13:50:00,264 INFO [main] org.apache.hadoop.mapred.ReduceTask: Using ShuffleConsumerPlugin: org.apache.hadoop.mapreduce.task.reduce.Shuffle@180da663
2015-10-15 13:50:00,899 INFO [main] org.apache.hadoop.mapred.Task: Task:attempt_1444759114226_0016_r_000000_0 is done. And is in the process of committing
2015-10-15 13:50:00,964 INFO [main] org.apache.hadoop.mapred.Task: Task attempt_1444759114226_0016_r_000000_0 is allowed to commit now
2015-10-15 13:50:00,974 INFO [main] org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter: Saved output of task 'attempt_1444759114226_0016_r_000000_0' to hdfs://abu1.cluster:8
2015-10-15 13:50:01,021 INFO [main] org.apache.hadoop.mapred.Task: Task 'attempt_1444759114226_0016_r_000000_0' done.
```

Log Type: syslog.shuffle
Log Upload Time: Thu Oct 15 13:50:09 +0200 2015
Log Length: 8352
Showing 4096 bytes of 8352 total. Click here for the full log.

```
mpt_1444759114226_0016_m_000006_0
2015-10-15 13:50:00,491 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 101418, inMemoryMapOutputs.size() -> 4, c
2015-10-15 13:50:00,492 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.Fetcher: fetcher#9 about to shuffle output of map attempt_1444759114226_0016_m_000002_0 decomp: 1014
2015-10-15 13:50:00,492 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.InMemoryMapOutput: Read 101418 bytes from map-output for attempt_1444759114226_0016_m_000002_0
2015-10-15 13:50:00,492 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 101418, inMemoryMapOutputs.size() -> 5, c
2015-10-15 13:50:00,493 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.Fetcher: fetcher#9 about to shuffle output of map attempt_1444759114226_0016_m_000001_0 decomp: 1451
2015-10-15 13:50:00,494 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.InMemoryMapOutput: Read 145170 bytes from map-output for attempt_1444759114226_0016_m_000001_0
2015-10-15 13:50:00,494 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 145170, inMemoryMapOutputs.size() -> 6, c
2015-10-15 13:50:00,494 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.Fetcher: fetcher#9 about to shuffle output of map attempt_1444759114226_0016_m_000003_0 decomp: 9417
2015-10-15 13:50:00,495 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.InMemoryMapOutput: Read 94174 bytes from map-output for attempt_1444759114226_0016_m_000003_0
2015-10-15 13:50:00,495 INFO [fetcher#9] org.apache.hadoop.mapreduce.task.reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 94174, inMemoryMapOutputs.size() -> 7, co
```

Node Manager

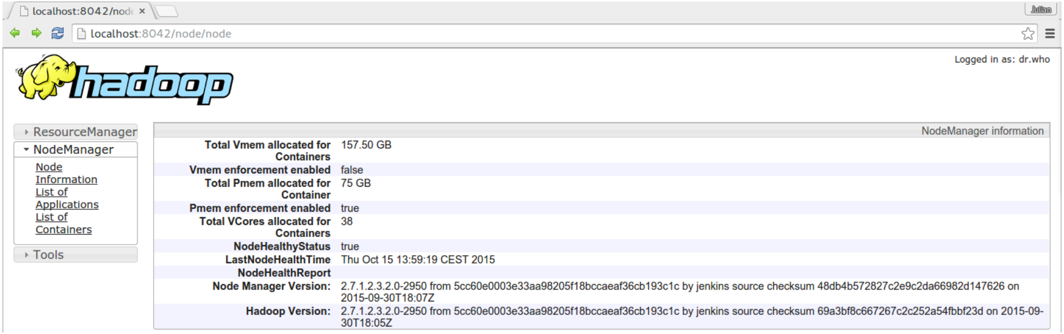


Figure: The Node Manager provides information about a particular node

Summary

- Hadoop provides the file system HDFS and concepts for processing
- HDFS
  - Single writer, multiple reader concurrency
  - Robust and high availability
- MapReduce: fixed function pipeline, reliable execution
- Hadoop2 with YARN: refined architecture for resource management
- TEZ: Execution of DAGs with various configurations

Bibliography

1 Book: Lillian Pierson. **Data Science for Dummies**. John Wiley & Sons

10 Wikipedia

12 Hortonworks http://hortonworks.com/

13 B. Ramamurthy. Hadoop File System. http://www.cse.buffalo.edu/faculty/bina/MapReduce/HDFS.ppt

14 Hadoop Wiki. https://wiki.apache.org/hadoop/

15 B. Hedlund. Understanding Hadoop Clusters and the Network. http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

16 jcdenton. H-stack Introduction Guide. https://github.com/jcdenton/hadoop-guide/blob/master/hadoop.md

17 http://tutorials.techmytalk.com/2014/11/14/mapreduce-composite-key-operation-part2/

18 http://hadoop.apache.org/docs/

19 http://hortonworks.com/blog/introducing-tez-faster-hadoop-processing/

20 Presentation: H. Shah. Apache Tez. Hortonworks.

21 Hadoop I/O http://blog.cloudera.com/blog/2011/01/hadoop-io-sequence-map-set-array-bloommap-files/

22 http://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html

23 http://www.dummies.com/how-to/content/input-splits-in-hadoops-mapreduce.html