

Hadoop

Lecture BigData Analytics

Julian M. Kunkel

julian.kunkel@googlemail.com

University of Hamburg / German Climate Computing Center (DKRZ)

16-10-2015



Outline

- 1 Hadoop
- 2 Map Reduce
- 3 Hadoop 2
- 4 TEZ Execution Engine
- 5 Development
- 6 Summary

1 Hadoop

- Version 1
- Architecture
- File Formats
- I/O Path

2 Map Reduce

3 Hadoop 2

4 TEZ Execution Engine

5 Development

6 Summary

Hadoop File System Shell

Overview

- Invoke via: `hadoop fs <command> <args>`
 - Example: `hadoop fs -ls hdfs://abu1.cluster/`

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

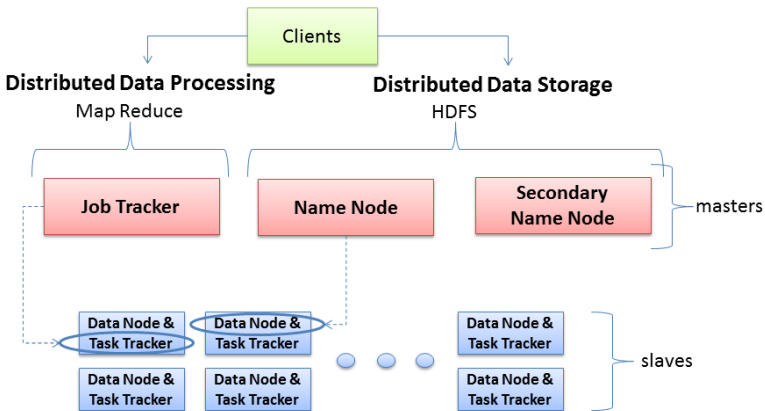
- `distcp`: map reduce parallelized copy between clusters
- `checksum`
- `expunge` (clear trash)
- `setrep` (change replication factor)

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
 - Direct communication to 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

Hadoop Server Roles

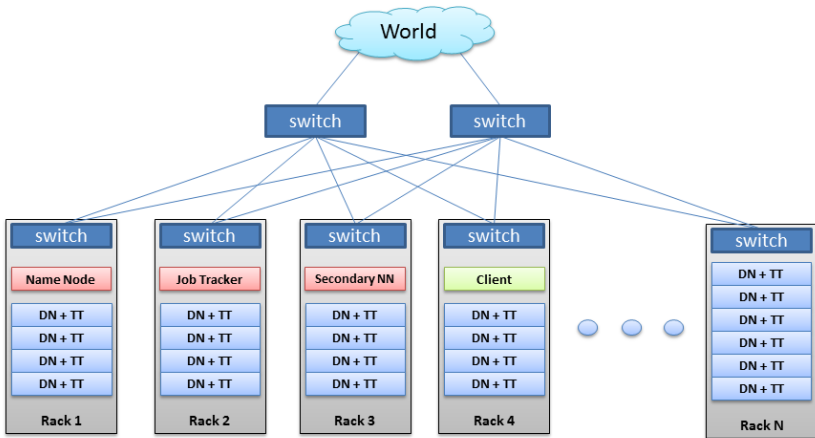


BRAD HEDLUND .com

Source: B. Hedlund. [15]

System-Level Perspective of Hadoop Clusters

Hadoop Cluster



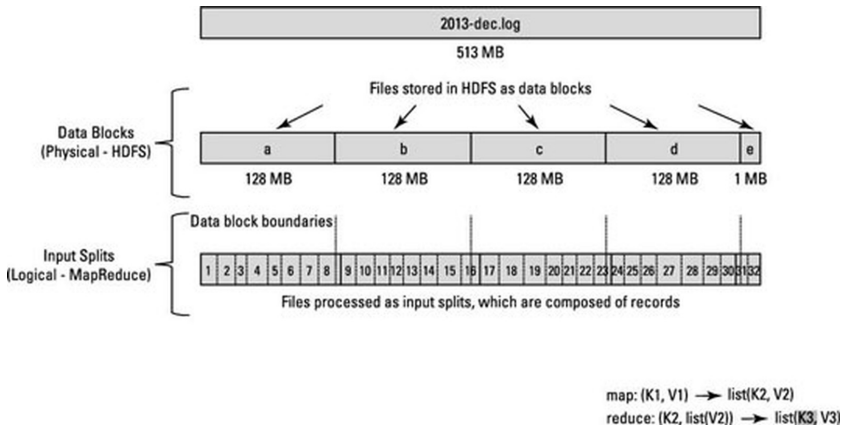
BRAD HEDLUND .com

Source: B. Hedlund. [15]

Parallel Access to Files

- Files are split into blocks
 - Blocks are distributed across nodes
 - A typical block size is 64 MiB
 - Blocks may be compressed individually
- MapReduce jobs process input splits
 - Input splits == logical organization of blocks
 - Each input split is processed by one mapper (local processing preferred)
 - Input splitting depends on the file format
 - Processing for records spanning blocks
 - Skip partial records at the beginning of a split
 - Read data from a remote for truncated records
- File formats not splittable must be avoided
 - 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]



Source: [23]

File Formats

Text files

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

```
This is a simple file.\nWith three lines - \nthis is the end.
```

Comma-separated values (CSV)

- Without header; JSON records are supported
- Does not support block compression

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

Sequence files

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

File Formats (2)

MapFile [21]

- Extends the sequence file
- Provides an index for keys

Avro

- Apache Avro's serialization system format
 - Self-describing data format, allows inference of schema but can be changed upon read
 - Enables exchange of data types between tools
- ⇒ Popular file format for Hadoop ecosystem

File Formats (3)

Record columnar file (RCFile) [10]

- Partition table into row groups
- For each group serialize by column-major
- More efficient to access individual columns

Mapping of data structures into files is done by serialization systems

(De)Serialization

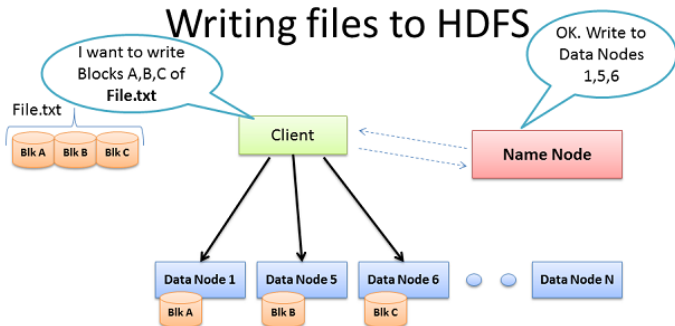
- Serialization is the process creating a byte stream from a data structure
- De-serialization creates a data structure in memory from the byte stream
- Structure \Rightarrow byte stream \Rightarrow structure
- Byte streams can be transferred via network or stored on block storage

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/>

The HDFS I/O Path



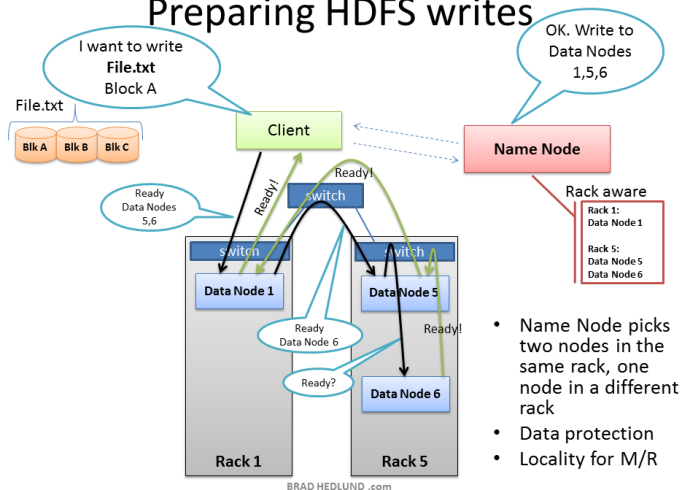
- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

Source: B. Hedlund. [15]

The HDFS Write Path

Preparing HDFS writes

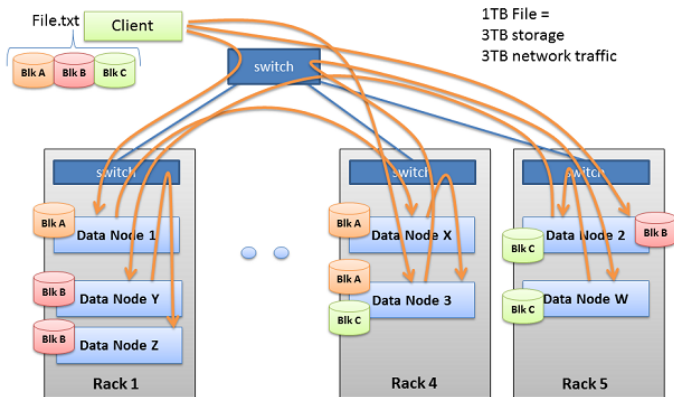


- Name Node picks two nodes in the same rack, one node in a different rack
- Data protection
- Locality for M/R

Source: B. Hedlund [15]

The HDFS Write Path

Multi-block Replication Pipeline

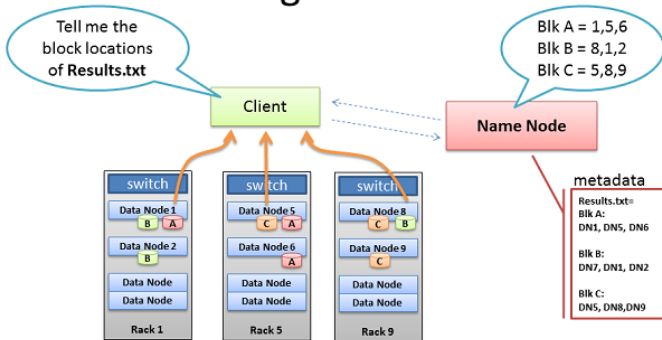


BRAD HEDLUND .com

Source: B. Hedlund [15]

The HDFS Read Path

Client reading files from HDFS



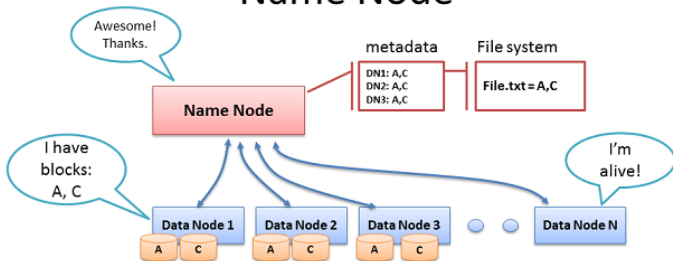
- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

BRAD HEDLUND .com

Source: B. Hedlund [15]

Name Node and High Availability

Name Node



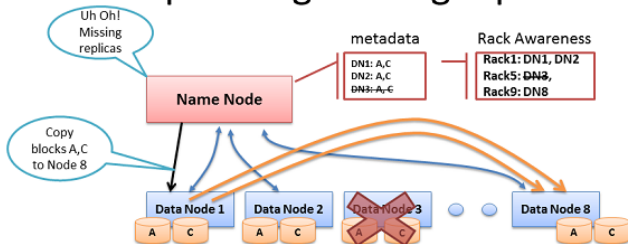
- Data Node sends Heartbeats
- Every 10th heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds
- If Name Node is down, HDFS is down

BRAD HEDLUND .com

Source: B. Hedlund. [15]

Name Node and High Availability

Re-replicating missing replicas



- Missing Heartbeats signify lost Nodes
- Name Node consults metadata, finds affected data
- Name Node consults Rack Awareness script
- Name Node tells a Data Node to re-replicate

BRAD HEDLUND .com

Source: B. Hedlund. [15]

1 Hadoop

2 Map Reduce

- Overview
- Execution on Hadoop

3 Hadoop 2

4 TEZ Execution Engine

5 Development

6 Summary

Map Reduce Execution Paradigm

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

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 depends on the configuration

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

```
1 map(key, text): # input: key=position, text=line
2   for each word in text:
3     Emit(word,1) # outputs: key/value
4
5 reduce(key, list of values): # input: key == word, our mapper output
6   count = 0
7   for each v in values:
8     count += v
9   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		Country	State	Population
USA,	CA,	Su,	12	⇒	USA	CA	54
USA,	CA,	SA,	42		USA	MO	33
USA,	MO,	XY,	23				
USA,	MO,	AB,	10				

Algorithm

```

1 map(key, line):
2   (county, state, city, population) = line.split(',')
3   EmitIntermediate( (country, state), population )
4
5 reduce(key, values): # key=(country,state) values=list of populations
6   count = 0
7   for each v in values:
8     count += v
9   Emit(key, count)

```

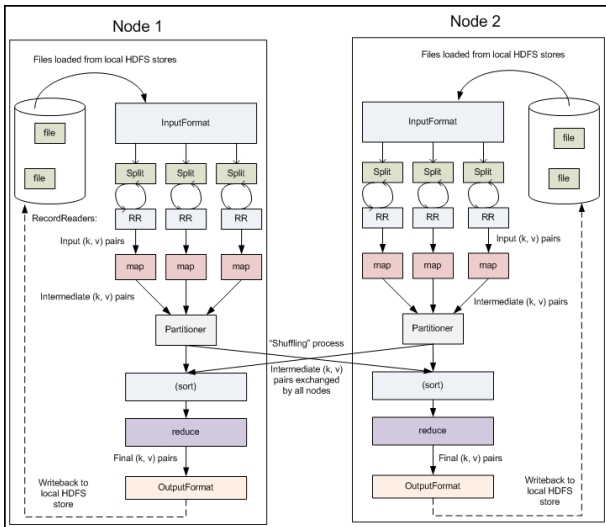

Phases of MapReduce Execution

Phases of MapReduce Execution

- 0 Distribute code (JAR files)
- 1 Determine blocks and file splits, assign mappers to splits and slots
- 2 Map: Invoke (local) map functions
- 3 Shuffle: Sort by the key, exchange data
- 4 Combine: Perform a local reduce by the key
- 5 Partition: Partition key space among reducers (typically via hashing)
- 6 Reduce: Invoke reducers
- 7 Write output, each reducer writes to its own file²

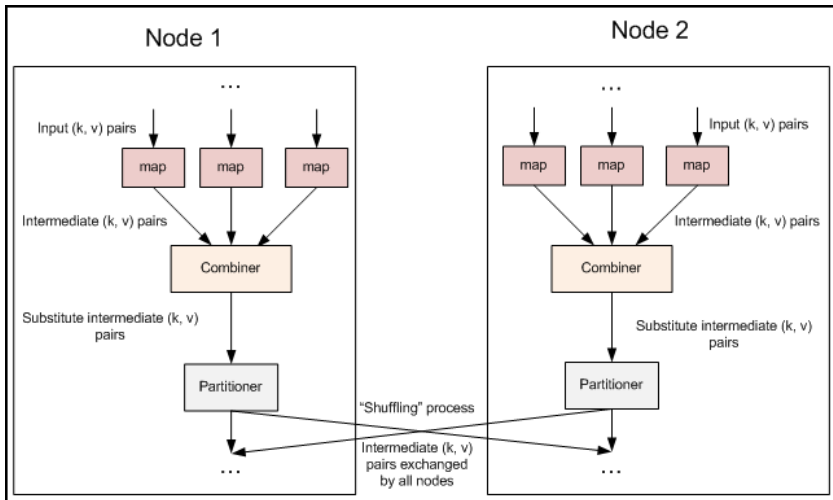
²Use `hadoop fs -getmerge <HDFSDIR> file.txt` to retrieve merged output

Execution of MapReduce – the Big Picture



Source: jcdenton. [16]

Execution of MapReduce on HDFS – the Combiner



Source: jcdenton. [16]

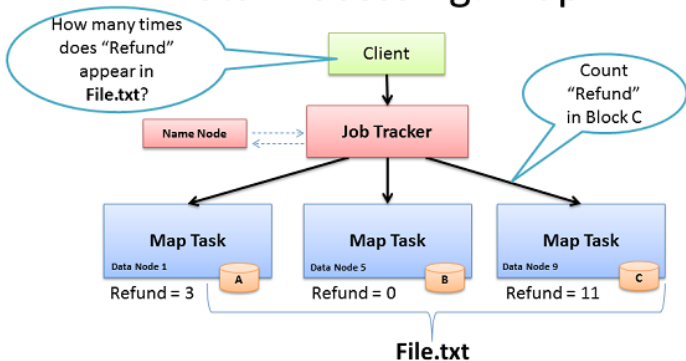
Execution of MapReduce Tasks on Hadoop [14]

Steps in the execution of tasks

- 1 Client submits jobs 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 task on the TaskTracker nodes
- 5 Monitoring of TaskTrack nodes
 - 1 If heartbeat signals are missed, work is rescheduled on another TaskTracker
 - 2 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

Data Processing: Map



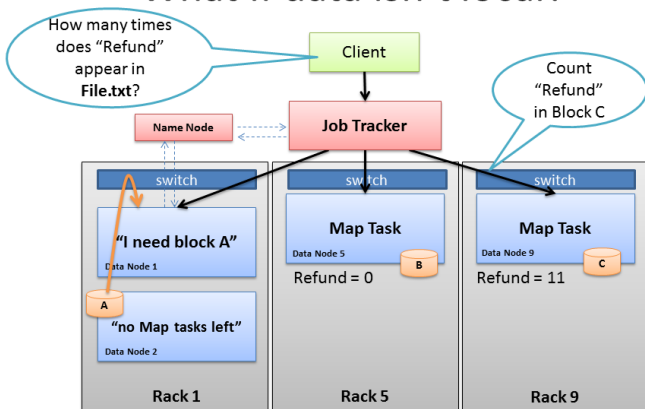
- **Map:** "Run this computation on your local data"
- Job Tracker delivers Java code to Nodes with local data

BRAD HEDLUND .com

Source: B. Hedlund. [15]

Execution of MapReduce

What if data isn't local?



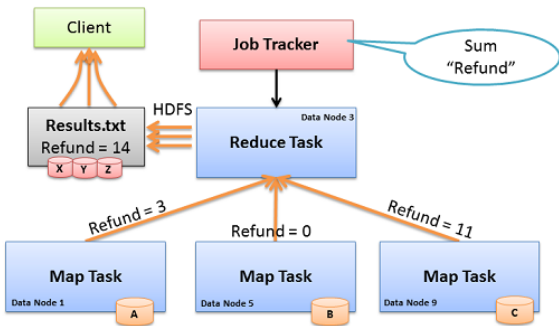
- Job Tracker tries to select Node in same rack as data
- Name Node rack awareness

BRAD HEDLUND .com

Source: B. Hedlund. [15]

Execution of MapReduce

Data Processing: Reduce



- **Reduce:** “Run this computation across Map results”
- Map Tasks send output data to Reducer over the network
- Reduce Task data output written to and read from HDFS

BRAD HEDLUND .com

Source: B. Hedlund. [15]

1 Hadoop

2 Map Reduce

3 Hadoop 2

■ Overview

■ System Architecture

4 TEZ Execution Engine

5 Development

6 Summary

System Architecture

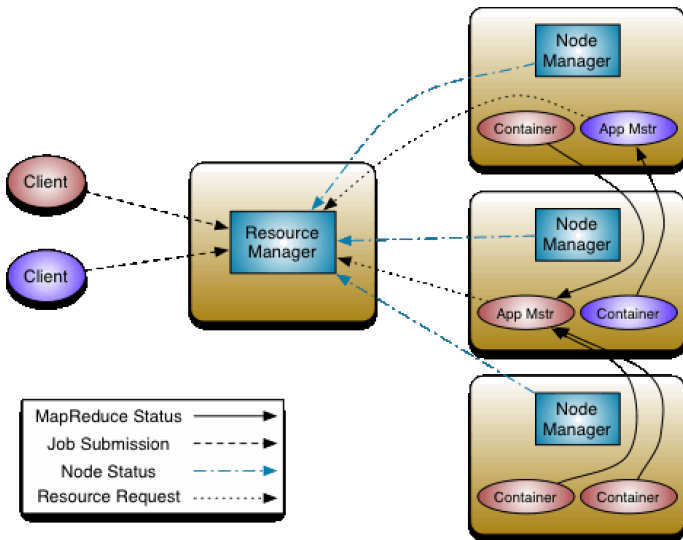
Yarn modularizes JobTracker functionality

- 1 Resource management
- 2 Job scheduling/execution incl. monitoring

Data computation framework

- Applications are executed in containers
- ResourceManager component (global daemon)
 - Partitions resources and schedules applications
 - Scheduler: distributes resources among applications
 - ApplicationsManager: accepts jobs, negotiates execution of ApplicationMaster
- 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



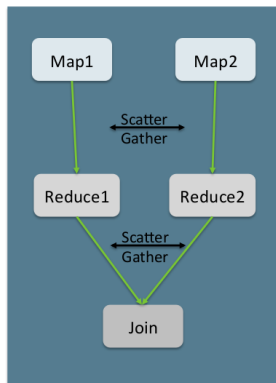
Source: Apache Hadoop NextGen [18]

TEZ Example DAG [20]

```

1 // Define DAG
2 DAG dag = new DAG();
3 // Define Vertex, which class to execute
4 Vertex Map1 = new Vertex(Processor.class);
5 // Define Edge
6 Edge edge = Edge(Map1, Reduce2,
7     SCATTER_GATHER, // Distribution of data
8     ↪ from source to target(s)
9     PERSISTED, // Persistency of data
10    SEQUENTIAL, // Scheduling: either
11    ↪ concurrent or sequential
12    ↪ execution
13    Output.class, Input.class);
14 // Connect edges with vertex
15 dag.addVertex(Map1).addEdge(edge)...

```

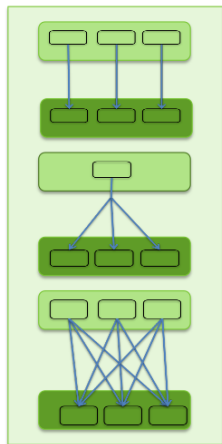


Source: Apache Tez. H. Shah [20]

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

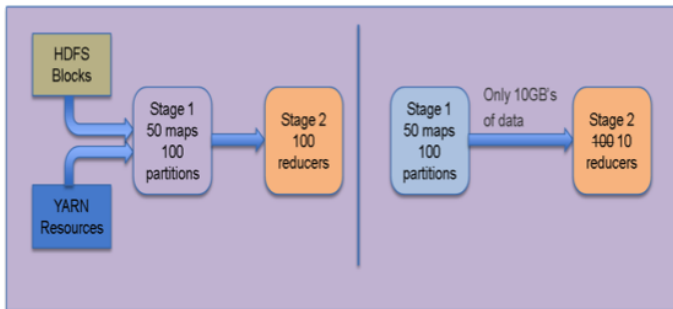


Source: Apache Tez. H. Shah [20]

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

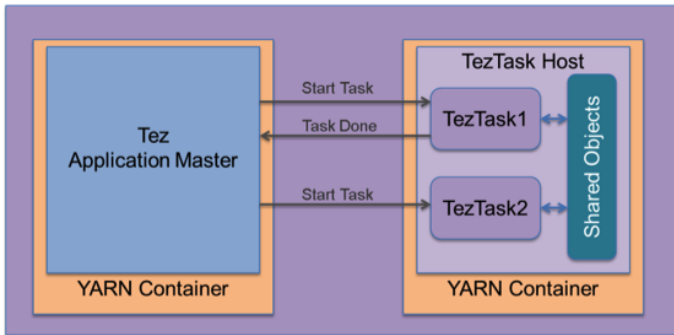
Example: Adaption of the number of reducers



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

TEZ Resource Management

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



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

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: Wordcount, Mapper & Reducer

Goal: Count the frequency of each word in a text

```

1 package org.myorg;
2 import java.io.IOException; import java.util.*; import org.apache.hadoop.fs.Path; import org.apache.hadoop.conf.*;
3 import org.apache.hadoop.io.*; import org.apache.hadoop.mapred.*; import org.apache.hadoop.util.*;
4
5 public class WordCount {
6     public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
7         private final static IntWritable one = new IntWritable(1); // for small optimization of object cleaning, reuse object
8
9         // Mapper splits sentence and creates the tuple (word, 1) for each word
10        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws
11            ↳ IOException {
12            String line = value.toString();
13            Text word = new Text();
14            StringTokenizer tokenizer = new StringTokenizer(line);
15            while (tokenizer.hasMoreTokens()) {
16                word.set(tokenizer.nextToken());
17                output.collect(word, one);
18            }
19        }
20
21        // Reducer accumulates tuples with the same key by summing their frequency
22        public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
23            public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter)
24                ↳ throws IOException {
25                int sum = 0;
26                while (values.hasNext()) {
27                    sum += values.next().get();
28                }
29                output.collect(key, new IntWritable(sum));
30            }
31        }
32    } // Continued => see the next slide

```



Coding: Wordcount, Main Method

The main method configures the Hadoop Job³

```
1 public static void main(String[] args) throws Exception {
2     JobConf conf = new JobConf(WordCount.class);
3     conf.setJobName("wordcount");
4
5     // Set data types of output
6     conf.setOutputKeyClass(Text.class);
7     conf.setOutputValueClass(IntWritable.class);
8
9     // Set classes for map, reduce and combiner
10    conf.setMapperClass(Map.class);
11    conf.setReducerClass(Reduce.class);
12    conf.setCombinerClass(Reduce.class);
13
14    // Set file input and output format
15    conf.setInputFormat(TextInputFormat.class);
16    conf.setOutputFormat(TextOutputFormat.class);
17
18    // Configure input and output paths
19    FileInputFormat.setInputPaths(conf, new Path(args[0]));
20    FileOutputFormat.setOutputPath(conf, new Path(args[1]));
21
22    JobClient.runJob(conf);
23 }
24 }
```

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

³There are more modern interfaces available, you'll see in the exercise.

Compilation

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

```
1 # Java classpath with all required JAR files
2 CP=/usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core.jar:
   ↪ /usr/hdp/current/hadoop-hdfs-client/hadoop-hdfs.jar
   ↪ ./usr/hdp/2.3.2.0-2950/hadoop/hadoop-common.jar
3
4 # Compile a Java file and output all artifacts to the classes directory
5 # Repeat this step until all required sources are compiled to byte code
6 javac -classpath $CP -d classes AveragePerformance.java
7
8 # Create a JAR package from the classes directory
9 jar -cvf averagePerformance.jar -C classes .
10
11 # Now we are ready to submit the job to HADOOP
```

Execution

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

Example

```
1 > hadoop jar averagePerformance.jar de.wr.AveragePerformance data-energy-efficiency.csv summary
2 STARTUP: Computing average ## NOTE: This is output of the main() method
3 15/10/15 13:49:24 INFO impl.TimelineClientImpl: Timeline service address: http://abu5.cluster:8188/ws/v1/timeline/
4 15/10/15 13:49:25 INFO client.RMPProxy: Connecting to ResourceManager at abu5.cluster/10.0.0.65:8050
5 15/10/15 13:49:25 INFO impl.TimelineClientImpl: Timeline service address: http://abu5.cluster:8188/ws/v1/timeline/
6 15/10/15 13:49:25 INFO client.RMPProxy: Connecting to ResourceManager at abu5.cluster/10.0.0.65:8050
7 15/10/15 13:49:26 INFO mapred.FileInputFormat: Total input paths to process : 1
8 15/10/15 13:49:26 INFO mapreduce.JobSubmitter: number of splits:8
9 15/10/15 13:49:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1444759114226_0016
10 15/10/15 13:49:27 INFO impl.YarnClientImpl: Submitted application application_1444759114226_0016
11 15/10/15 13:49:27 INFO mapreduce.Job: The url to track the job: http://abu5.cluster:8088/proxy/application_1444759114226_0016/
12 15/10/15 13:49:27 INFO mapreduce.Job: Running job: job_1444759114226_0016
13 15/10/15 13:49:37 INFO mapreduce.Job: Job job_1444759114226_0016 running in uber mode : false
14 15/10/15 13:49:37 INFO mapreduce.Job: map 0% reduce 0%
15 15/10/15 13:49:54 INFO mapreduce.Job: map 11% reduce 0%
16 15/10/15 13:50:02 INFO mapreduce.Job: map 100% reduce 100%
17 15/10/15 13:50:02 INFO mapreduce.Job: Job job_1444759114226_0016 completed successfully
18 15/10/15 13:50:02 INFO mapreduce.Job: Counters: 50
19 File System Counters
20 FILE: Number of bytes read=768338
21 FILE: Number of bytes written=2679321
22 FILE: Number of read operations=0
23 FILE: Number of large read operations=0
24 FILE: Number of write operations=0
25 HDFS: Number of bytes read=1007776309
26 HDFS: Number of bytes written=1483856
27 HDFS: Number of read operations=27
28 HDFS: Number of large read operations=0
29 HDFS: Number of write operations=2
30 Job Counters
```

Retrieving Output Data

The output is a directory containing one file per reducer

```
1 # Retrieve the summary directory
2 $ hadoop fs -get summary
3 $ ls -lah summary/
4 -rw-r--r-- 1 kunkel wr 1,5M 0kt 15 14:45 part-00000
5 -rw-r--r-- 1 kunkel wr    0 0kt 15 14:45 _SUCCESS
6 $ head summary/part-00000
7 ESM_example_ESM_example_ESM_example_ESM_example 4397 112.69512266727315
   ↪ 186388.93997432772 ...
8 EXX_example_EXX_example_EXX_example_EXX_example 4511 118.44219725094219
   ↪ 251865.2199417397 ...
9 ...
10
11 # A merged file can be retrieved via getmerge
12 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.

```

1 yarn jar /usr/hdp/2.3.2.0-2950/hadoop-mapreduce/hadoop-streaming.jar \
2   -Dmapred.map.tasks=11 -mapper $PWD/mein-map.py \
3   -Dmapred.reduce.tasks=1 -reducer $PWD/mein-reduce.py \
4   -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:

```

1 #!/usr/bin/python3
2 import sys
3
4 for line in sys.stdin:
5     print("\t".join(line.split(","))) # Split CSV into key (first word) and values

```

- Easy testing on the shell:

```

1 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

```

1 #!/usr/bin/env Rscript
2
3 # WordCount Example
4 # Discard error messages for loading libraries (if needed)
5 sink(file=NULL, type="message")
6 library('stringi')
7 # Remove redirection
8 sink(type="message")
9
10 stdin=file('stdin', open='r')
11
12 # Batch processing of multiple lines, here 100 elements
13 while(length( lines=readLines(con=stdin, n=100L) ) > 0){
14   # paste concatenates all lines (the array) together
15   # stri_extract_all_words() returns an 2D array of lines with words
16   # Instead of paste, we could use unlist() to take care of multiple lines and returns a single array
17   # table() counts number of occurrences of factor levels (that are strings)
18   tblWithCounts = table(stri_extract_all_words(paste(lines, collapse=" ")))
19   words = names(tblWithCounts)
20   counts = as.vector(tblWithCounts)
21   cat(stri_paste(words, counts, sep="\t"), sep="\n")
22 }

```

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

```

1 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

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

- Properties: `mapreduce.map.log.level`, `mapreduce.reduce.log.level`
- Dump the current configuration by calling
 - 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

```

1 > yarn logs -applicationId application_1444759114226_0016
2
3 Container: container_1444759114226_0016_01_000005 on abu3.cluster_45454
4 =====
5 LogType:stderr
6 Log Upload Time:Thu Oct 15 13:50:09 +0200 2015
7 LogLength:243
8 Log Contents:
9 log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.impl.MetricsSystemImpl).
10 log4j:WARN Please initialize the log4j system properly.
11 log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
12 End of LogType:stderr
13
14 LogType:stdout
15 Log Upload Time:Thu Oct 15 13:50:09 +0200 2015
16 LogLength:751944
17 Log Contents:
18 ...
19 KEY: 134195662 word cpu_idl_idl_idl
20 ACCEPTING LINE
21 KEY: 134204510 word cpu_idl_idl_idl
22 ACCEPTING LINE
23 KEY: 134213460 word cpu_idl_idl_idl
24 ACCEPTING LINE
25 End of LogType:stdout
26 ...

```

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 our firewalled cluster, SSH forwards are required
 - `ssh -L 8080:abu5:8088 -L 19888:abu5:19888 kunkel@cluster.wr.informatik.uni-hamburg.de`

Example

```
1 # Output when submitting the job:
2 15/10/15 13:49:27 INFO mapreduce.Job: The url to track the job: http://abu5.cluster:8088/proxy/application_1444759114226_0016/
3
4 # After SSH forward visit localhost:8088, you may need to change the hostname from abu5.cluster to localhost again
```

Job Status

MapReduce Job job_... x
localhost:19888/jobhistory/job/job_1444759114226_0016

hadoop

MapReduce Job job_1444759114226_0016

Logged in as: dr.who

Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
Tools

Job Overview

Job Name: avg
User Name: kunkel
Queue: default
State: SUCCEEDED
Uberized: false
Submitted: Thu Oct 15 13:49:26 CEST 2015
Started: Thu Oct 15 13:49:35 CEST 2015
Finished: Thu Oct 15 13:50:01 CEST 2015
Elapsed: 25sec
Diagnostics:
Average Map Time: 18sec
Average Shuffle Time: 3sec
Average Merge Time: 0sec
Average Reduce Time: 0sec

ApplicationMaster		Start Time	Node	Logs
1	Attempt Number	Thu Oct 15 13:49:28 CEST 2015	abu1.cluster8042	logs

Task Type	Total	Complete
Map	8	8
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	8
Reduces	0	0	1

Overview, when using the tracking url




Job Configuration

Configuration for Ma... x

localhost:19888/jobhistory/conf/job_1444759114226_0016

Logged in as: dr.who



Configuration for MapReduce Job job_1444759114226_0016

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

Application

Job

- Overview
- Counters
- Configuration
- Map tasks
- Reduce tasks

Tools

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
dfs.block.access.token.lifetime	600	hdfs-site.xml job.xml
dfs.block.scanner.volume.bytes.per.second	1048576	hdfs- default.xml
dfs.blockreport.initialDelay	120	job.xml
dfs.blockreport.intervalMsec	21600000	hdfs-site.xml job.xml
dfs.blockreport.split.threshold	1000000	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	Counter Group	Name	Map	Reduce	Total		
Job	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	Job Counters	Data-local map tasks	0	0	7
				Launched map tasks	0	0	8
				Launched reduce tasks	0	0	1
				Back-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	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 x Admin

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:

Name	State	Task			Successful Attempt		
		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/job: x

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 13:49:59,064 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.Shuffle180da663
 2015-10-15 13:50:00,889 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.
 npt_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 decom: 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 decom: 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 decom: 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, c

Node Manager

localhost:8042/node/node

Logged in as: dr.who

hadoop

ResourceManager

NodeManager

Node Information List of Applications List of Containers

Tools

NodeManager information

Total Vmem allocated for Containers	157.50 GB
Vmem enforcement enabled	false
Total Pmem allocated for Container	75 GB
Pmem enforcement enabled	true
Total VCores allocated for Containers	38
NodeHealthyStatus	true
LastNodeHealthTime	Thu Oct 15 13:59:19 CEST 2015
NodeHealthReport	
Node Manager Version:	2.7.1.2.3.2.0-2950 from 5cc60e0003e33aa98205f18bccaeaf36cb193c1c by jenkins source checksum 48db4b572827c2e9c2da66982d147626 on 2015-09-30T18:07Z
Hadoop Version:	2.7.1.2.3.2.0-2950 from 5cc60e0003e33aa98205f18bccaeaf36cb193c1c by jenkins source checksum 69a3bf8c667267c2c252a54fbf23d on 2015-09-30T18:05Z

The Node Manager provides information about a particular node

Summary

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/>