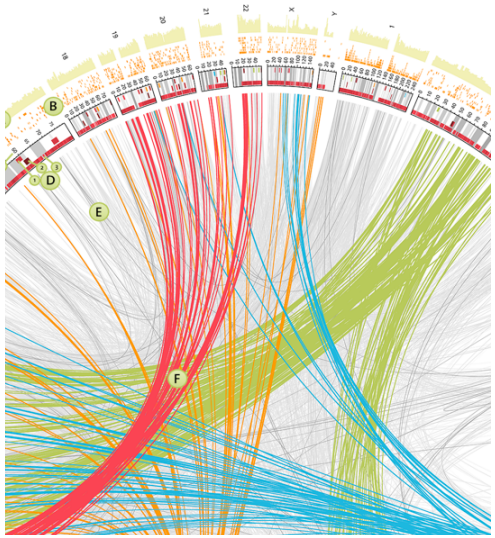
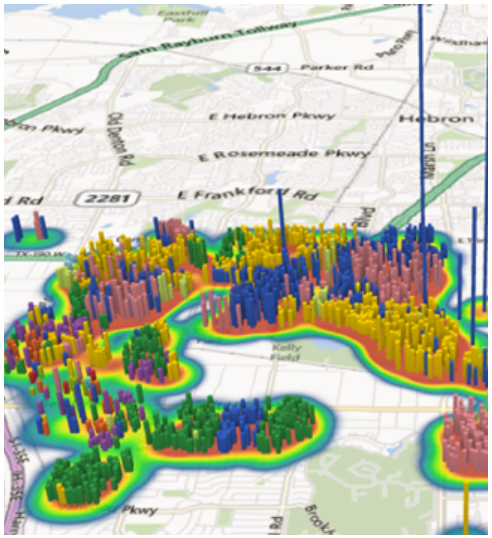
Big Data Analytics — Developing a MapReduce application

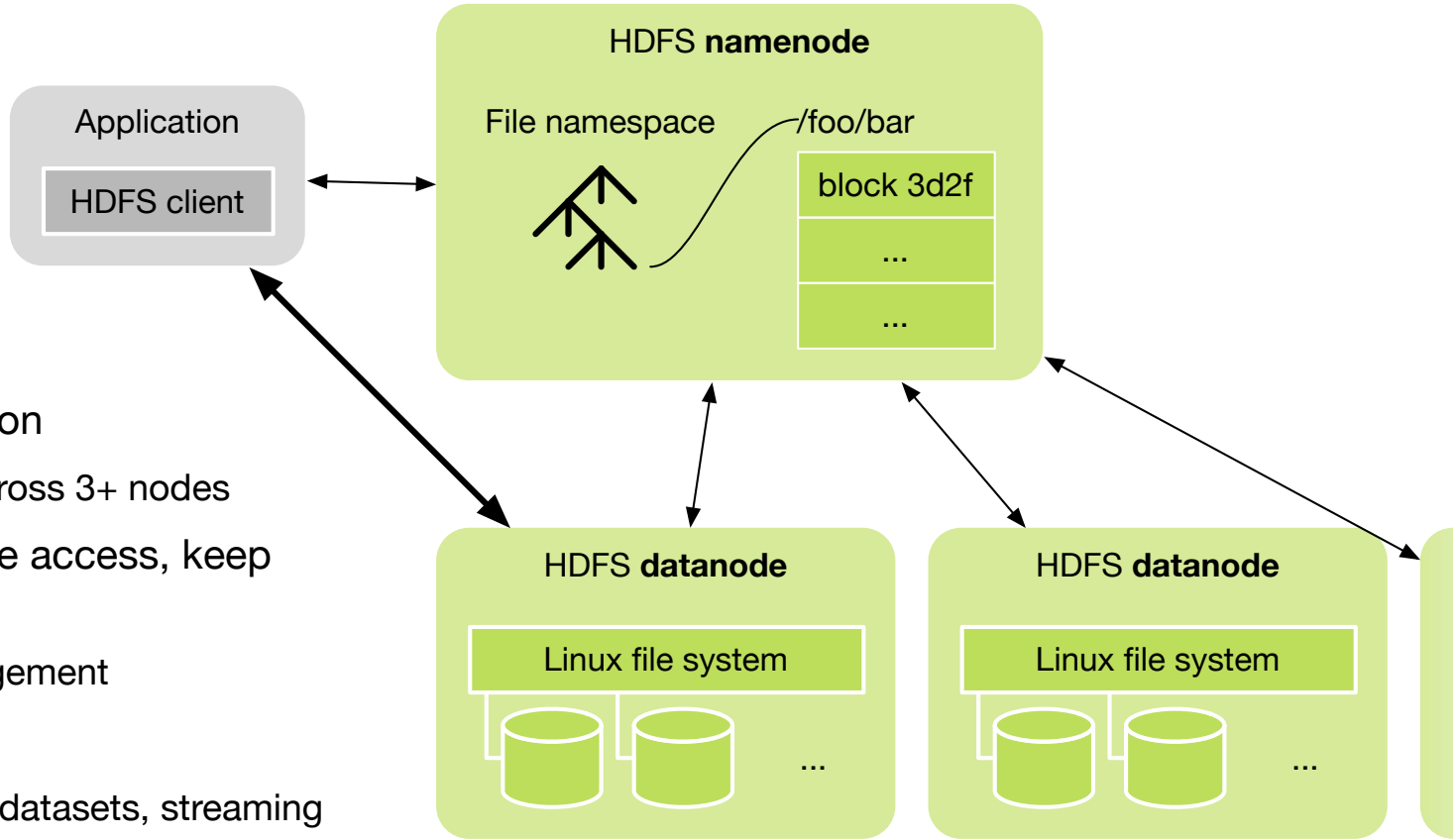


Prof. Marcel Graf | School of Business and Engineering Vaud (HEIG-VD)

Anatomy of a Hadoop cluster

Distributed file system HDFS

- HDFS design decisions:
  - Files stored as chunks
    - Fixed size (64MB)
  - Reliability through replication
    - Each chunk replicated across 3+ nodes
  - Single master to coordinate access, keep metadata
    - Simple centralized management
  - No data caching
    - Little benefit due to large datasets, streaming reads
  - Simplify the API
    - Push some of the issues onto the client (e.g., data layout)



Anatomy of a Hadoop cluster

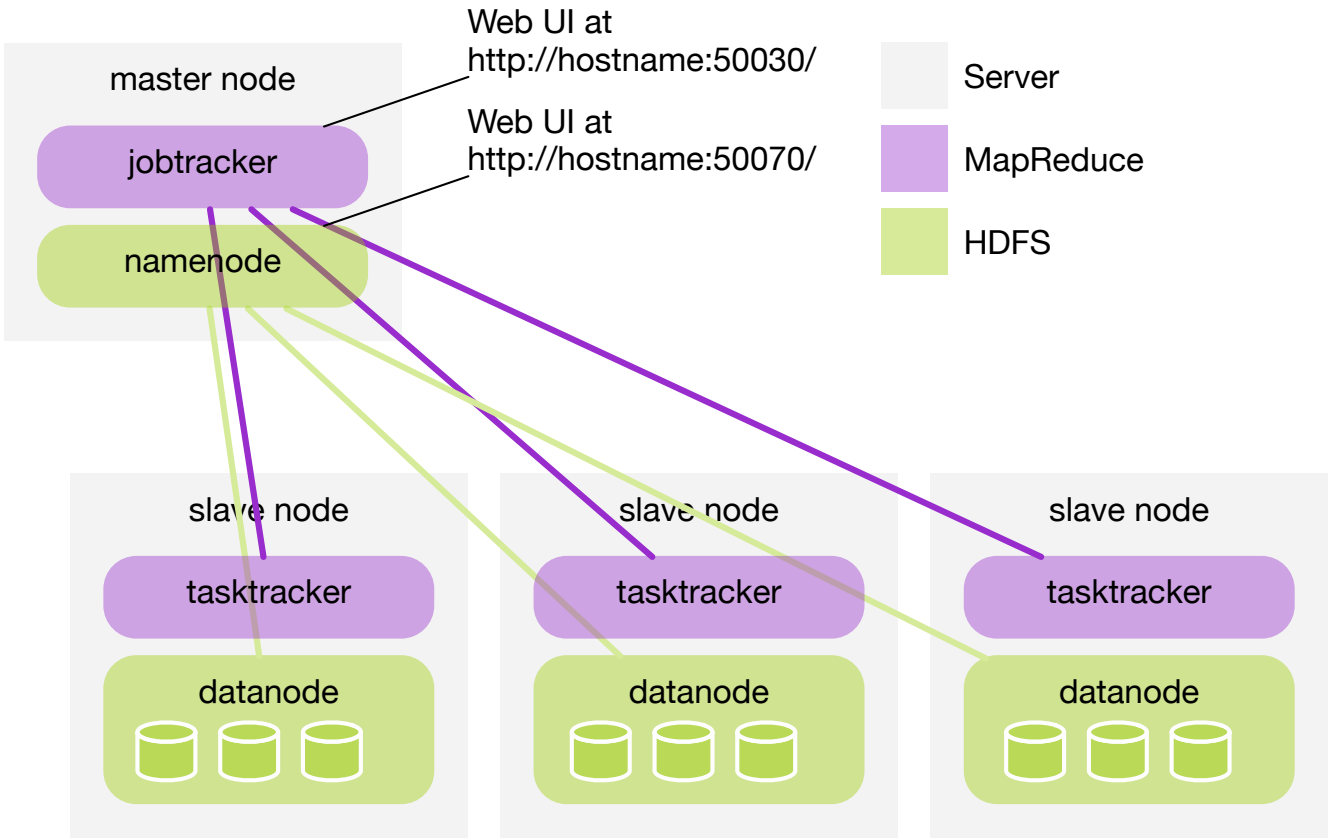
Namenode responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

Anatomy of a Hadoop cluster

Putting everything together

- Per cluster:
  - One Namenode (NN): master node for HDFS
  - One Jobtracker (JT): master node for job submission
- Per slave machine:
  - One Tasktracker (TT): contains multiple task slots
  - One Datanode (DN): serves HDFS data blocks



Anatomy of a Hadoop cluster

Web UI of the jobtracker

Quick Links

ip-10-250-110-47 Hadoop Map/Reduce Administration

State: RUNNING
Started: Sat Apr 11 08:11:53 EDT 2009
Version: 0.20.0, r763504
Compiled: Thu Apr 9 05:18:40 UTC 2009 by ndaley
Identifier: 200904110811

Cluster Summary (Heap Size is 53.75 MB/888.94 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
53	30	2	11	88	88	16.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0002	NORMAL	root	Max temperature	47.52%	101	48	15.25%	30	0	NA

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0001	NORMAL	gonzo	word count	100.00%	14	14	100.00%	30	30	NA

Failed Jobs

none

Local Logs

Log directory, Job Tracker History

Hadoop, 2009.

Anatomy of a Hadoop cluster

Web UI of the jobtracker — Job detail

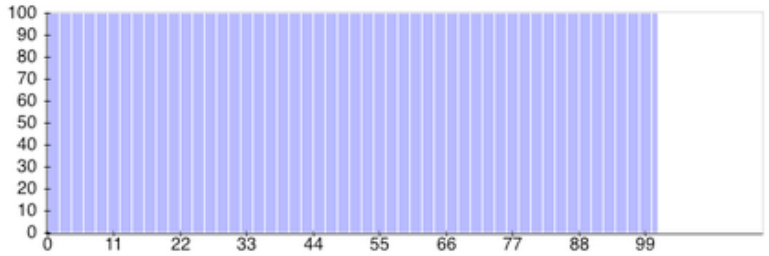
Hadoop job_200904110811_0002 on ip-10-250-110-47

User: root
Job Name: Max temperature
Job File: hdfs://ip-10-250-110-47.ec2.internal/mnt/hadoop/mapred/system/job_200904110811_0002/job.xml
Job Setup: Successful
Status: Running
Started at: Sat Apr 11 08:15:53 EDT 2009
Running for: 5mins, 38sec
Job Cleanup: Pending

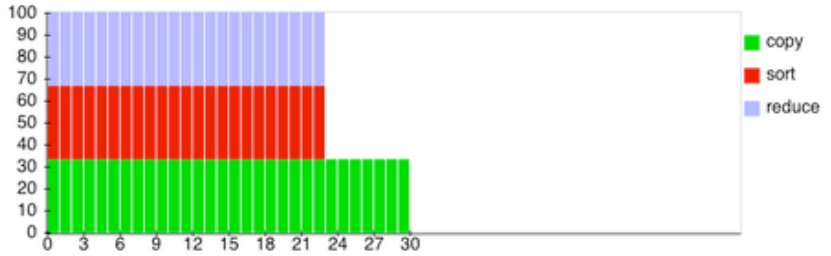
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	101	0	0	101	0	0 / 26
reduce	70.74%	30	0	13	17	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	32
	Rack-local map tasks	0	0	82
	Launched map tasks	0	0	127
	Data-local map tasks	0	0	45
FileSystemCounters	FILE_BYTES_READ	12,665,901	564	12,666,465
	HDFS_BYTES_READ	33,485,841,275	0	33,485,841,275
	FILE_BYTES_WRITTEN	988,084	564	988,648
	HDFS_BYTES_WRITTEN	0	360	360
Map-Reduce Framework	Reduce input groups	0	40	40
	Combine output records	4,489	0	4,489
	Map input records	1,209,901,509	0	1,209,901,509
	Reduce shuffle bytes	0	18,397	18,397
	Reduce output records	0	40	40
	Spilled Records	9,378	42	9,420
	Map output bytes	10,282,306,995	0	10,282,306,995
	Map input bytes	274,600,205,558	0	274,600,205,558
	Map output records	1,142,478,555	0	1,142,478,555
	Combine input records	1,142,482,941	0	1,142,482,941
	Reduce input records	0	42	42

Map Completion Graph - close



Reduce Completion Graph - close



Go back to JobTracker

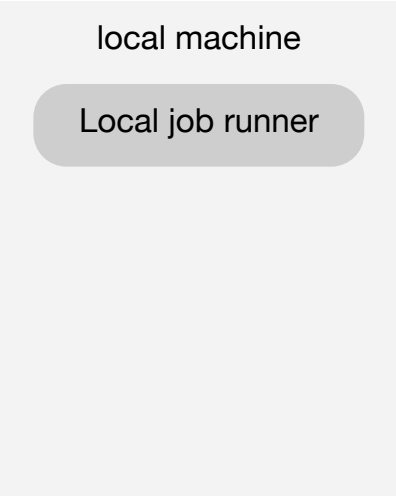
Hadoop, 2009.

Developing a MapReduce application

Special configurations for development

- For developing MapReduce applications it is recommended to start on a simplified configuration, and only after the code is thoroughly debugged deploy on a cluster.

- **Standalone mode**
  - No HDFS: I/O to local file system instead
  - A single JVM executes all phases of a MapReduce job
  - System.print() output will be displayed on standard output



- **Pseudo-distributed mode**
  - A one-node cluster, where the node is both master and slave

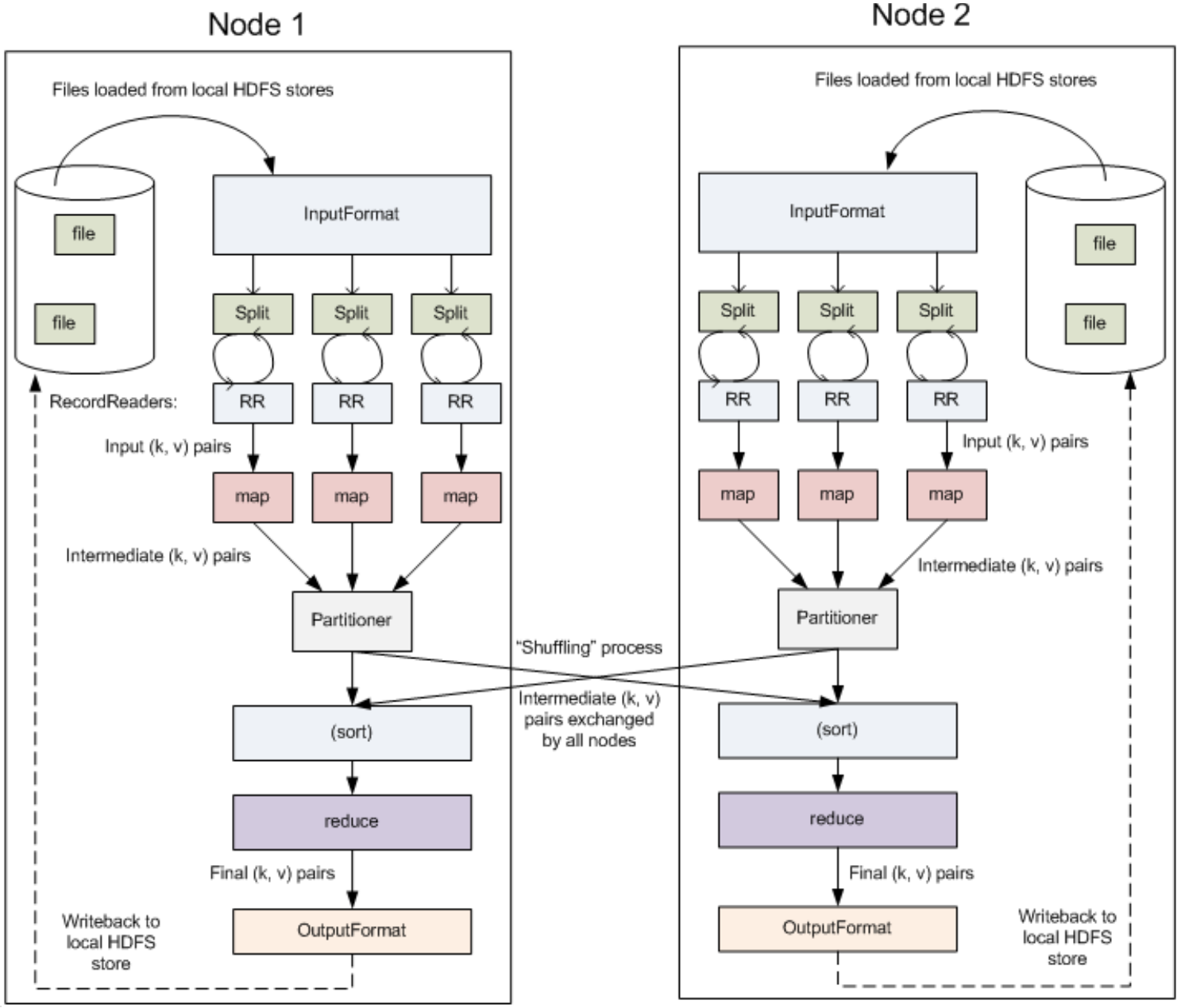
Developing a MapReduce application

Example output of executing a job in standalone mode

```
% export HADOOP_CLASSPATH=hadoop-examples.jar
% hadoop MaxTemperature input/ncdc/sample.txt output
12/02/04 11:50:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
12/02/04 11:50:41 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should
implement Tool for the same.
12/02/04 11:50:41 INFO input.FileInputFormat: Total input paths to process : 1 12/02/04 11:50:41 INFO
mapred.JobClient: Running job: job_local_0001
12/02/04 11:50:41 INFO mapred.Task: Using ResourceCalculatorPlugin : null
12/02/04 11:50:41 INFO mapred.MapTask: io.sort.mb = 100
12/02/04 11:50:42 INFO mapred.MapTask: data buffer = 79691776/99614720
12/02/04 11:50:42 INFO mapred.MapTask: record buffer = 262144/327680
12/02/04 11:50:42 INFO mapred.MapTask: Starting flush of map output
12/02/04 11:50:42 INFO mapred.MapTask: Finished spill 0
12/02/04 11:50:42 INFO mapred.Task: Task:attempt_local_0001_m_000000_0 is done. And i s in the process of commiting
12/02/04 11:50:42 INFO mapred.JobClient: map 0% reduce 0%
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:44 INFO mapred.Task: Task 'attempt_local_0001_m_000000_0' done.
12/02/04 11:50:44 INFO mapred.Task: Using ResourceCalculatorPlugin : null
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:44 INFO mapred.Merger: Merging 1 sorted segments
12/02/04 11:50:44 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 57 bytes
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:45 INFO mapred.Task: Task:attempt_local_0001_r_000000_0 is done. And
is in the process of commiting
12/02/04 11:50:45 INFO mapred.LocalJobRunner:
12/02/04 11:50:45 INFO mapred.Task: Task attempt_local_0001_r_000000_0 is allowed to commit now
12/02/04 11:50:45 INFO output.FileOutputCommitter: Saved output of task 'attempt_local _0001_r_000000_0' to output
12/02/04 11:50:45 INFO mapred.JobClient: map 100% reduce 0%
12/02/04 11:50:47 INFO mapred.LocalJobRunner: reduce > reduce
12/02/04 11:50:47 INFO mapred.Task: Task 'attempt_local_0001_r_000000_0' done.
12/02/04 11:50:48 INFO mapred.JobClient: map 100% reduce 100%
12/02/04 11:50:48 INFO mapred.JobClient: Job complete: job_local_0001
12/02/04 11:50:48 INFO mapred.JobClient: Counters: 17
12/02/04 11:50:48 INFO mapred.JobClient:   File Output Format Counters
12/02/04 11:50:48 INFO mapred.JobClient:     Bytes Written=29
12/02/04 11:50:48 INFO mapred.JobClient:   FileSystemCounters
12/02/04 11:50:48 INFO mapred.JobClient:     FILE_BYTES_READ=357503
```

MapReduce

Detailed data flow



MapReduce
Details — Input files

- The input files contain the data for a MapReduce program.
- Typically they are located in a distributed file system (HDFS, Amazon S3, ...).
- The format of the input file can be arbitrary
  - Log files in text format
  - Binary files
  - Records over several lines
  - ...
- The files can be very big, several tens of GB or more.

MapReduce

Details — *InputFormat*

- The way the files are divided into pieces and read is defined by the *InputFormat*.
- *InputFormat* is a class that
  - selects the files that will be used as input
  - defines the *InputSplits* which divide a file
  - provides a factory for *RecordReader* objects which will read the file.
- Hadoop ships with a number of predefined *InputFormats*

InputFormat	Description	Key	Value
TextInputFormat	Default format. Reads the lines of a text file.	The offset in bytes of the line in the file.	The line.
KeyValueInputFormat	Parses the lines of a text file into key-value pairs.	Everything before the first tab character.	The remainder of the line.
SequenceFileInputFormat	A high-performance binary format specific to Hadoop.	User-defined.	User-defined.

MapReduce

Details — *InputSplit*

- The class *InputSplit* describes a unit of work which corresponds to a *Map* task in a MapReduce program.
- By default the *InputFormat* divides a file into *splits* of 64 MB (which corresponds to the size of a *chunk* in HDFS).
- By dividing the file into *splits* several *Map* tasks are able to work on parallel on the same file.
  - If the file is very large this improves performance significantly.
- Each *Map* task corresponds to a single *split*.

MapReduce

Details — *RecordReader*

- The *InputSplit* defines a piece of data to process, but does not specify how to access it.
- The class *RecordReader* reads the data from its source and converts it into key-value pairs which are ready to be consumed by a *Mapper.*
- The *RecordReader* is called repeatedly until the entire *split* is consumed.
  - Each invocation of the *RecordReader* leads to another call of the *Map* function written by the developer.

MapReduce

Details — *Mapper* and *Reducer*

- The *Mapper* carries out the user-defined processing in the first phase of the MapReduce program.
  - A new *Mapper* instance is created for each *split*.
- The *Reducer* carries out the user-defined processing in the second phase of the MapReduce program.
  - A new *Reducer* instance is created for each *partition*.
  - For each key in the *partition* the *Reducer* is called once.

MapReduce

Details — *OutputFormat*

- The class *OutputFormat* defines how the key-value pairs produced by the *Reducers* are written to the output files.
- Hadoop ships with *OutputFormats* which write files to HDFS or to the local disk.
- Each *Reducer* writes into its own file, but in a common directory.
- Hadoop ships with the following *OutputFormats:*

OutputFormat	Description
TextOutputFormat	Default format. Writes a key-value pair per line, the two parts being separated by a tab character.
SequenceFileOutputFormat	Writes binary files which can be read by subsequent MapReduce programs.
NullOutputFormat	Does not produce an output file.

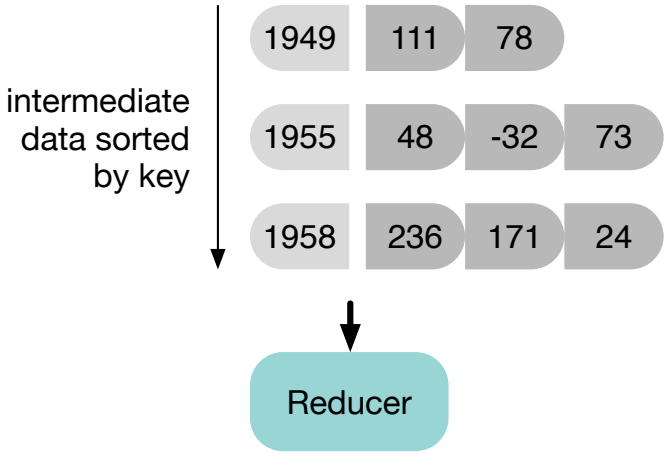
MapReduce

Details — *Partitioner*

- After the Mappers have run there are intermediate results stored on every node of the cluster, in key-value format. The framework now needs to prepare the next phase of processing by the Reducers.
  - The data needs to be transmitted between nodes, because values with the same key are guaranteed to be processed by the same Reducer.
  - Potentially there is a big number of keys.
  - One wants to execute several Reducers in parallel on the cluster. The developer specifies how many Reducer instances should be available.
  - The framework divides the key space into **partitions** and assigns to each partition one Reducer instance.
- Each mapper may emit key-value pairs to any partition.
- Therefore, the map nodes must all agree on where to send different pieces of intermediate data.
- The *partitioner* class determines which partition a given key-value pair will go to.
- The default partitioner computes a **hash value** for a given key and assigns it to a partition based on this result.

MapReduce
Details — Sort

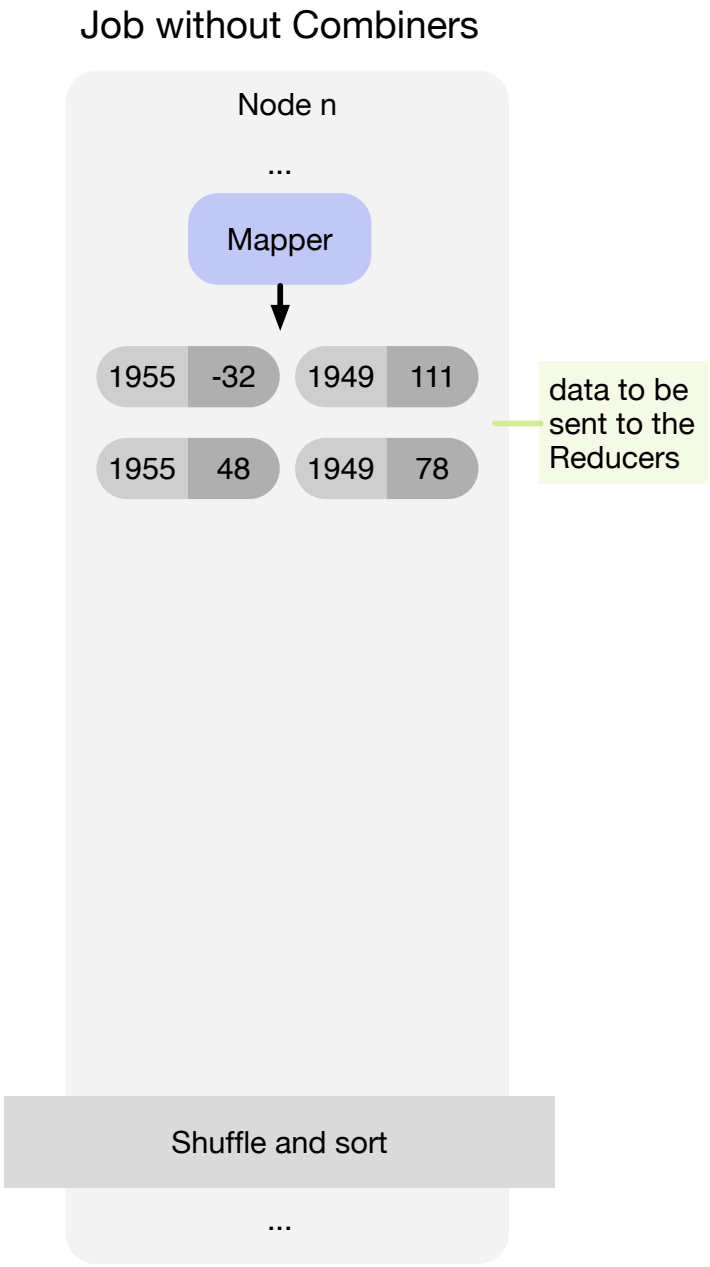
- Key-value pairs with the same key are guaranteed to arrive at the same Reducer.
- On the other hand Hadoop may use a Reducer to process more than one key value.
  - In that case Hadoop *sorts* the key-value pairs by key before presenting them to the Reducer.
  - The values however are not sorted.



MapReduce

Optimization of the Reduce phase with *Combiners*

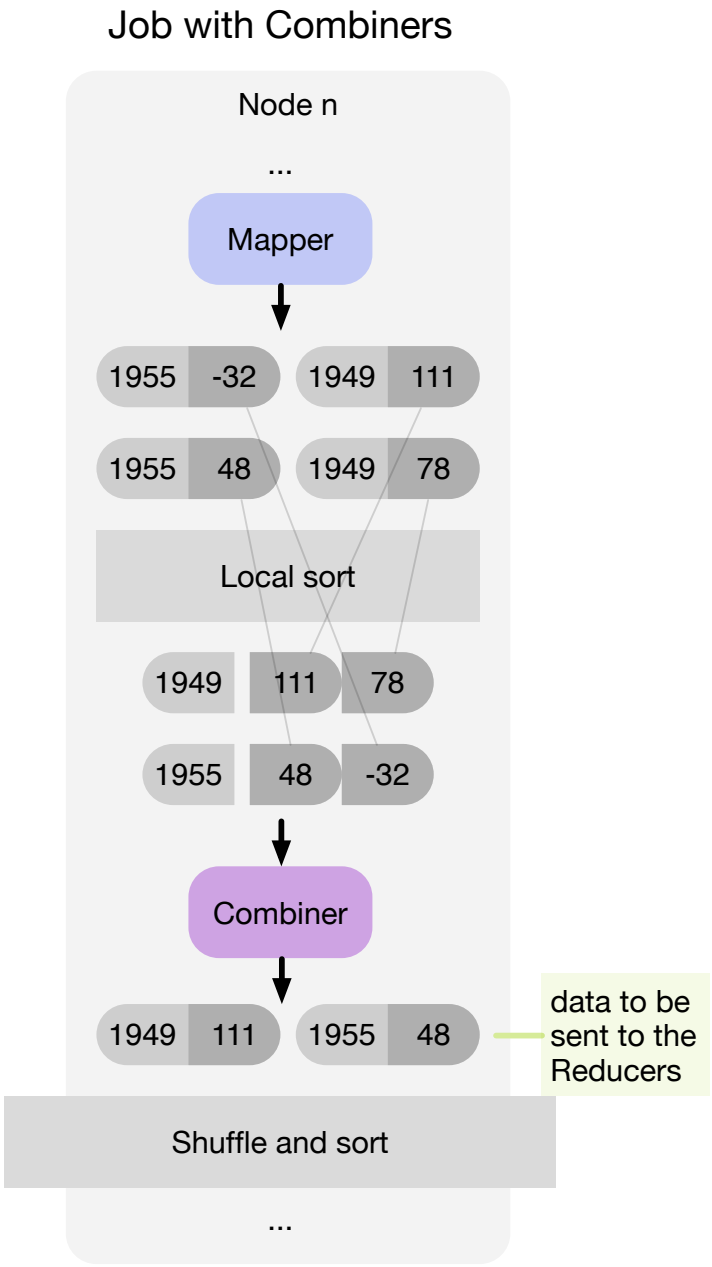
- The developer has the possibility to optimize the Reduce phase by writing a *Combiner*.
  - The Combiner is a kind of additional Reducer.



MapReduce

Optimization of the Reduce phase with *Combiners*

- If the developer has written a Combiner, the framework inserts it into the processing pipeline on the nodes that just have terminated the Map phase.
  - The Combiner executes after the Map phase, just before the intermediate data is sent to other nodes.
  - It receives the data that the Map phase produced **on the local node**.
  - It produces key-value pairs that will be sent to the Reducers instead of the original data.



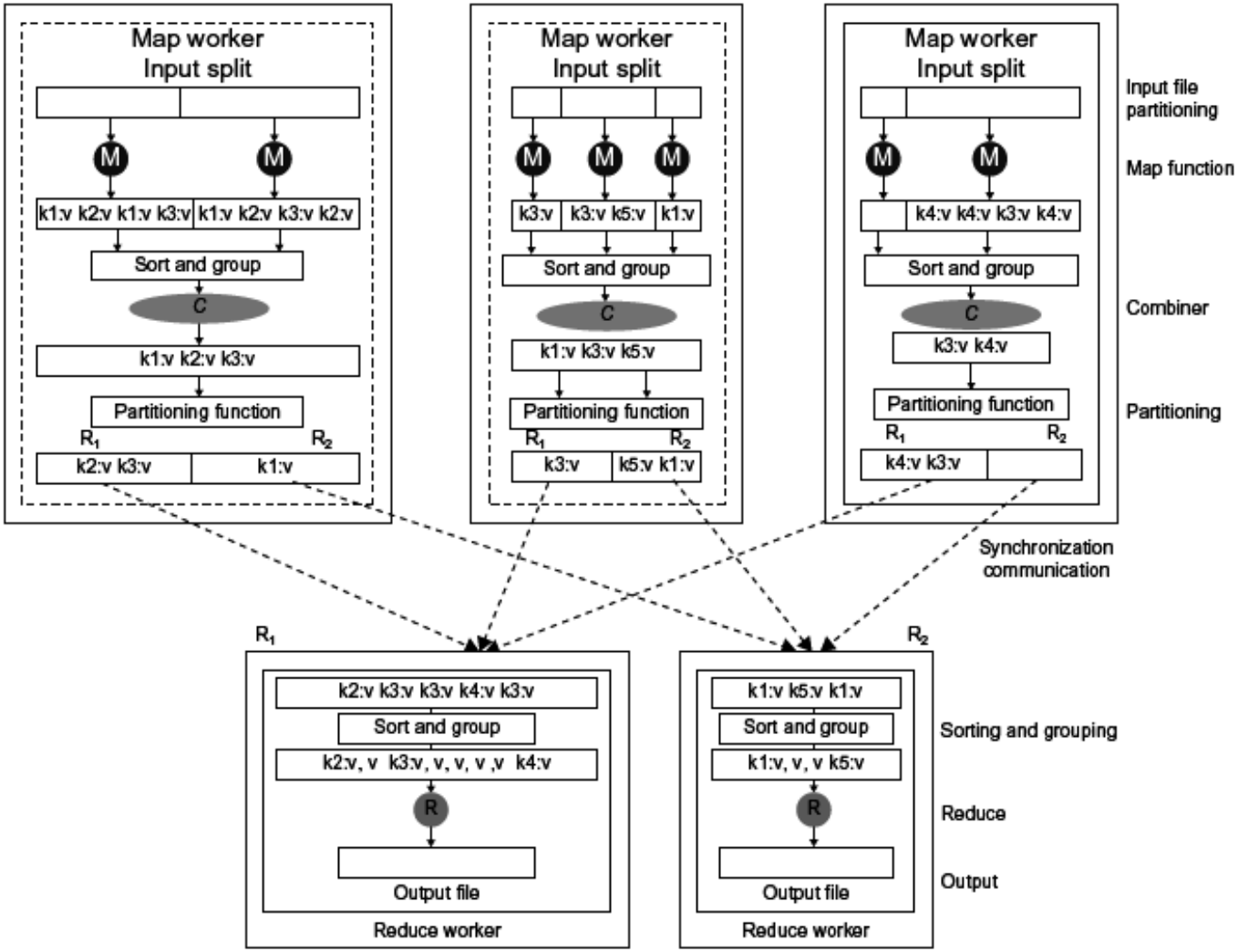
MapReduce

Optimization of the Reduce phase with *Combiners* — When to use

- The Combiner can be used in cases when one can already start reducing the data without having all the data available.
  - For example for determining the maximum temperature Combiners work very well.
  - The Combiner calculates the maximum temperature for the data on the local node.
  - Instead of sending the key-value pairs
    - (1955, -32), (1949, 111), (1955, 48) and (1949, 78) to the Reducers, one sends only
    - (1949, 111) and (1955, 48).
- Combiners implement the same API as Reducers.

MapReduce

Complete data flow with Partitioners and Combiners



Anatomy of a Hadoop cluster

Important counters

Phase	Measure	Counter name
Map	Number of input records consumed by all mappers	Map input records
	Number of key/value pairs produced by all mappers	Map output records
Shuffle and sort	The number of bytes of map output copied by the shuffle to reducers (may be compressed)	Reduce shuffle bytes
Reduce	Number of unique keys fed into the reducers	Reduce input groups
	Number of key/value pairs produced by all reducers	Reduce output records