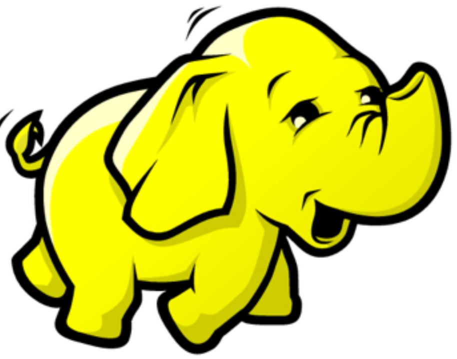
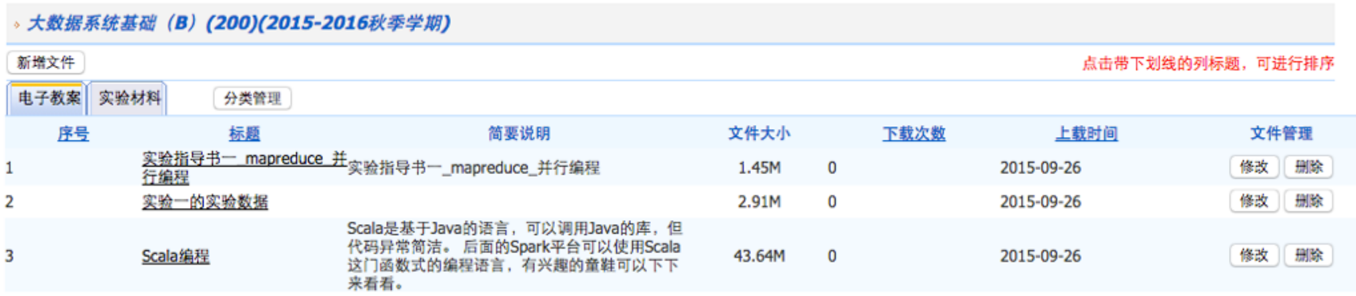
MapReduce / Hadoop



Zhi Wang
wangzhi@sz.tsinghua.edu.cn

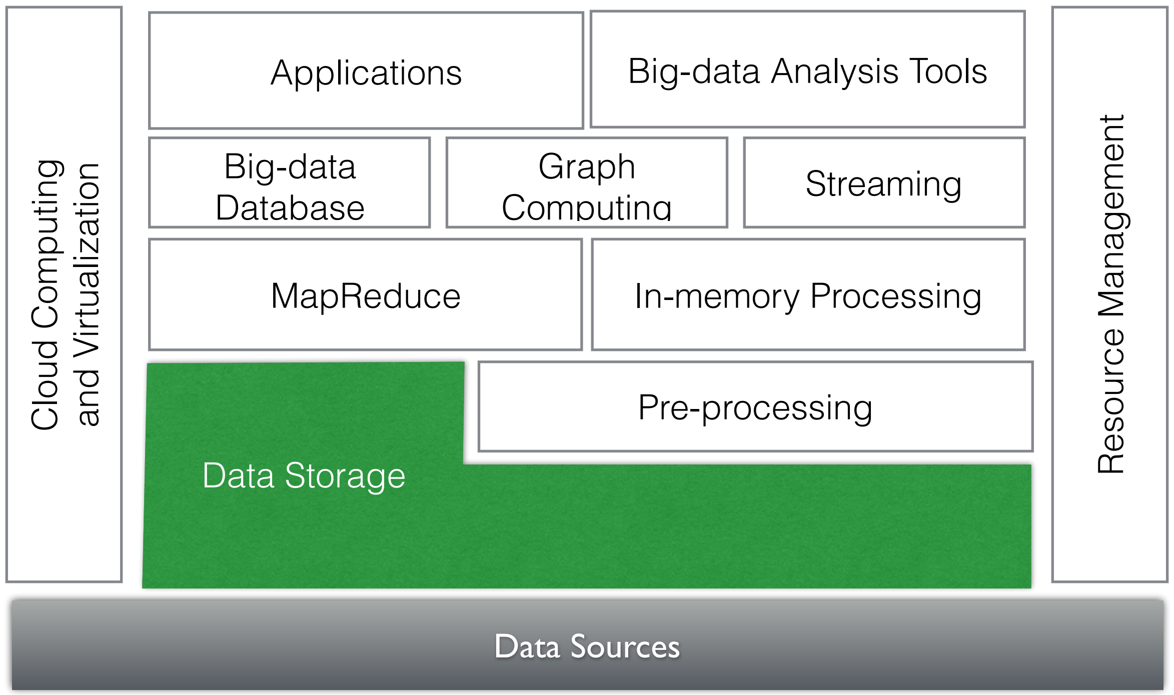
实验材料

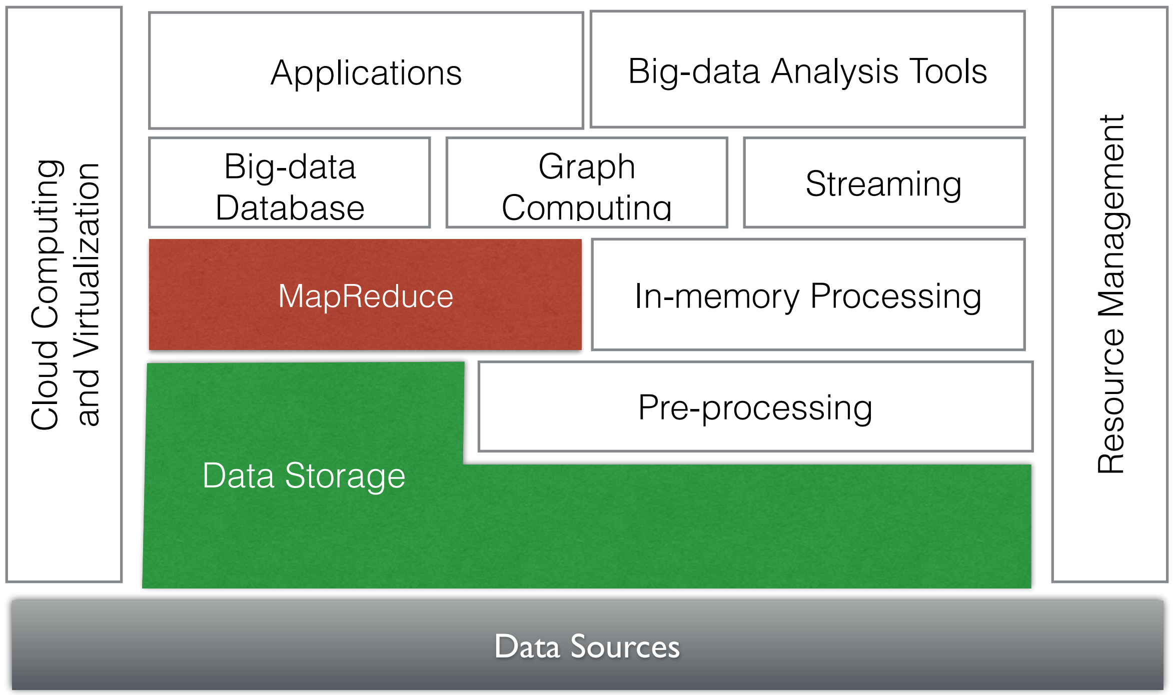


› 大数据系统基础（B）(200)(2015-2016秋季学期)

序号	标题	简要说明	文件大小	下载次数	上载时间	文件管理
1	实验指导书一_mapreduce_并行编程	实验指导书一_mapreduce_并行编程	1.45M	0	2015-09-26	修改 删除
2	实验一的实验数据		2.91M	0	2015-09-26	修改 删除
3	Scala编程	Scala是基于Java的语言，可以调用Java的库，但代码异常简洁。后面的Spark平台可以使用Scala这门函数式的编程语言，有兴趣的童鞋可以下下来看看。	43.64M	0	2015-09-26	修改 删除

第一次实验课安排：第五周（B205）





Outline

- MapReduce Basis
- MapReduce Programming Model
- Algorithms in MapReduce

Motivation: Large-scale Data Processing

- Many tasks: Process lots of data to produce other data
- Want to use hundreds or thousands of CPUs
  - but this needs to be easy

Typical Large-Data Problems

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Key idea: provide a functional abstraction for these two operations

What is MapReduce?

- MapReduce is a programming model Google has used successfully is processing its “big-data” sets (~ 20000 peta bytes per day)
- Users specify the computation in terms of a map and a reduce function,
- Underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, and
- Underlying system also handles machine failures, efficient communications, and performance issues.

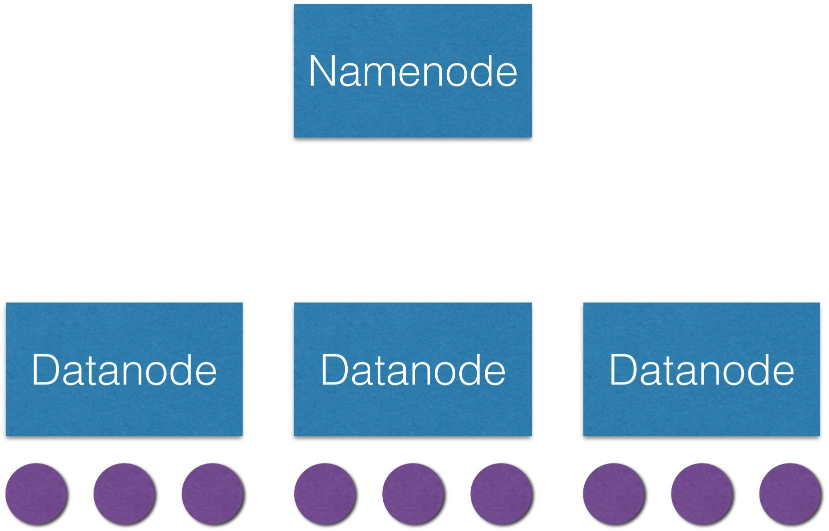
How MapReduce is Structured

- Functional programming meets distributed computing
- A batch data processing system
- Factors out many reliability concerns from application logic

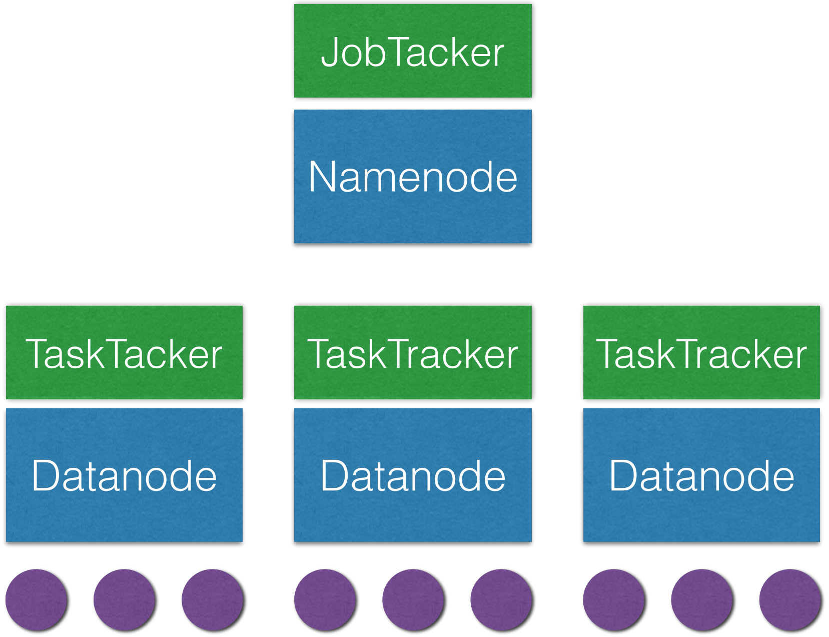
MapReduce Provides

- Automatic parallelization & distribution
- Fault-tolerance
- Status and monitoring tools
- A clean abstraction for programmers

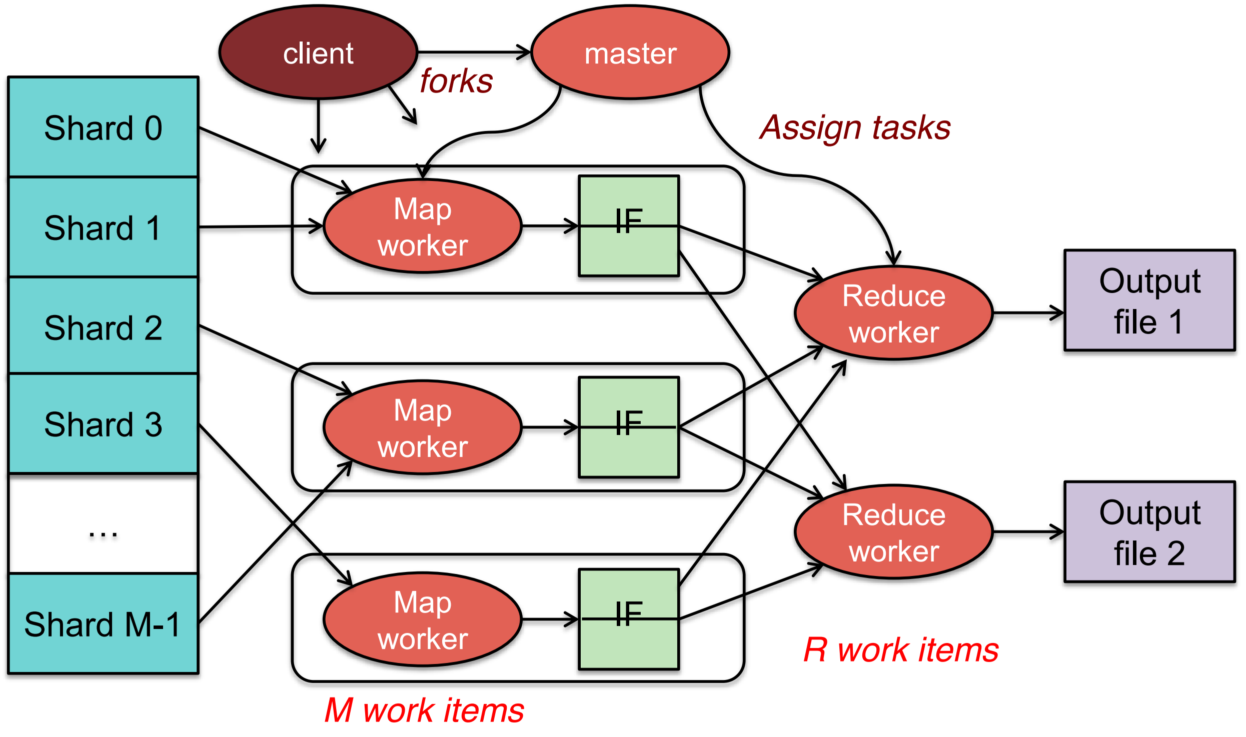
Architecture



Architecture



MapReduce: the complete picture



client

forks

master

Assign tasks

Shard 0

Shard 1

Shard 2

Shard 3

…

Shard M-1

Map worker

IF

Map worker

IF

Map worker

IF

Reduce worker

Reduce worker

Output file 1

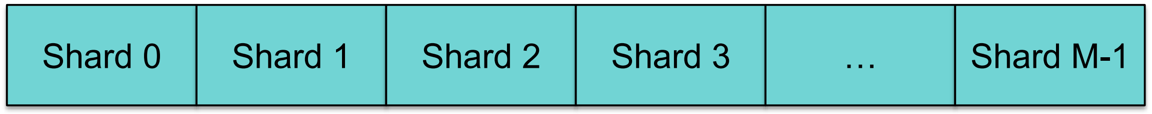
Output file 2

R work items

M work items

Step 1: Split input files into chunks (shards)

- Break up the input data into *M* pieces (typically 64 MB)



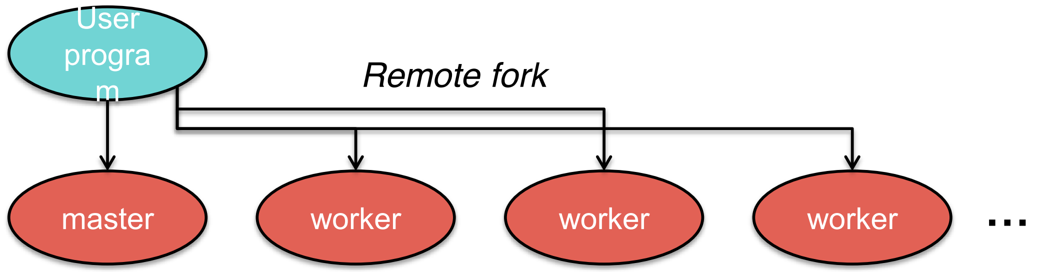
Shard 0	Shard 1	Shard 2	Shard 3	…	Shard M-1

Input files

Divided into *M* shards

Step 2: Fork processes

- Start up many copies of the program on a cluster of machines
  - 1 master: scheduler & coordinator
  - Lots of workers
- Idle workers are assigned either:
  - map tasks (each works on a shard) – there are *M* map tasks
  - reduce tasks (each works on intermediate files) – there are *R*
    - *R* = # partitions, defined by the user



User progra m

Remote fork

master

worker

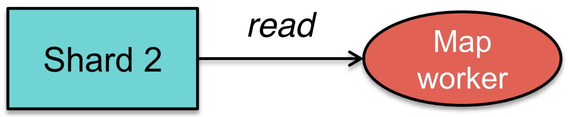
worker

worker

…

Step 3: Map Task

- Reads contents of the input shard assigned to it
- Parses key/value pairs out of the input data
- Passes each pair to a user-defined *map* function
  - Produces intermediate key/value pairs
  - These are buffered in memory



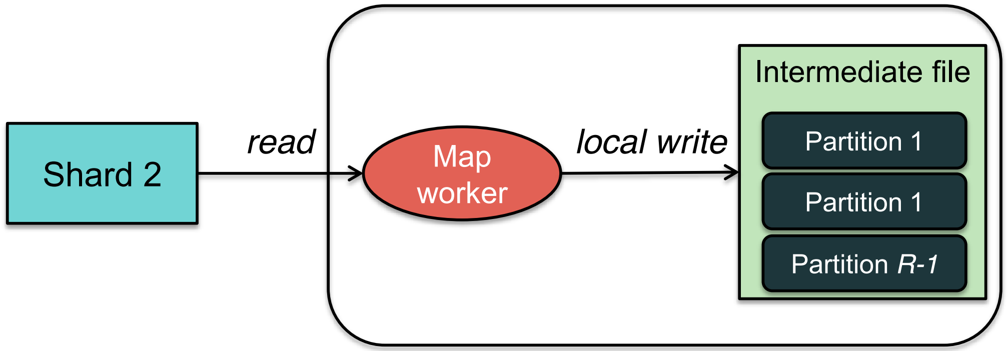
Shard 2

read

Map worker

Step 4: Create intermediate files

- Intermediate key/value pairs produced by the user's *map* function buffered in memory and are periodically written to the local disk
  - Partitioned into *R* regions by a partitioning function

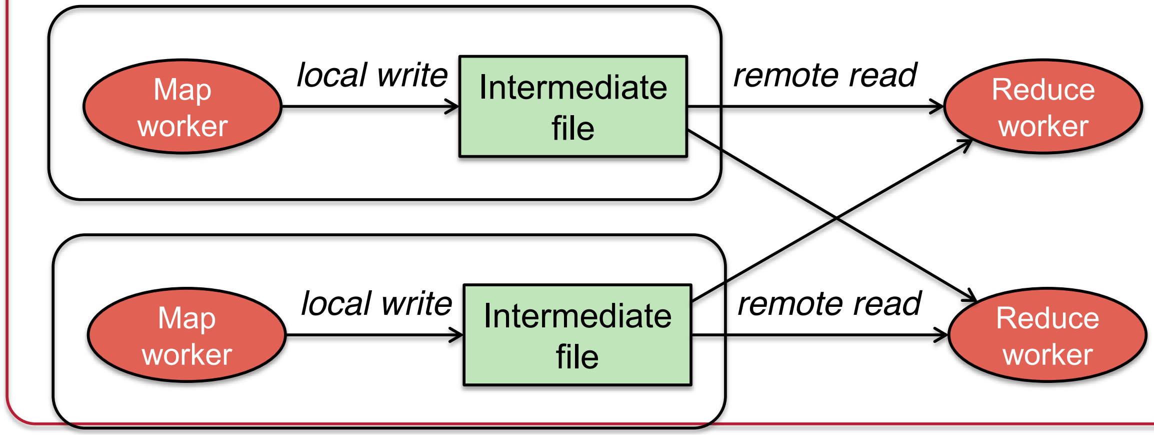


Step 4a. Partitioning

- Map data will be processed by Reduce workers
  - The user's *Reduce* function will be called once per unique key generated by *Map*.
- This means we will need to sort all the (key, value) data by keys and decide which Reduce worker processes which keys – the Reduce worker will do this
- Partition function: decides which of *R* reduce workers will work on which key
  - Default function: *hash(key) mod R*
  - Map worker partitions the data by keys
- Each Reduce worker will read their partition from every Map worker

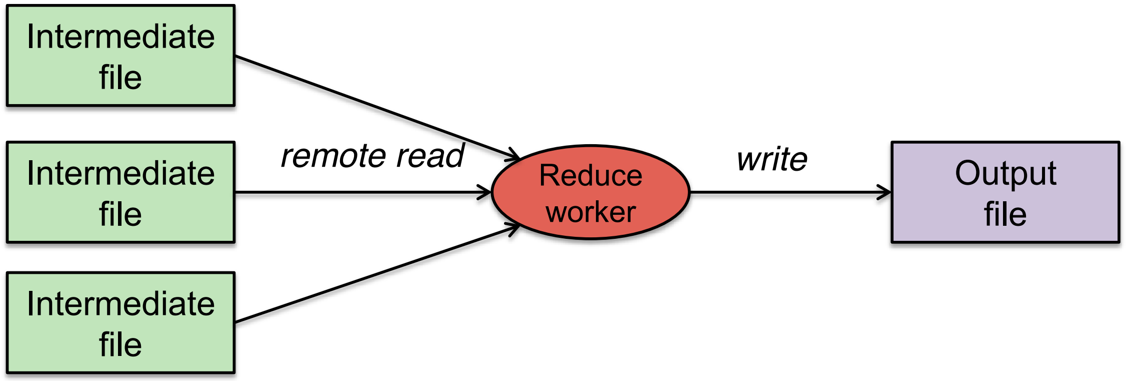
Step 5: Reduce Task: sorting

- Reduce worker gets notified by the master about the location of intermediate files for its partition
- Uses RPCs to read the data from the local disks of the map workers
- When the *reduce* worker reads intermediate data for its partition
  - It sorts the data by the intermediate keys
  - All occurrences of the same key are grouped together



Step 6: Reduce Task: *Reduce*

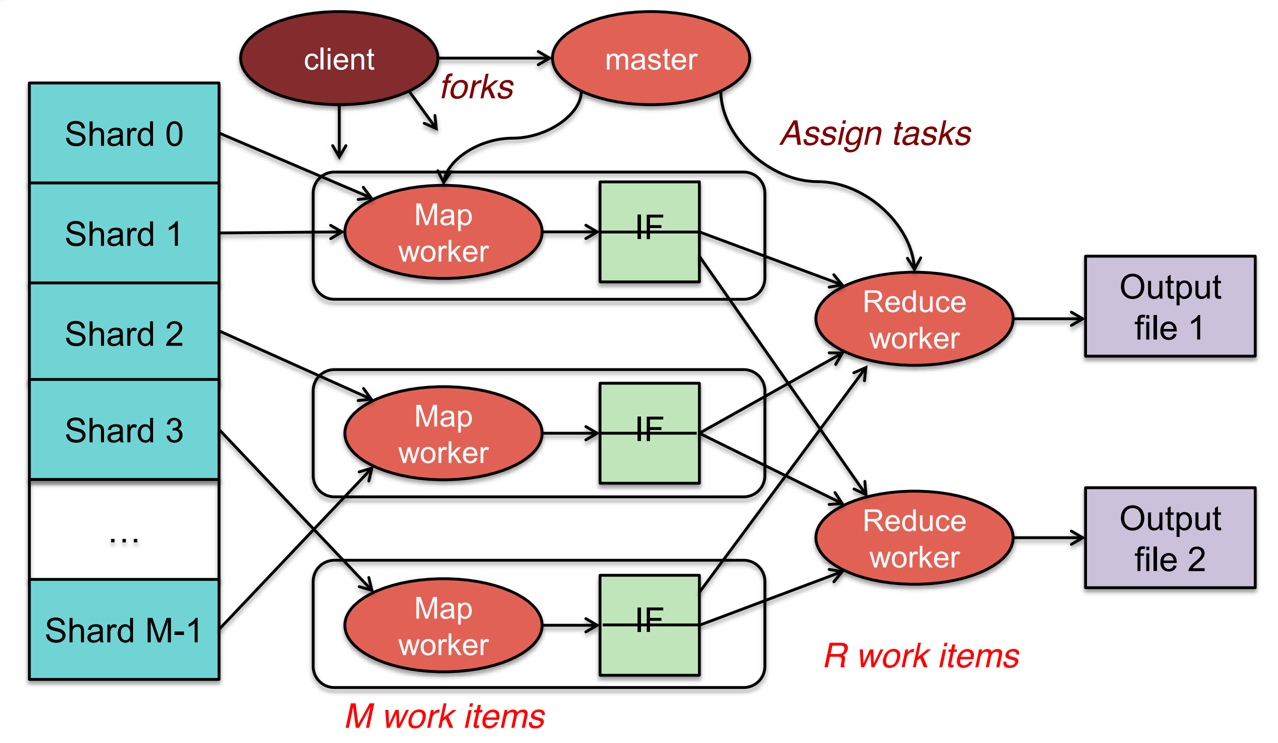
- The sort phase grouped data with a unique intermediate key
- User's *Reduce* function is given the key and the set of intermediate values for that key
  - < key, (value1, value2, value3, value4, …) >
- The output of the *Reduce* function is appended to an output file



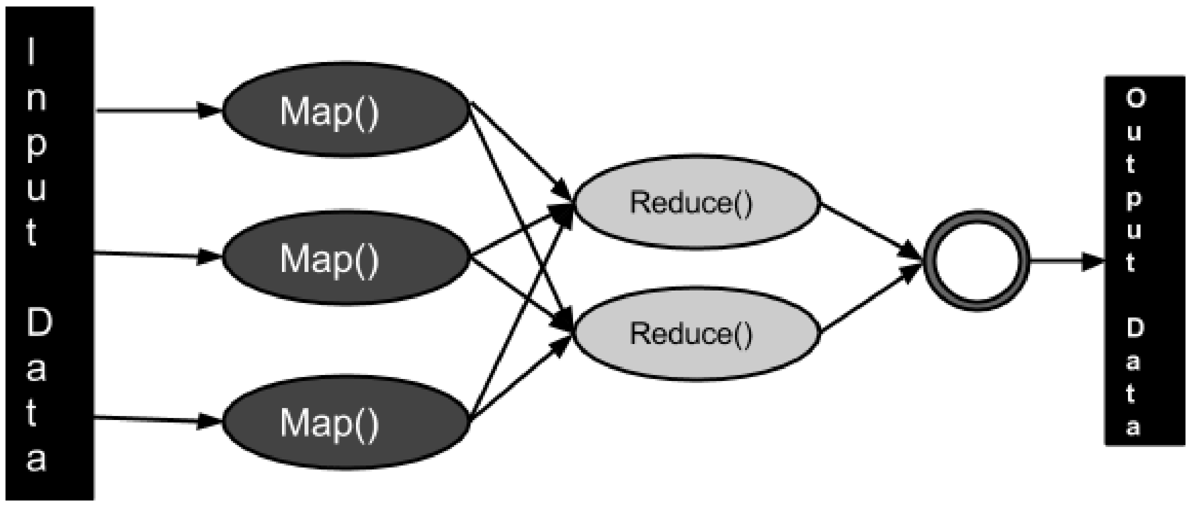
Step 7: Return to user

- When all *map* and *reduce* tasks have completed, the master wakes up the user program
- The *MapReduce* call in the user program returns and the program can resume execution.
  - Output of *MapReduce* is available in *R* output files

MapReduce: the complete picture



Paradigm



	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

Z. Wang, Foundations for Big Data Systems 2015

Example

It will be seen that this mere painstaking burrower and grub-worm of a poor devil of a Sub-Sub appears to have gone through the long Vaticans and street-stalls of the earth, picking up whatever random allusions to whales he could anyways find in any book whatsoever, sacred or profane. Therefore you must not, in every case at least, take the higgledy-piggledy whale statements, however authentic, in these extracts, for veritable gospel cetology. Far from it. As touching the ancient authors generally, as well as the poets here appearing, these extracts are solely valuable or entertaining, as affording a glancing bird's eye view of what has been promiscuously said, thought, fancied, and sung of Leviathan, by many nations and generations, including our own.

MAP

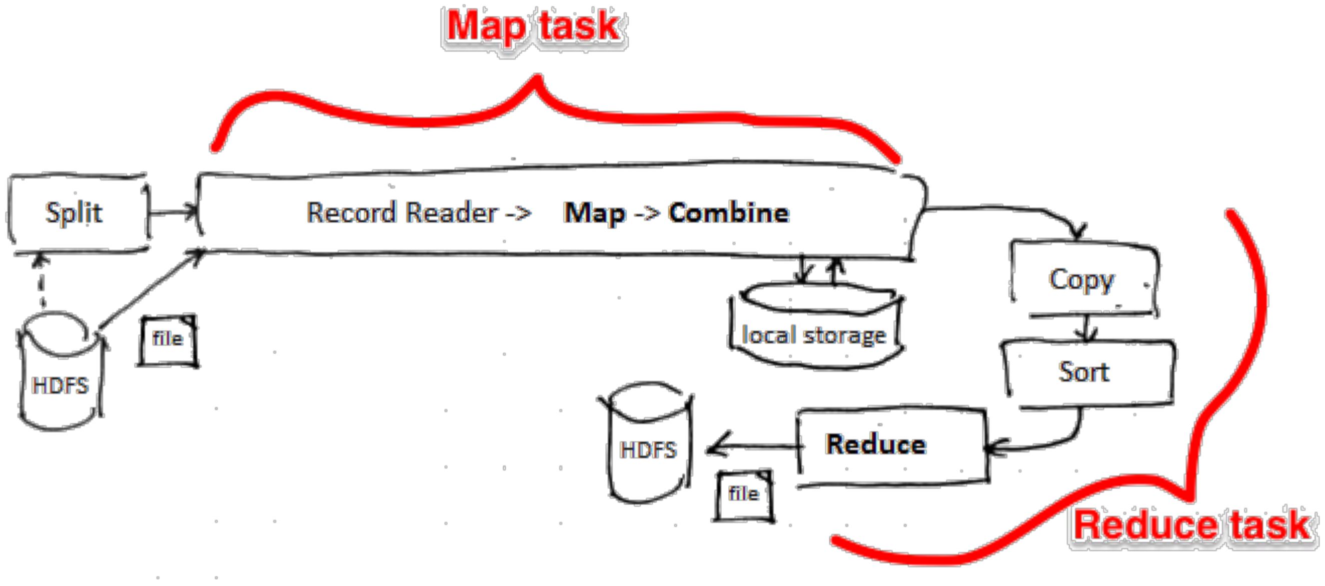
it 1
will 1
be 1
seen 1
that 1
this 1
mere 1
painstaking 1
burrower 1
and 1
grub-worm 1
of 1
a 1
poor 1
devil 1
of 1
a 1
sub-sub 1
appears 1
to 1
have 1
gone 1

…
a 1
a 1
aback 1
aback 1
abaft 1
abaft 1
abandon 1
abandon 1
abandon 1
abandoned 1
abandoned 1
abandoned 1
abandoned 1
abandoned 1
abandoned 1
abandoned 1
abandonedly 1
abandonment 1
abandonment 1
abased 1
abased 1

REDUCE

a 4736
aback 2
abaft 2
abandon 3
abandoned 7
abandonedly 1
abandonment 2
abased 2
abasement 1
abashed 2
abate 1
abated 3
abatement 1
abating 2
abbreviate 1
abbreviation 1
abeam 1
abed 2
abednego 1
abel 1
abhorred 3
abhorrence 1

MapReduce: whole picture



Map task

Split

Record Reader -> **Map** -> **Combine**

HDFS

file

local storage

Copy

Sort

Reduce

HDFS

file

Reduce task

Movie of this week

The Trueman Show (1998)