Map-reduce, Hadoop and The communication bottleneck

Yoav Freund

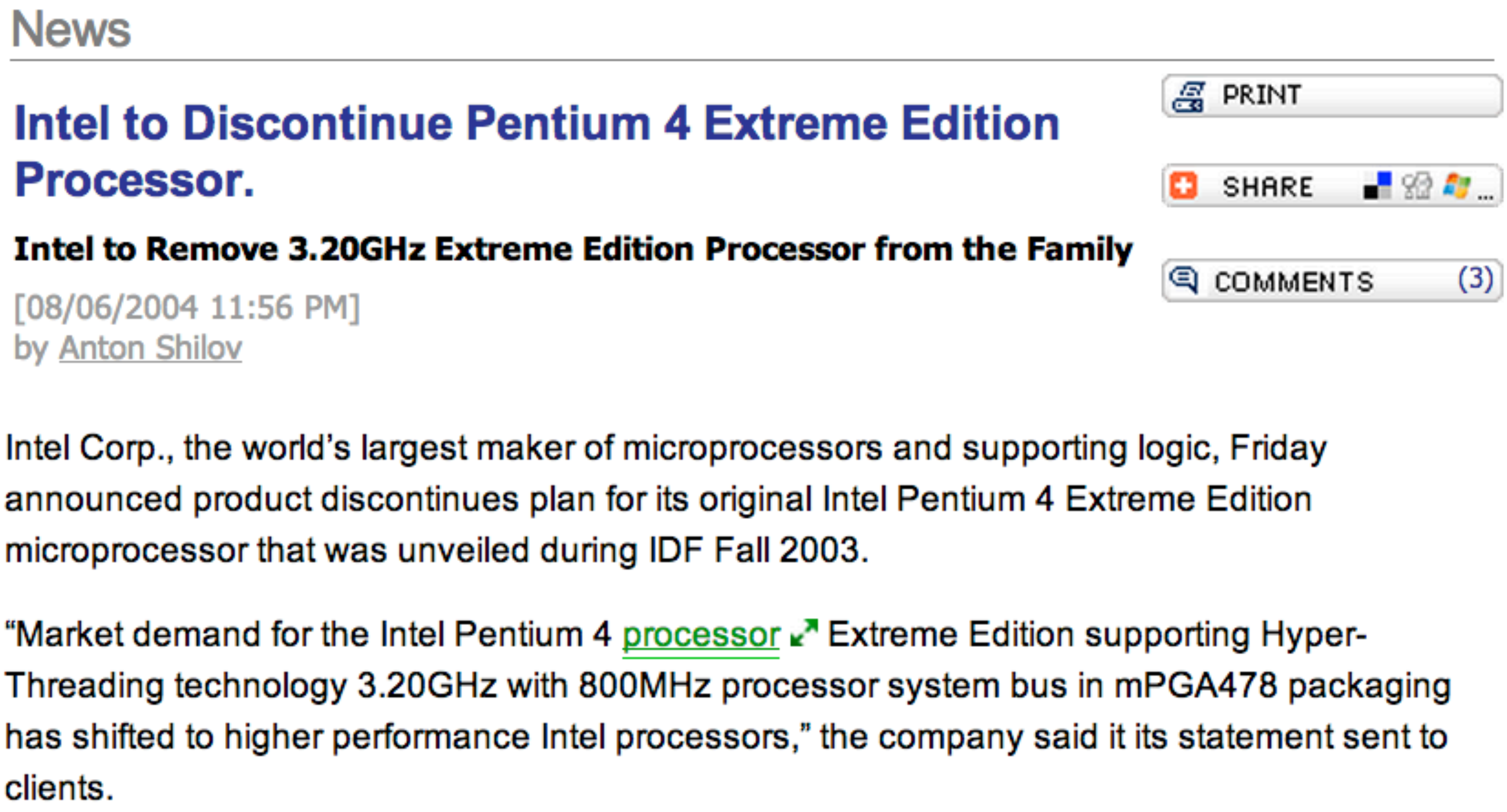
UCSD /

Computer Science and Engineering

Plan of the talk

- Why is Hadoop so popular?
- HDFS
- Map Reduce
- Word Count – example using Hadoop streaming and python.
- Computing pairwise interactions using map-reduce.
- The communication bottleneck: sort and shuffle.
- K-means analysis.
- Spark

How did hadoop become so popular?



News

Intel to Discontinue Pentium 4 Extreme Edition Processor.

Intel to Remove 3.20GHz Extreme Edition Processor from the Family

[08/06/2004 11:56 PM]
by Anton Shilov

Intel Corp., the world's largest maker of microprocessors and supporting logic, Friday announced product discontinues plan for its original Intel Pentium 4 Extreme Edition microprocessor that was unveiled during IDF Fall 2003.

"Market demand for the Intel Pentium 4 processor Extreme Edition supporting Hyper-Threading technology 3.20GHz with 800MHz processor system bus in mPGA478 packaging has shifted to higher performance Intel processors," the company said it its statement sent to clients.

What happened?

- Going above 3.2Ghz proved impractical
  - The power wall
  - The temperature wall
- Number of transistors still doubles but they cannot all be powered.
- Multi-core CPU chips: 2,4,8,…
- Large computer clusters / data-centers / the cloud.
- Big Data

What is big data?

- So much data you cannot store it?
  - No!

Seagate Backup Plus 4 TB USB 3.0 Desktop External Hard Drive STCA4000100

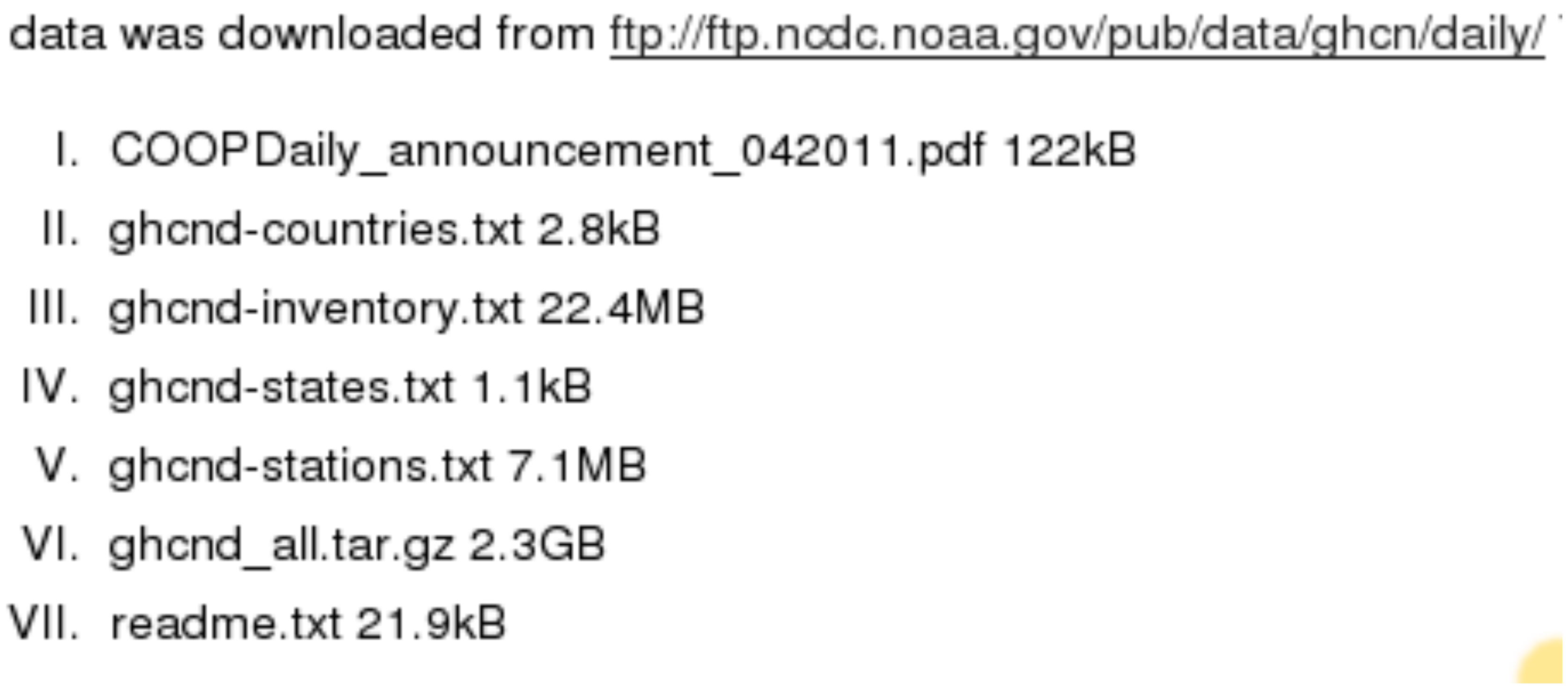
~~$239.99~~ $169.99 ✓Prime

- So much data you cannot process it?
  - No! Bandwidth of a single CPU is 100 GB/sec
- So much data you cannot **communicate** it?
  - YES!!

Prices of bandwidth in the memory hierarchy (October 2012)

	Cost	Power	Block Size	Bandwidth
L1-L2	On-Chip with CPU	low-end: iTouch: 1Watt high-end: Intel Core i7-950: 130Watt	L1:32-64 Bytes L2: 64-256 Bytes	100s of GB/sec
DRAM	8-16/GB	1-2 Watts/GB	max throughput at: 8-16KB	50-70 GB/sec
SSD	High end, $11/GB. Low end: $1-4/GB	high end: 0.15W/GB Low End: 0.05 W/GB	4KB	high end: 1-3GB/sec low end: .1-.2GB/sec
Disk	0.03-0.1$/GB	0.01W/GB	4KB	100MB/sec sequential 0.4-2.0MB/sec random Getting to each block takes 2-10ms

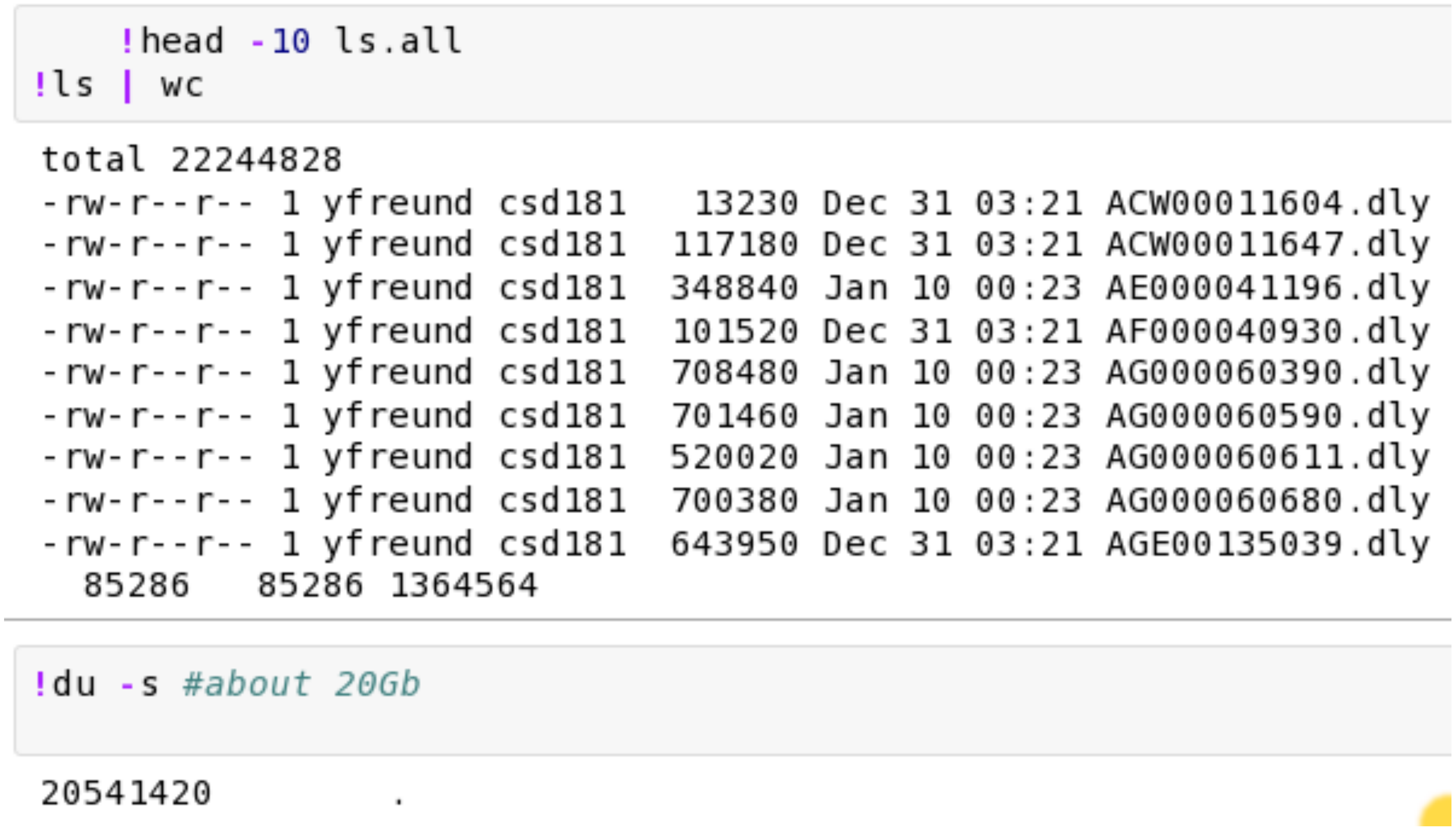
Weather Database



data was downloaded from ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/

I. COOPDaily_announcement_042011.pdf 122kB
II. ghcnd-countries.txt 2.8kB
III. ghcnd-inventory.txt 22.4MB
IV. ghcnd-states.txt 1.1kB
V. ghcnd-stations.txt 7.1MB
VI. ghcnd_all.tar.gz 2.3GB
VII. readme.txt 21.9kB

The format of the data



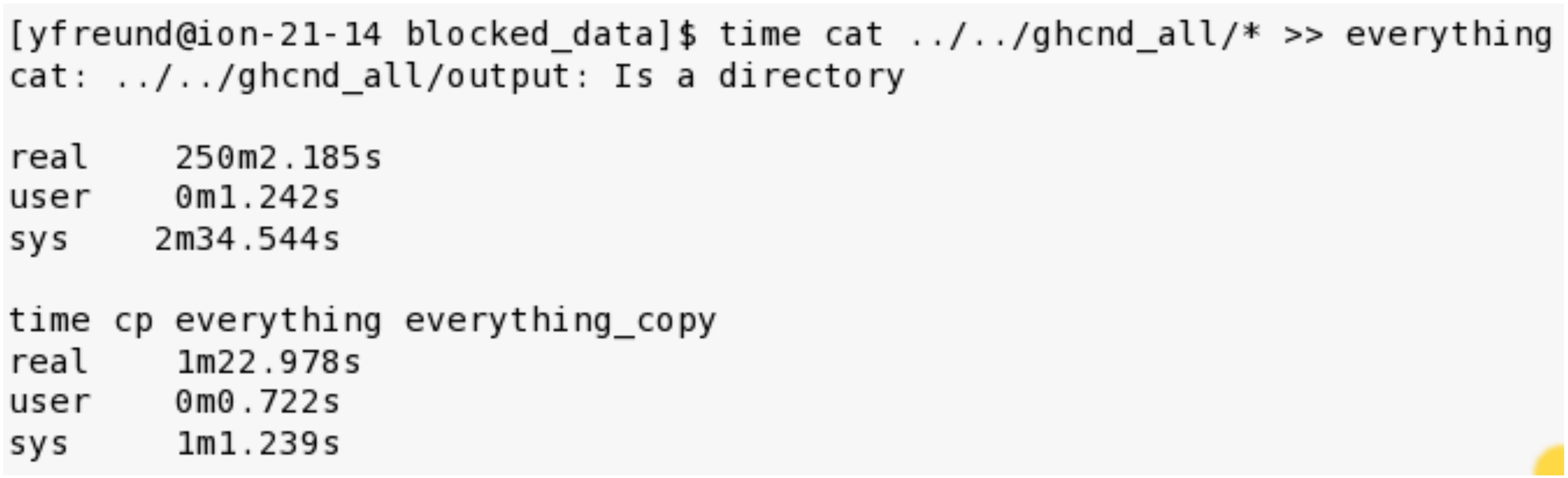
```
    !head -10 ls.all
!ls | wc
```

```
total 22244828
-rw-r--r-- 1 yfreund csd181   13230 Dec 31 03:21 ACW00011604.dly
-rw-r--r-- 1 yfreund csd181  117180 Dec 31 03:21 ACW00011647.dly
-rw-r--r-- 1 yfreund csd181  348840 Jan 10 00:23 AE000041196.dly
-rw-r--r-- 1 yfreund csd181  101520 Dec 31 03:21 AF000040930.dly
-rw-r--r-- 1 yfreund csd181  708480 Jan 10 00:23 AG000060390.dly
-rw-r--r-- 1 yfreund csd181  701460 Jan 10 00:23 AG000060590.dly
-rw-r--r-- 1 yfreund csd181  520020 Jan 10 00:23 AG000060611.dly
-rw-r--r-- 1 yfreund csd181  700380 Jan 10 00:23 AG000060680.dly
-rw-r--r-- 1 yfreund csd181  643950 Dec 31 03:21 AGE00135039.dly
  85286   85286 1364564
```

```
!du -s #about 20Gb
```

```
20541420        .
```

Reading many small files takes a long time



```
[yfreund@ion-21-14 blocked_data]$ time cat ../../ghcnd_all/* >> everything
cat: ../../ghcnd_all/output: Is a directory

real	250m2.185s
user	0m1.242s
sys	2m34.544s

time cp everything everything_copy
real	1m22.978s
user	0m0.722s
sys	1m1.239s
```

The disk drive

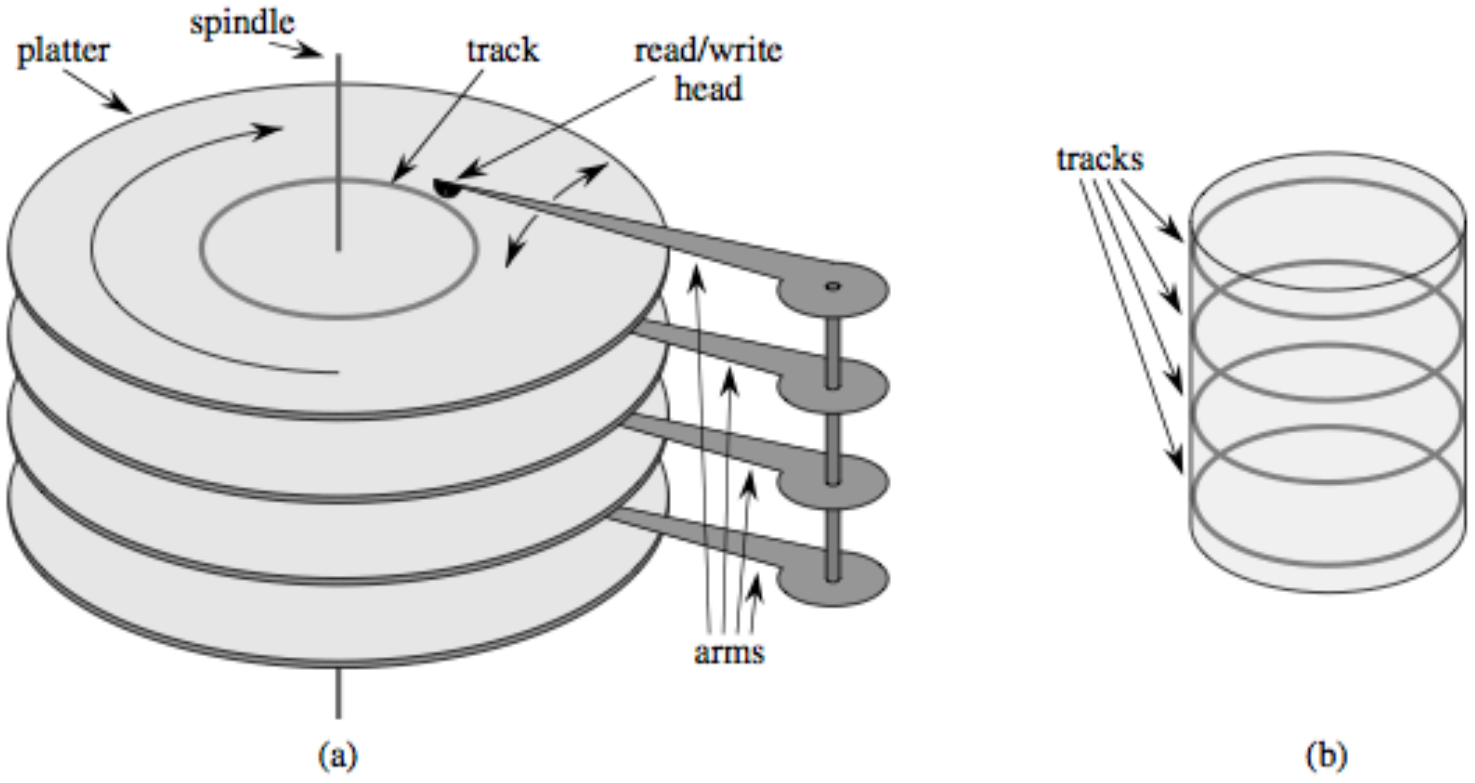
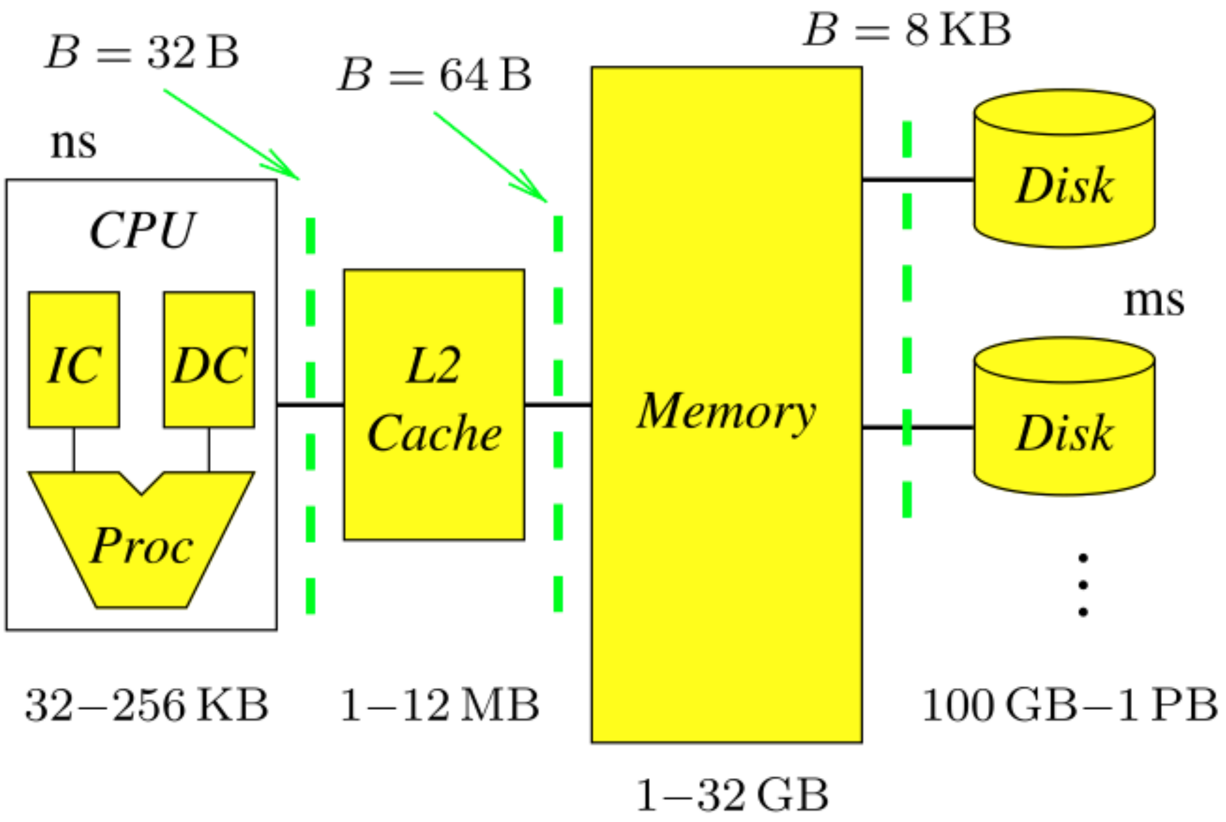


Fig. 2.1 Magnetic disk drive: (a) Data are stored on magnetized platters that rotate at a constant speed. Each platter surface is accessed by an arm that contains a read/write head, and data are stored on the platter in concentric circles called tracks. (b) The arms are physically connected so that they move in unison. The tracks (one per platter) that are addressable when the arms are in a fixed position are collectively referred to as a cylinder.

The Memory Hierarchy

B=Block size



Total Size:

Long-distance communication

- University: Download 10MByte/sec
  Upload 5MByte/sec
- Home Download 3MByte/sec
  Upload 0.2Mbyte/sec
- Transferring 4 Tera-Byte from UCSD to UCLA will take more than 4 days on a dedicated 10MB/sec line.
- It is cheaper and faster to FeDeX the 4TB disk.

Embarrassingly parallel problems

- Count the number of times each word appears in all of the books in the library of congress.
- Find all of the abnormal cells in a pap-smear image.
- Track the socio-economic characteristics of people that view each you-tube video.

Word Counting

- Given a large collections of texts, and a large cluster of computers, count the number of appearances of each word.
- The basic scheme:
  1. Partition the texts and put each part on the disk of a separate computer.
  2. Each computer counts the number of occurrences of each word in it collection.
  3. The word counts from the different computers are combined.

Map-Reduce for Word Count

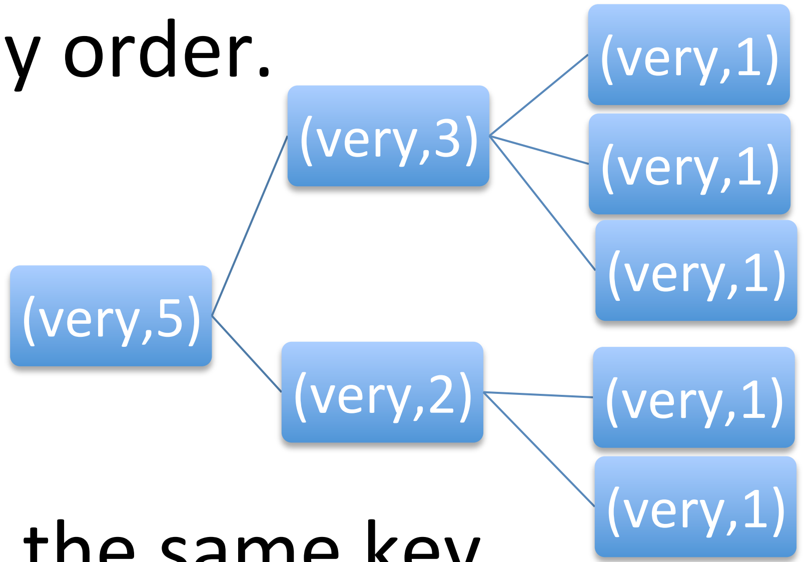
- Map Reduce: operates on (key,value) pairs:
- MAP: maps each k-v input to one or more k-v outputs.
  - (,“very very cool”) -> (very,1),(very,1),(cool,1)
- Reduce: Gets as input pairs with the same key value an produces a new pair with the same key and a combined value:
  - [(very,1),(very,1)] -> (very,2)
- Reductions can occur in any order.

Map-Reduce characteristics

- A map operation is nothing more than the observation that some of the computation can be performed independently on each piece of data.
- Maps require no communication between computers
- Reduction takes the outputs from the map

Distributed reductions

- Reductions can occur in any order.
- Each computer can do it's own reduction before communicating to other computers (combiner).
- Combining all records with the same key is done by sorting.
- But how are the texts distributed to the different computers in the first place?



Hadoop Distributed File System (HDFS)

- Started in Google (GFS)
- Connect the personal work-stations on the desk of each employee to create a huge super-computer.
- Goals:
  1. A huge data repository available to all employees.
  2. A seamless way to use all of the computers for a single computation task.
  3. A fault-tolerant system.

How does HDFS work?

- The data is broken into fixed-size blocks: default is 64MB.
- Each block is **replicated**. Typically 3 times.
- Each block is placed on a randomly chosen computer. A directory structure keeps tabs for the location of each block.

Load balancing with HDFS

1. Suppose we want to operate on a single block. As there are 3 copies of the block, we can choose the least loaded of these computers to perform the operation.
2. Suppose we want to operate on a million blocks and we have 1,000 computers. We can partition the blocks to the computers so that they will all finish at the same time.
3. Random location saves us from the need to perform initial communication to get data to the processing computer.

Fault tolerance of HDFS

- In a cluster of 1,000 commodity computers a few disks crash every day.
- As each block is replicated on 3 randomly chosen disks, the probability that all three copies crash at the same time is very small.
- When a disk crashes , all we need to do is insert a new blank disk. The system will automatically repopulate this disk to re-establish the replication system.
- The **computations** that took place on the computer that crashed are automatically restarted on other computers that hold the same disk block.

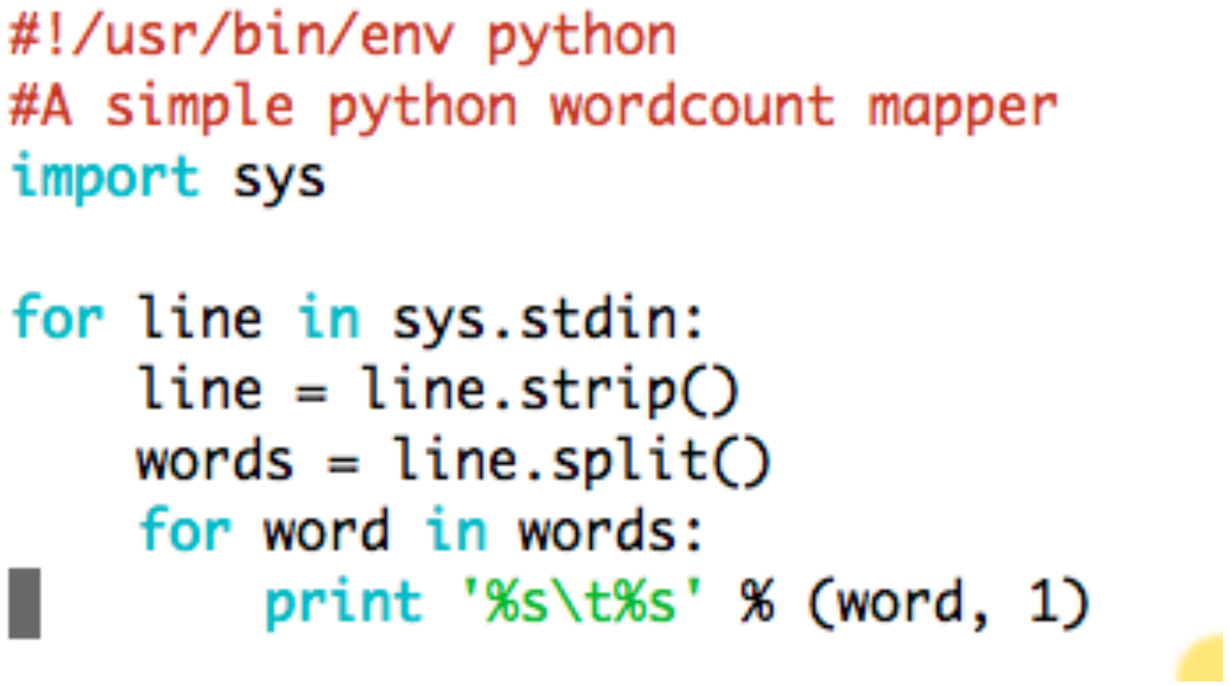
Hadoop

- Hadoop/HDFS is a free open-source implementation of the google distributed file system + map-reduce.
- Hadoop is written in Java
- It is possible to write mappers and reducers in any language by using the **streaming** interface

Hadoop streaming

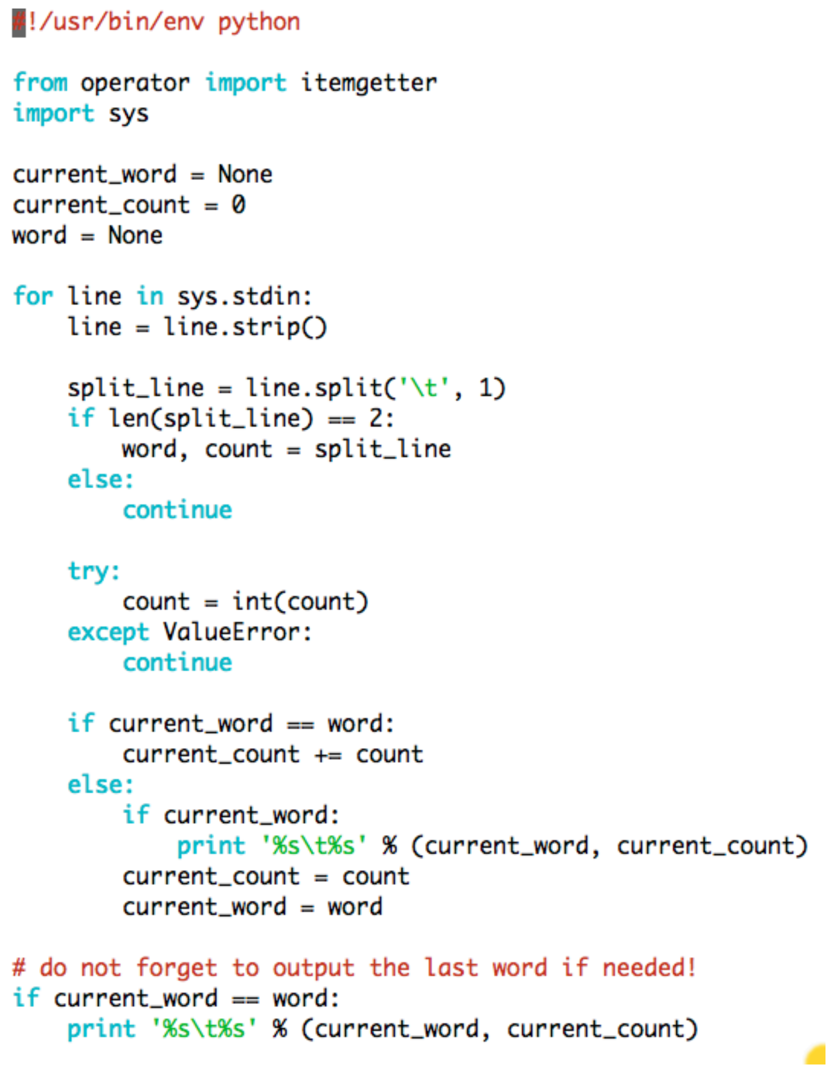
- Can use any language in which one can write a program that reads from stdin and writes to stdout : C/C++,Java, Python, PERL, Ruby, Fortran, …
- Need to write only three files:
  – A Mapper
  – A reducer (some tasks don't require a reducer)
  – An execution script

Word count mapper



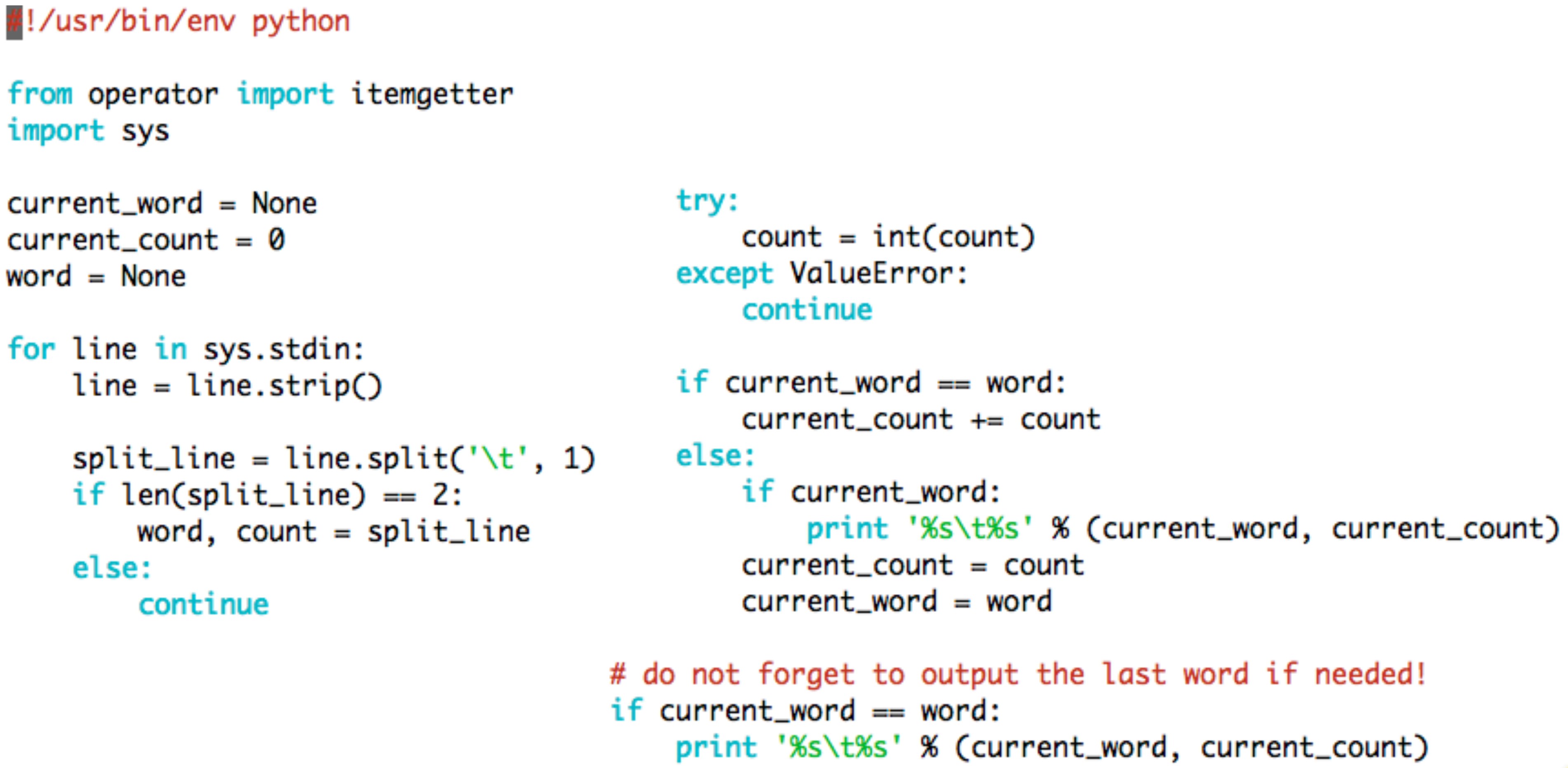
```
#!/usr/bin/env python
#A simple python wordcount mapper
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print '%s\t%s' % (word, 1)
```

Word Count Reducer



```python
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()

    split_line = line.split('\t', 1)
    if len(split_line) == 2:
        word, count = split_line
    else:
        continue

    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Word Count Reducer



```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()

    split_line = line.split('\t', 1)
    if len(split_line) == 2:
        word, count = split_line
    else:
        continue

    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Word Count script

```
#!/bin/bash

source $CSD181/hadoop/hadoop_shared/hadoop_bashrc.sh
shopt -s expand_aliases
# todo:
# Problems generating the output
echo "Usage: $0 [input folder] [hadoop options...]
For example: $0 Reuters -Dmapred.map.tasks=16 -Dmapred.reduce.tasks=1"

if [ "$1" ]  && [ -d $1 ]
then
  source=$1
else
  source=$CSD181/hadoop/sample_data/WordCount/Moby-Dick/
fi

dir_hdfs=/user/$USER/wordcount/Moby-Dick
code_dir=$CSD181/hadoop/hadoop_scripts/python_streaming_wordcount

hdfs -rmr $dir_hdfs

# create directory called /user/$USER/wordcount/$2/input and copy into it the files from local directory $1
hdfs -copyFromLocal $source $dir_hdfs/input # copy input into input directory

had jar /opt/hadoop/contrib/streaming/hadoop-*streaming*.jar $@ -file $code_dir/map.py -mapper $code_dir/map.py
 -file $code_dir/reduce.py -reducer $code_dir/reduce.py -input $dir_hdfs/input/* -output $dir_hdfs/output
```

Finding drug-drug interactions

- N=10,000 drugs
- A chemical/medical profile for each drug.
- Find pairs of drugs that are likely to have adverse interactions if taken at the same time.
- Input consists of (key, value) pairs where key identifies the drug and value contains drug profile.
- Assume keys can be ordered (numbers)

Straight-forward solution

- Mapper stores all keys in memory:
  - $j_1, j_2, \ldots, j_N$
- For each input (k_i,v_i) it generates N-1 KV pairs where Value=v_i and Key=(j_j,k_i) if $j_j<k_i$ and Key=(k_i,j_j) if $k_i<j_j$
- There are two KV pairs for each K. One with the profiles of each of the two drugs.
- The reducer takes each pair and computes the compatibility.
- The main problem with this solution: N-1 records are generated for each drug. Most of these will need to be communicated between computers in the cluster.

A better solution

- Divide the drugs into M groups of size N/M each.
- The mapper generates for each drug group M key-value pairs where the key consists of two group indexes (in increasing order) and the value consists of N/M profiles.
- This solution reduces the communication throughput by a factor of N/M. In addition, the communicated files are larger and therefor can be communicated more efficiently.

So why is Hadoop a big deal?

- Map/Reduce is less general than other distributed computation system such as MPI.
- Java is less efficient than C++
- The Map-Reduce paradigm is very easy to understand.
- The user does not need to concern herself with the details of communication and synchronization. Just produce the correct key and limit the duplication of the data.

Hadoop provides

- Hadoop provides the communication: it gets all records with the same key to a single reducer.
- Done through distributed sorting
- Shuffle and sort: the heart of hadoop map-reduce, gets more sophisticated and faster with each release.

Logical data flow

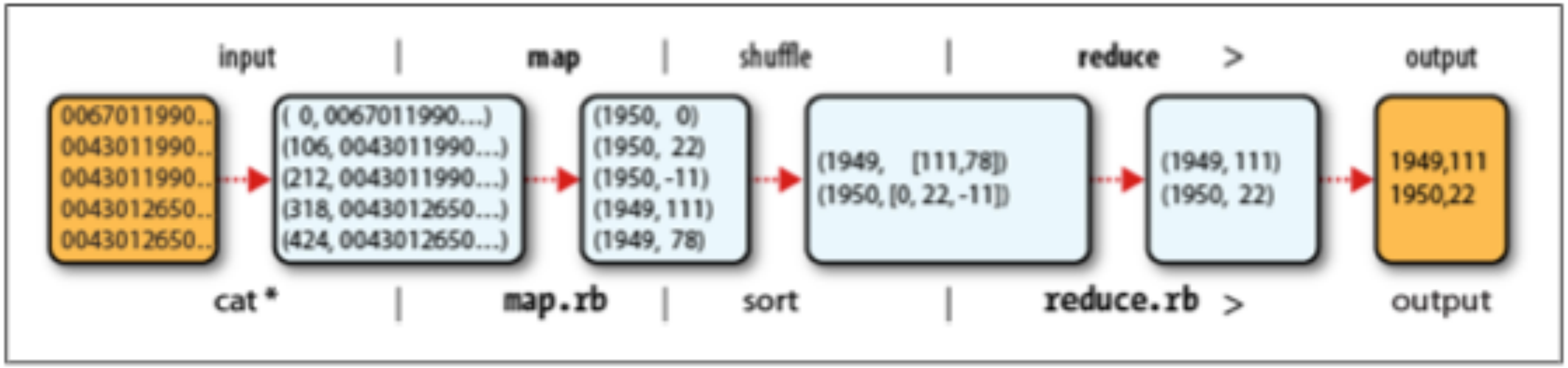
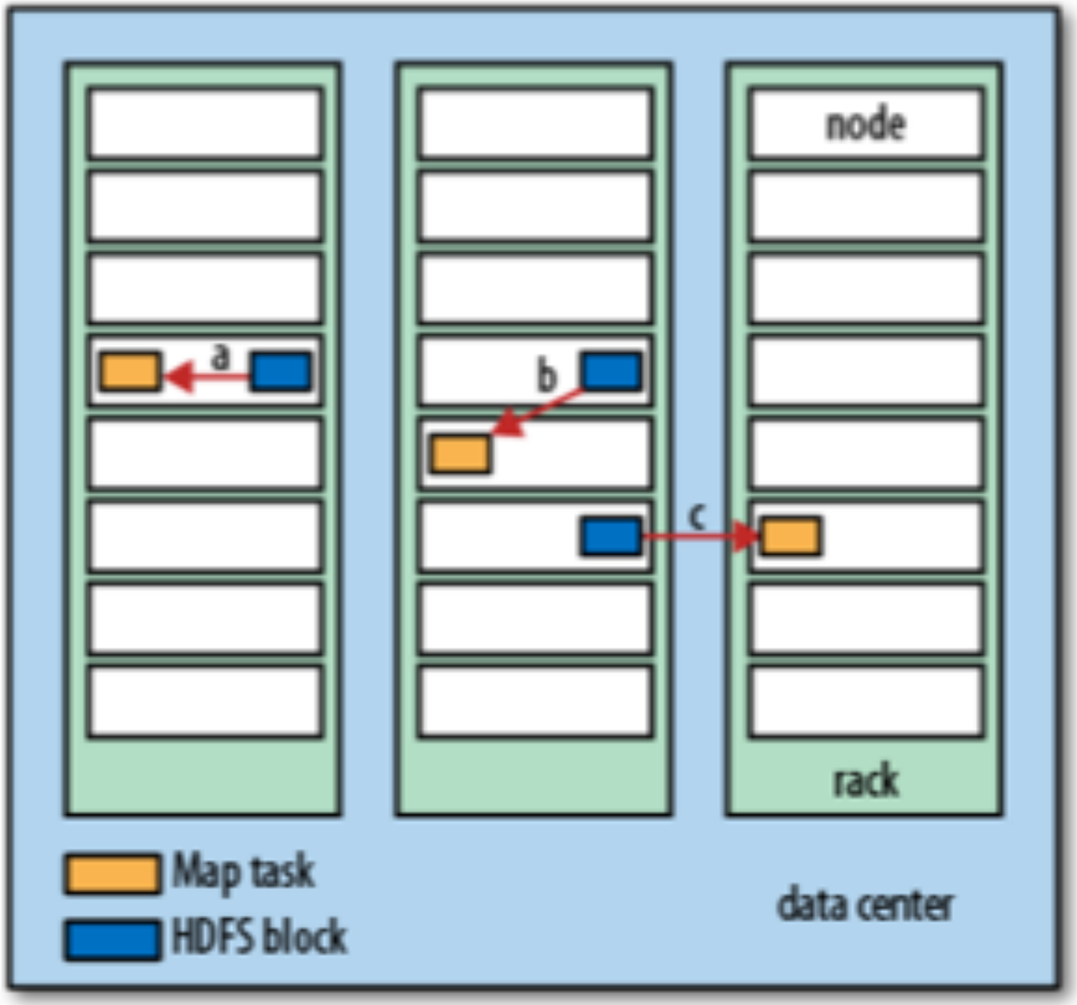
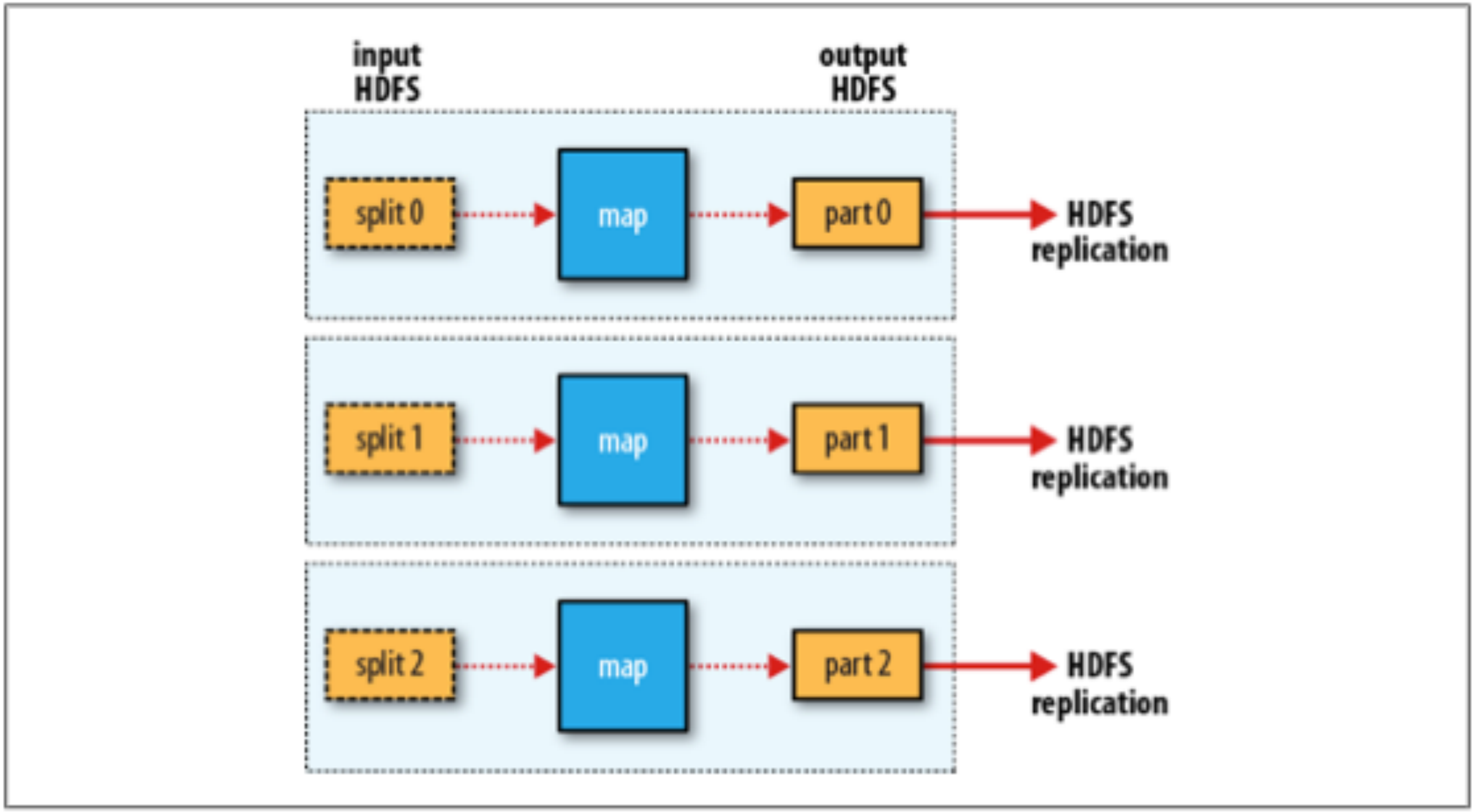


Figure 2-1. MapReduce logical data flow

mapper locality



Task with no reducers



Task with a single reducer

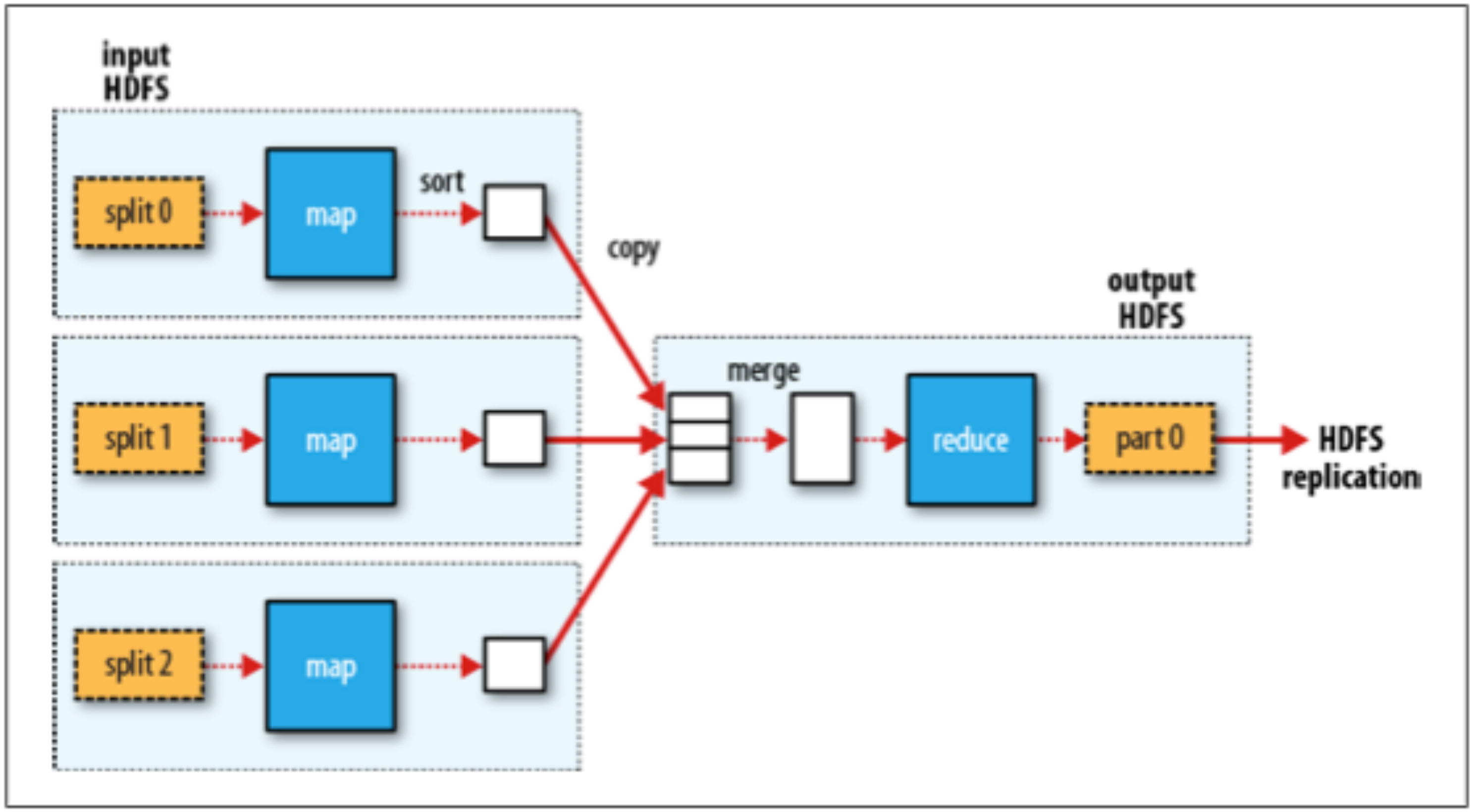


Figure 2-3. MapReduce data flow with a single reduce task

Task with two reducers

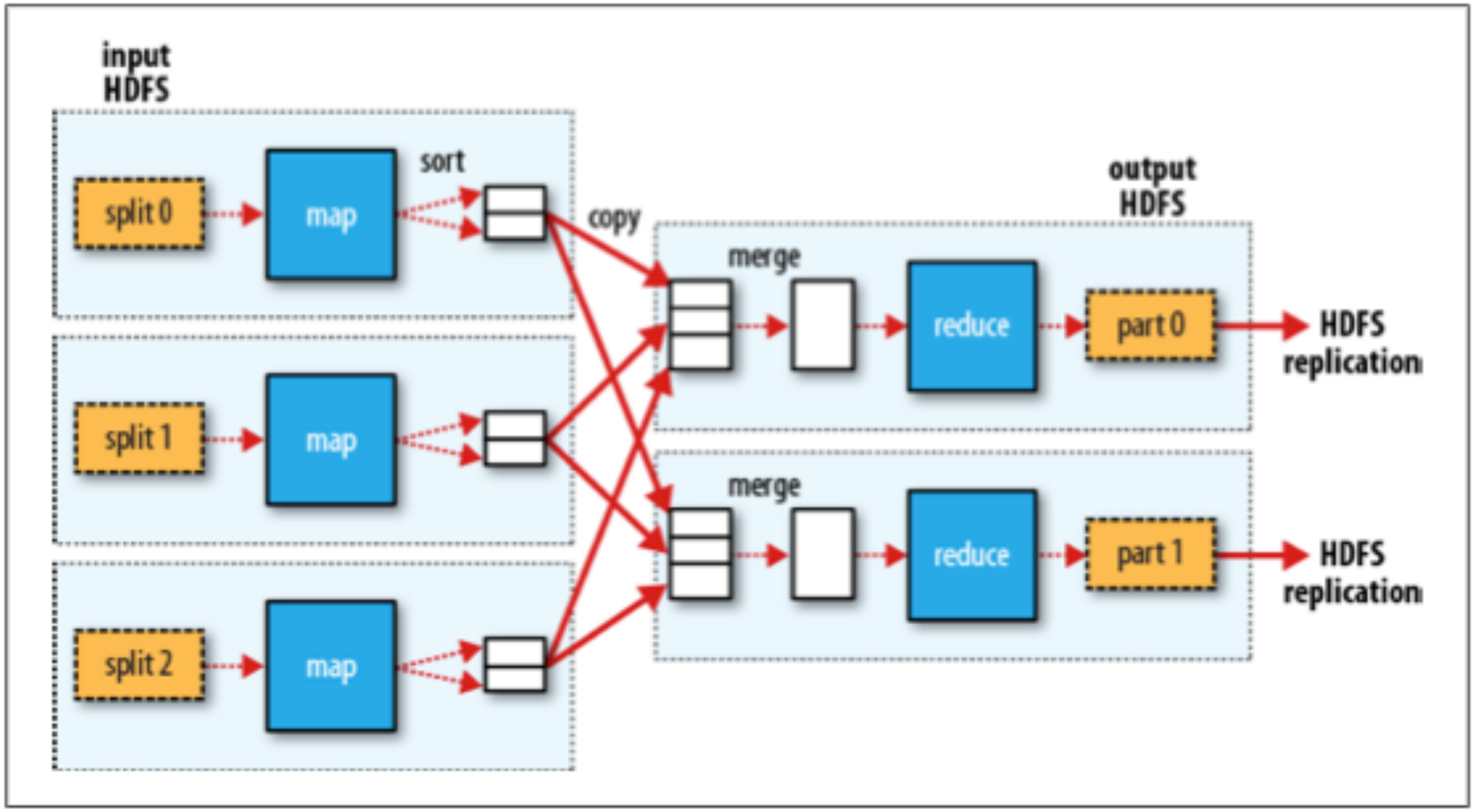


Figure 2-4. MapReduce data flow with multiple reduce tasks

Shuffle and Sort

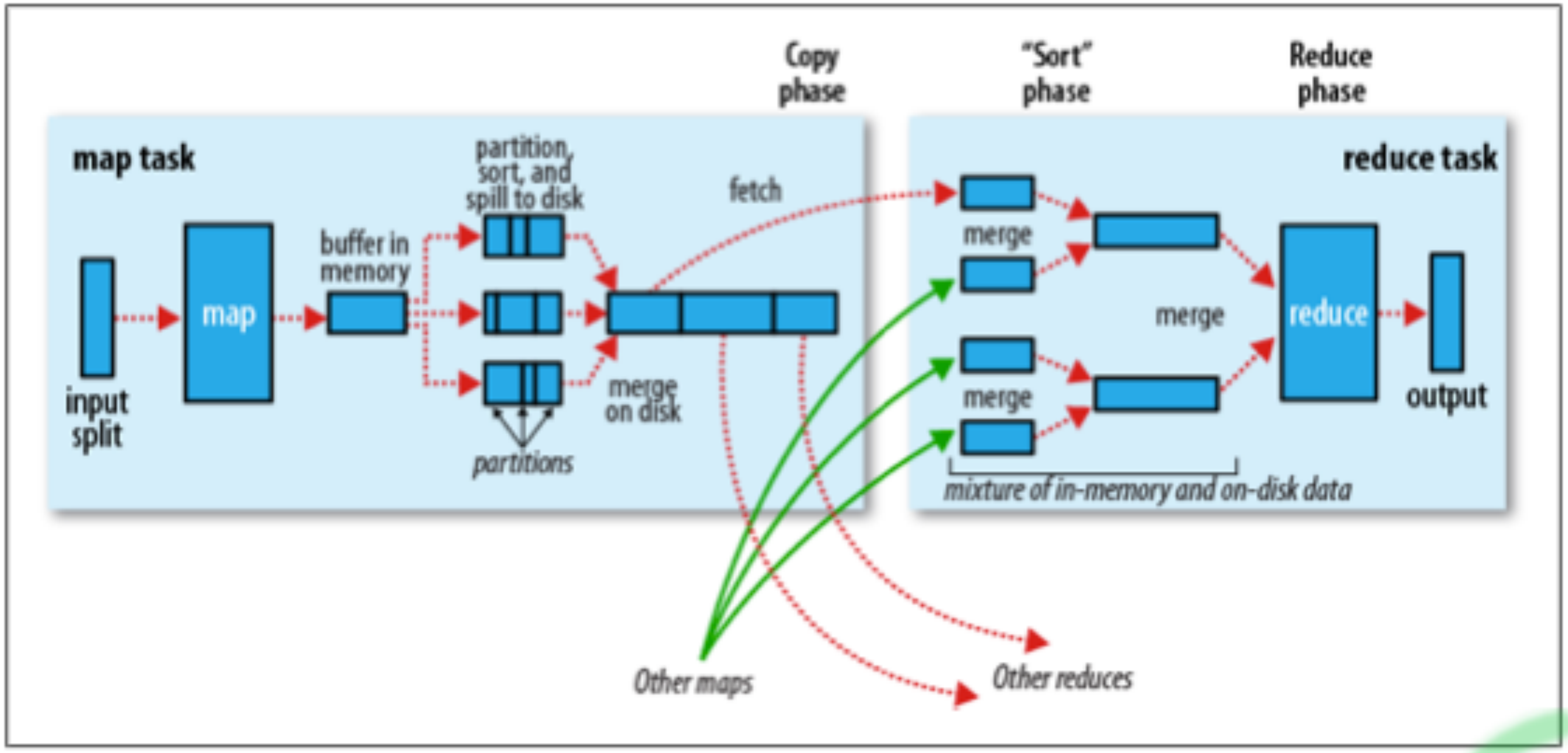


Figure 6-6. Shuffle and sort in MapReduce

The amnesia of Hadoop map reduce

- Sometimes we need to iterate over Map/Reduce steps and the map-reduce requires touching each record.
- In such cases Hadoop map-reduce requires loading the data from disk to memory at each step.
- This disk-to-memory transfer (de-serialization) becomes the major bottleneck.
- In general, Hadoop is a good method for batch jobs, not very good for interactive work.

K-Means

- A very popular clustering algorithm.
- Input:
  - N points in R^d: $x_1, \ldots, x_N$
  - Desired number of clusters K
- Desired output:
  - K centers in R^d: $c_1, \ldots, c_N$
  - Such that the average distance btwn x_i and the closest c_j is minimal

The K-Means algrorithm

- Randomly choose K points to serve as the initial centers.
- Repeat until convergence:
    1. Assign each point to the closest center point.
    2. Replace each center by the mean of the points assigned to it.

One iteration of K-Means using Map-Reduce

- Partition the N points into M parts, each corresponds to a mapper.
- M Mappers:
  - Input: current K centers. Requires reading N/M points to main memory
  - Output: the sum and the number of the points that are assigned to each center.
- K reducers:
  - Input: partial sums of the points assigned to a center.
  - Output: compute the average of all of the points associated with each center.

The problem and it's solution

- In Hadoop, mappers and reducers terminate after job is done.
- In an iterative algorithm such as K-means this means that data is repeatedly read from disk into memory.
- Spark http://spark-project.org/ a system for performing map-reduce computation with data cached in main memory.
- Fault tolerance is achieved through checkpointing = writing out the state of the memory from time to time.
- Using Spark one can perform real-time analysis on 100s of GB and get results within seconds.

Summary

- Big data = computations where the constraining limitation is the communication across the memory hierarchy.
- Hadoop = A way to analyze a lot of data on the cheap.
- Map-Reduce: a very simple, yet powerful, pattern for distributed computation.
- Spark – memory-resident distributed computation.
- This is only the tip of the iceberg.

Thank you!