CSE 544 Principles of Database Management Systems

Alvin Cheung

Fall 2015

Lecture 10 – Parallel Programming Models: Map Reduce and Spark

Announcements

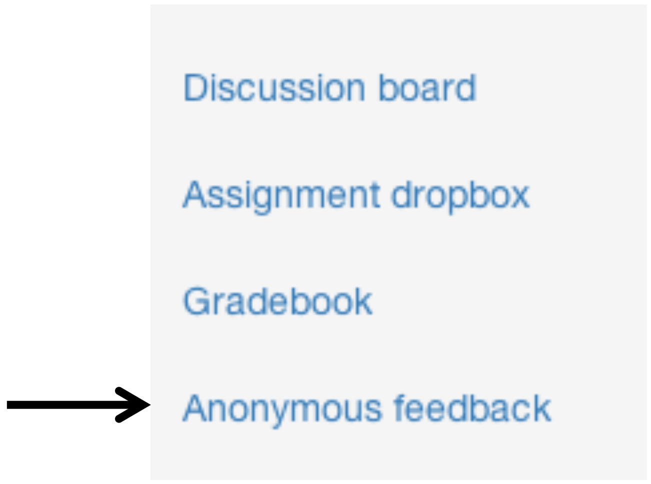
- HW2 due this Thursday

- AWS accounts
  - Any success?

- Feel free to drop by OH if you have project questions
  - We will do another round of meetings after project milestones

Mid-Term Evals

- Aspects that people find useful:
  - Pace of the class
  - Examples during lectures
  - Paper readings
  - Mix of theory and practical concepts
- Things that can be improved:
  - Paper readings and late policies
  - HW feedback (?)
  - Website
  - HW lengths
  - Anything else?



Programming Models for Analytics

- How real-world users perform data analytics
  - SQL queries (last 2 lectures)
  - Map Reduce programs (today)
  - Spark programs (today)

  - (there are many others as well: Pig, Hive, Pandas, etc)

References

- **MapReduce: A major step backwards.** DeWitt and Stonebraker, The Database Column, January 2008.

- **Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing.** Zaharia et al, NSDI 2012.

Map Reduce

- Google: [Dean 2004]
- Open source implementation: Hadoop

- MapReduce = high-level programming model and implementation for large-scale parallel data processing

- Core idea:
  - Explicit parallelism

Map Reduce Motivation

- Not designed to be a DBMS
- Designed to simplify task of writing parallel programs
  - A simple programming model that applies to many large-scale computing problems
- Hides messy details in MapReduce runtime library:
  - Automatic parallelization
  - Load balancing
  - Network and disk transfer optimizations
  - Handling of machine failures
  - Robustness
  - **Improvements to core library benefit all users of library!**

content in part from: Jeff Dean

Data Processing at Massive Scale

- Want to process petabytes of data and more
- Massive parallelism:
  - 100s, or 1000s, or 10000s servers (think data center)
  - Many hours
- Failure:
  - If medium-time-between-failure is 1 year
  - Then 10000 servers have one failure / hour

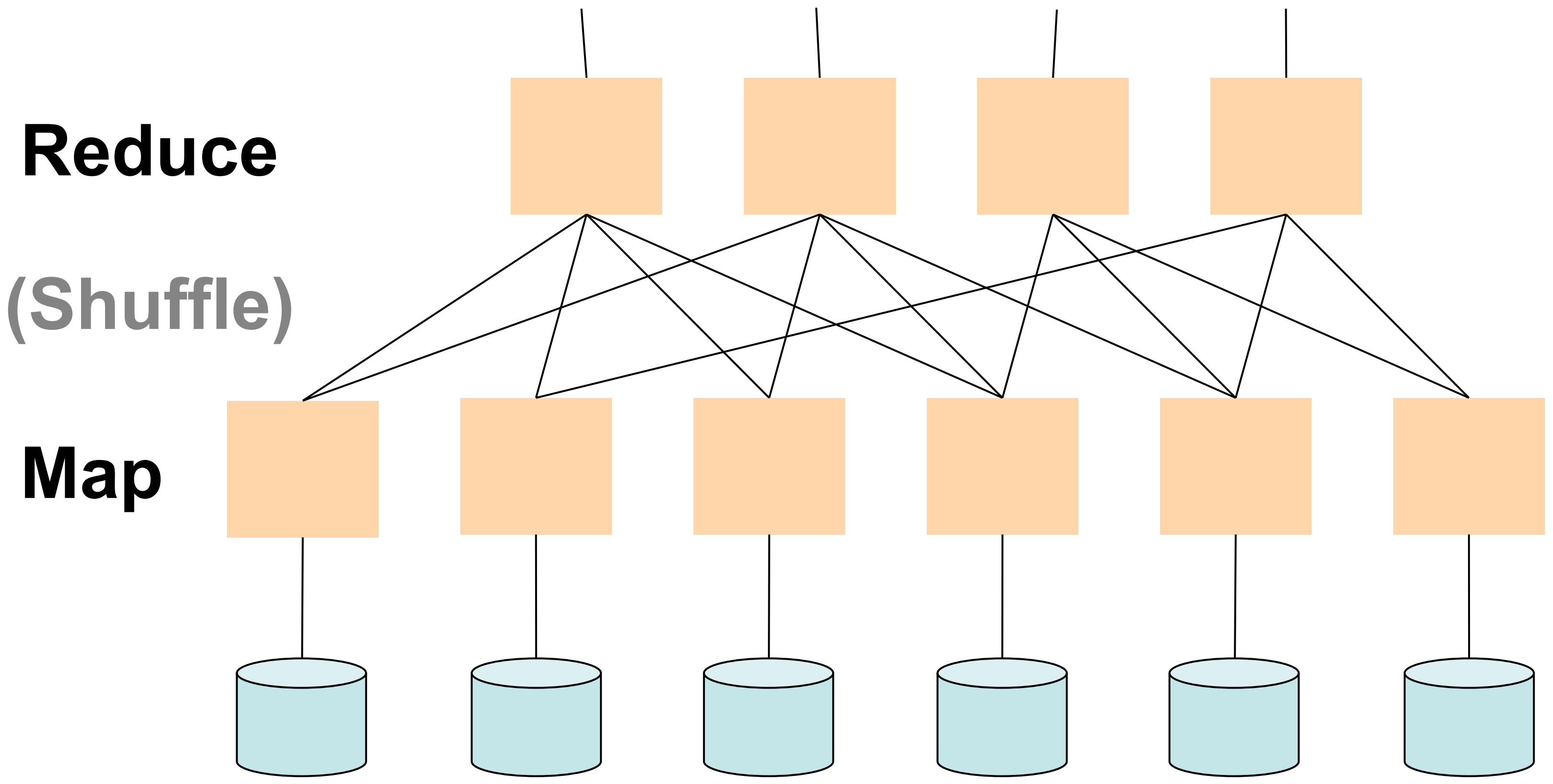
Data Storage: GFS/HDFS

- MapReduce job input is a file

- Common implementation is to store files in a highly scalable file system such as GFS/HDFS
  - GFS: Google File System (proprietary)
  - HDFS: Hadoop File System (open source)

  - Each data file is split into M blocks (64MB or more)
  - Blocks are stored on random machines & replicated
  - Files are append only

Running your favorite parallel algorithm…



Does this look familiar?

Typical Problems Solved by MR

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, transform
- Write the results

Outline stays the same,
map and reduce change to
fit the problem

slide source: Jeff Dean

Data Model

Files !

A file = a bag of **`(key, value)`** pairs

A MapReduce program:

- Input: a bag of **`(inputkey, value)`** pairs
- Output: a bag of **`(outputkey, value)`** pairs

Step 1: the MAP Phase

User provides the MAP-function:

- Input: **`(input key, value)`**
- Output: **bag** of **`(intermediate key, value)`**

System applies map function in parallel to all **`(input key, value)`** pairs in the input file

Step 2: the REDUCE Phase

User provides the REDUCE function:

- Input: **`(intermediate key, bag of values)`**
- Output (original MR paper): bag of output **`(values)`**
- Output (Hadoop): bag of **`(output key, values)`**

System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

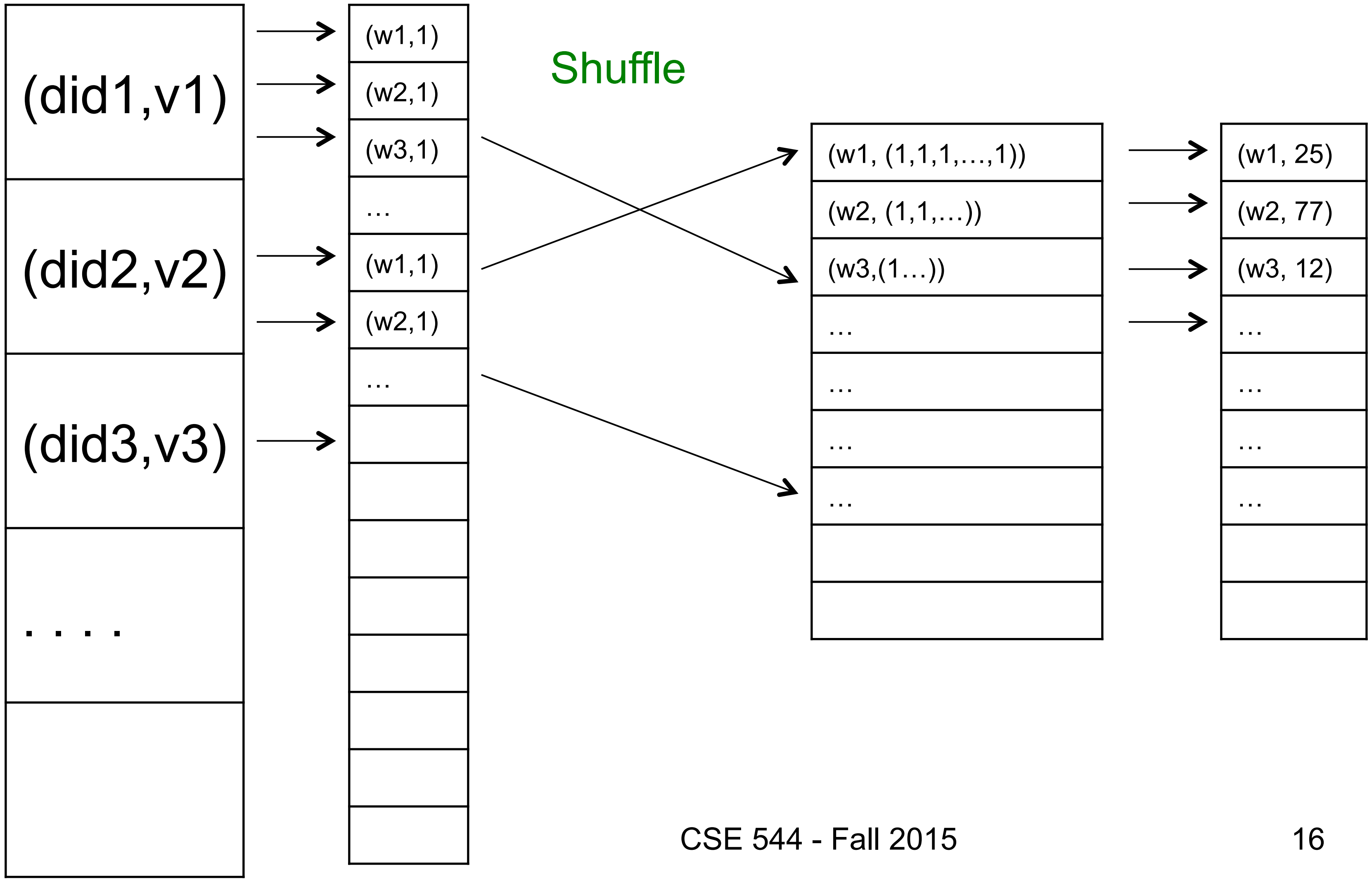
Famous (Infamous?) Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
            EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
            result += ParseInt(v);
Emit(AsString(result));
```

MAP REDUCE


(did1,v1)
(did2,v2)
(did3,v3)
. . . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
Shuffle
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…
…

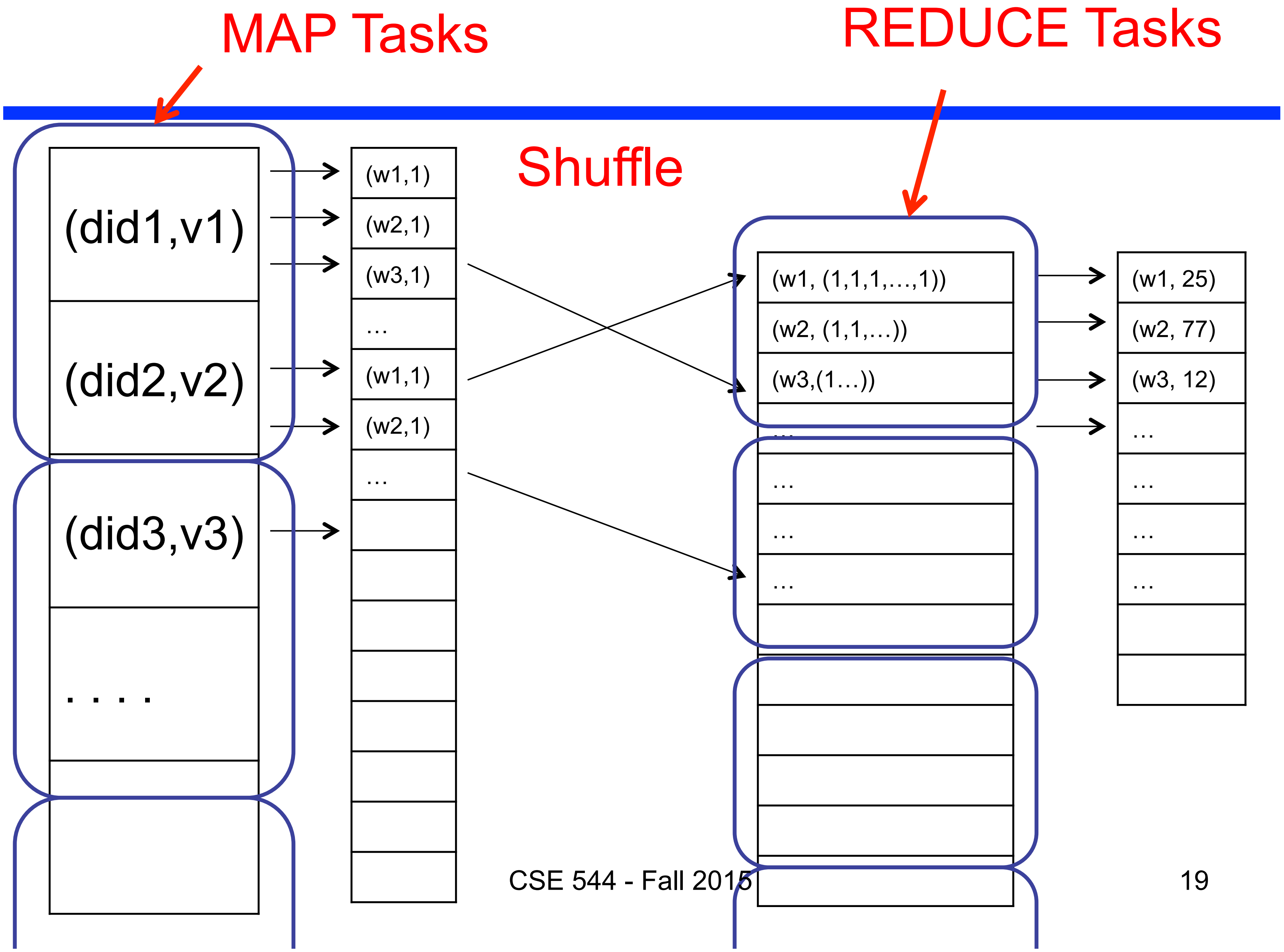
Jobs v.s. Tasks

- A MapReduce Job
  - One single “query,” e.g. count the words in all docs
  - More complex queries may consist of multiple jobs
- A Map Task, or a Reduce Task
  - A group of instantiations of the map-, or reduce-function, which are scheduled on a single worker

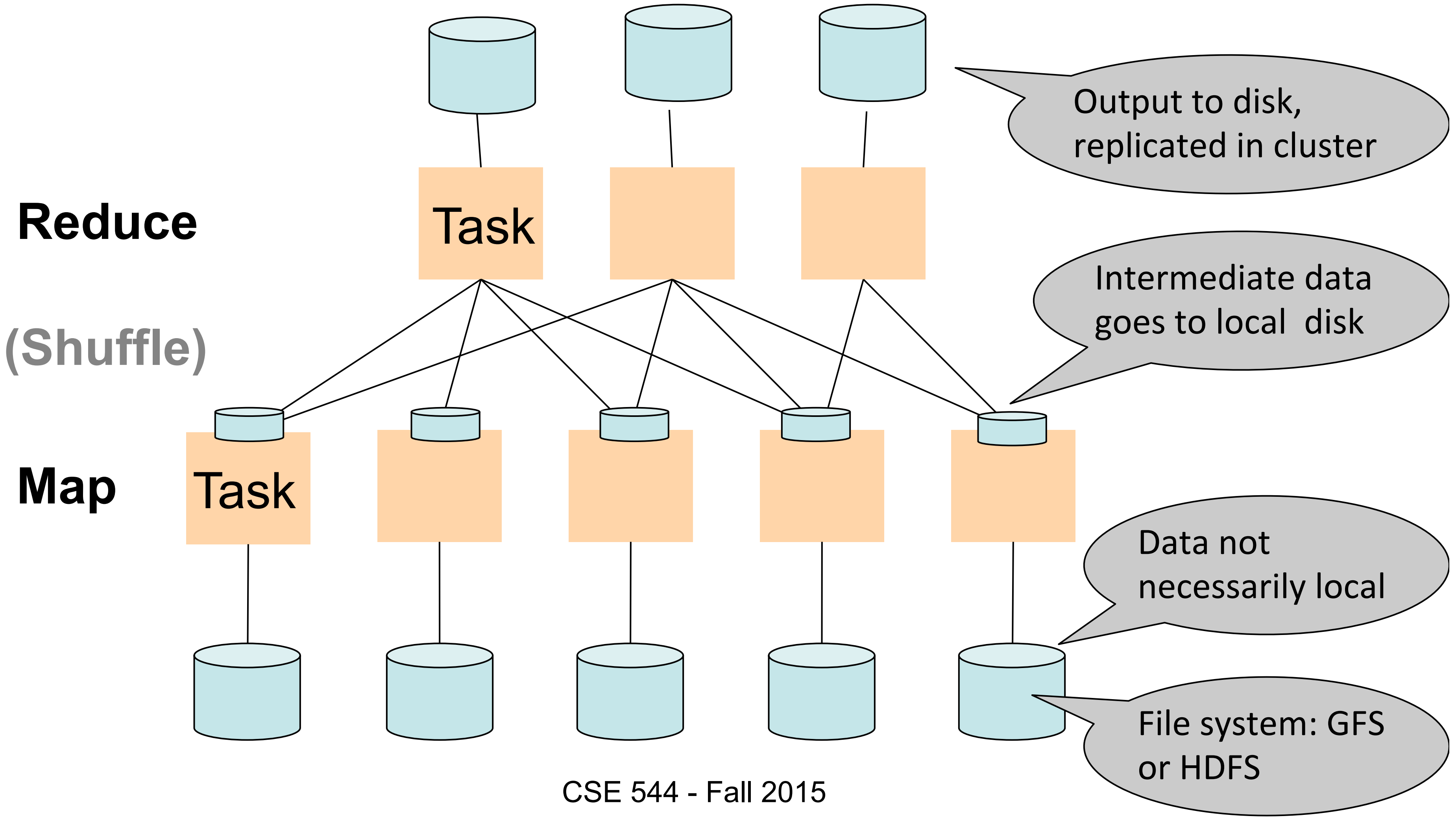
Workers

- A worker is a process that executes one task at a time
- Typically there is one worker per processor, hence 4 or 8 per node

- Often talk about “slots”
  - E.g., Each server has 2 map slots and 2 reduce slots


MAP Tasks
REDUCE Tasks
Shuffle
(did1,v1)
(did2,v2)
(did3,v3)
. . . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…
…

Parallel MapReduce Details

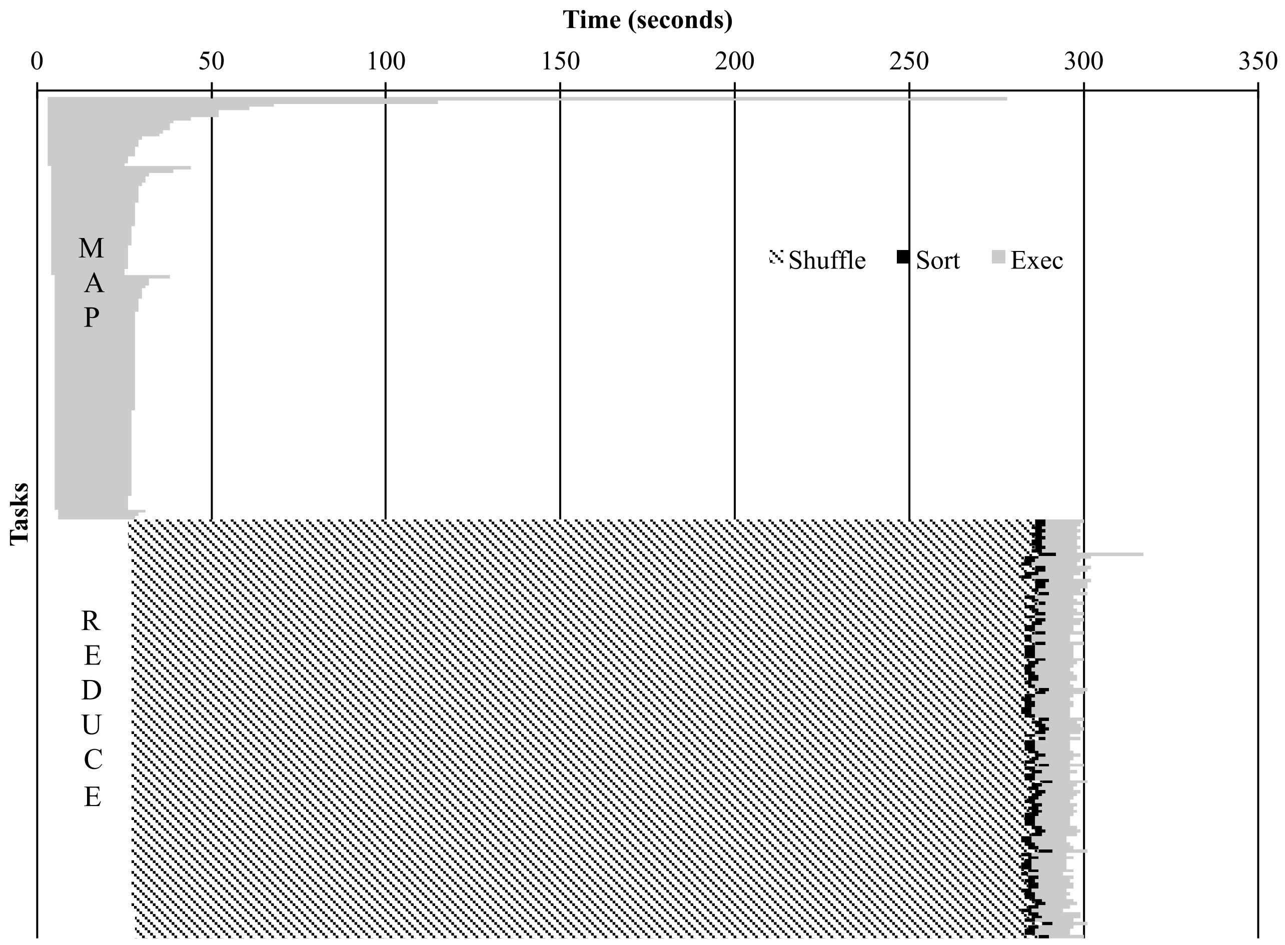

Output to disk, replicated in cluster
Reduce
Task
Intermediate data goes to local disk
(Shuffle)
Map
Task
Data not necessarily local
File system: GFS or HDFS

MapReduce Implementation

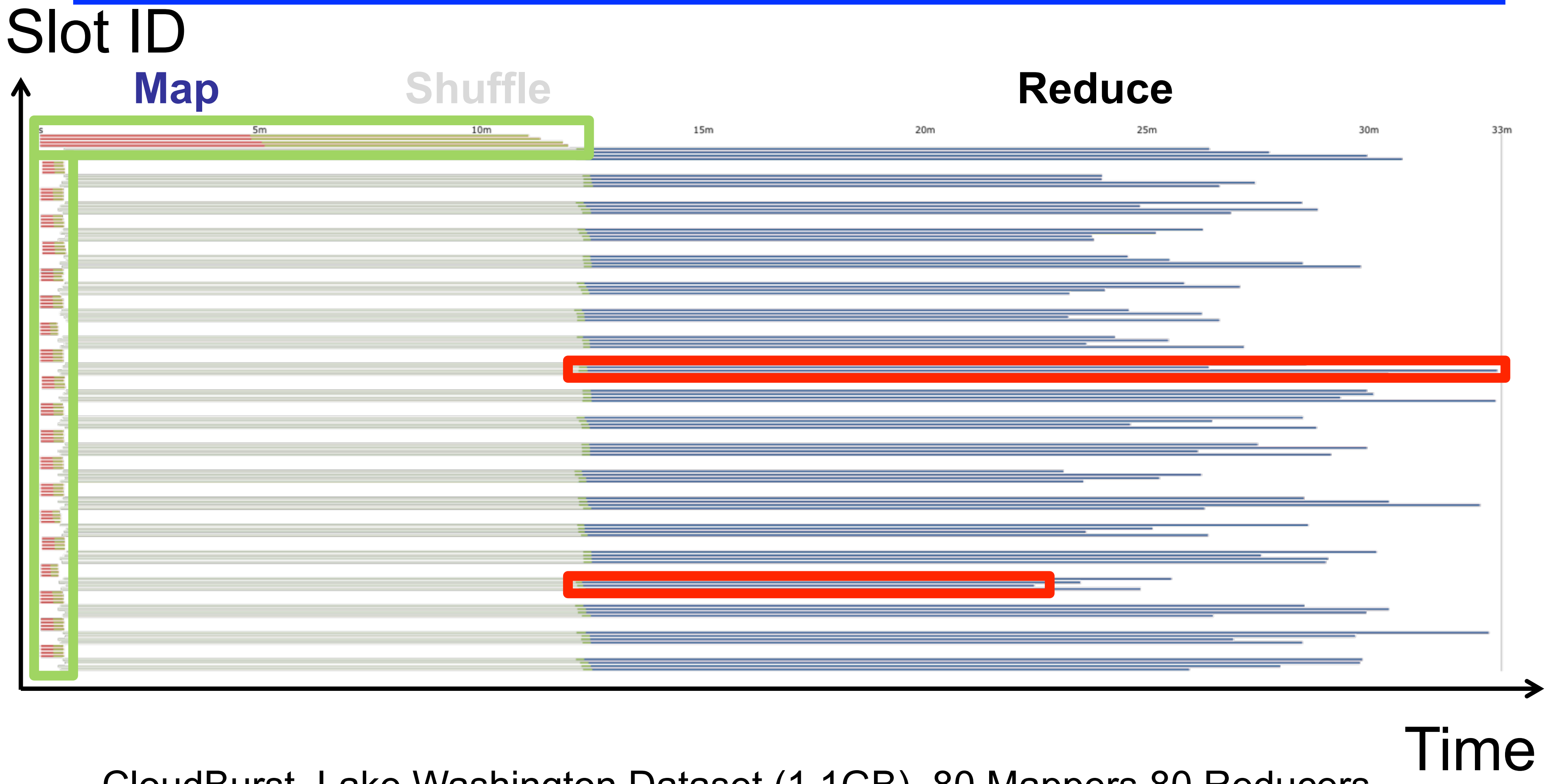
- There is one master node
- Input file gets partitioned further into *M' splits*
  - Each split is a contiguous piece of the input file
- Master assigns *workers* (=servers) to the *M' map tasks*, keeps track of their progress
- Workers write their output to local disk
- Output of each map task is partitioned into *R regions*
- Master assigns workers to the *R reduce tasks*
- Reduce workers read regions from the map workers' local disks

Example Map Reduce Execution

PageRank Application

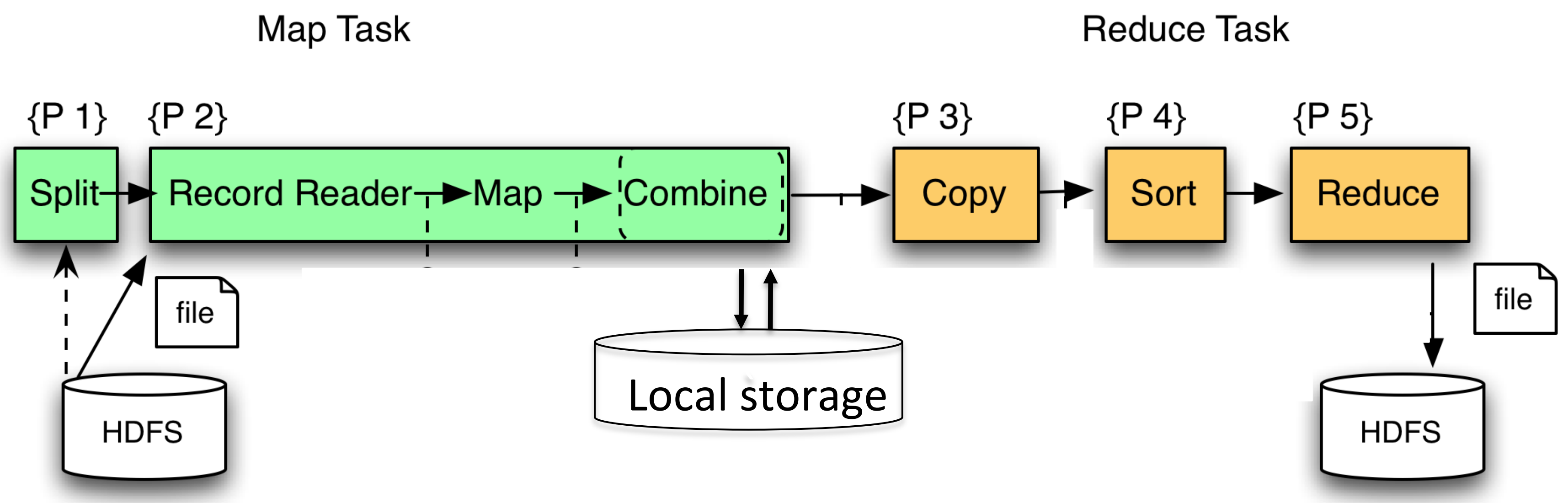


Example: CloudBurst



CloudBurst. Lake Washington Dataset (1.1GB). 80 Mappers 80 Reducers.

MapReduce Phases

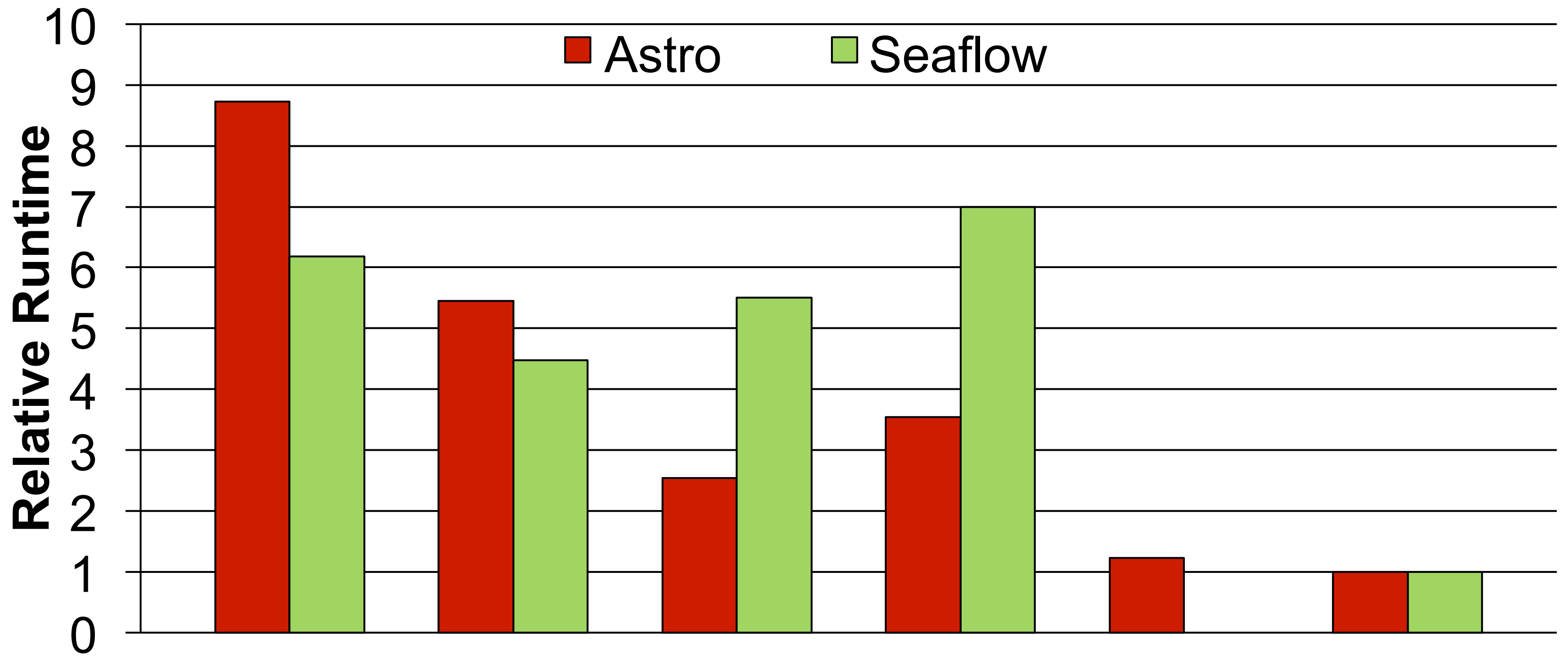


Interesting Implementation Details

- Worker failure:
  - Master pings workers periodically
  - If down then reassigns its task to *another* worker
  - (≠ a parallel DBMS restarts whole query)

- How many map and reduce tasks:
  - Larger is better for load balancing
  - But more tasks also add overheads
  - (≠ parallel DBMS spreads ops across all nodes)

MapReduce Granularity Illustration



Coarse	Fine	Finer	Finest	Manual	SkewReduce	
14.1	8.8	4.1	5.7	2.0	1.6	Hours
87.2	**63.1**	**77.7**	**98.7**	**-**	**14.1**	Minutes

Interesting Implementation Details

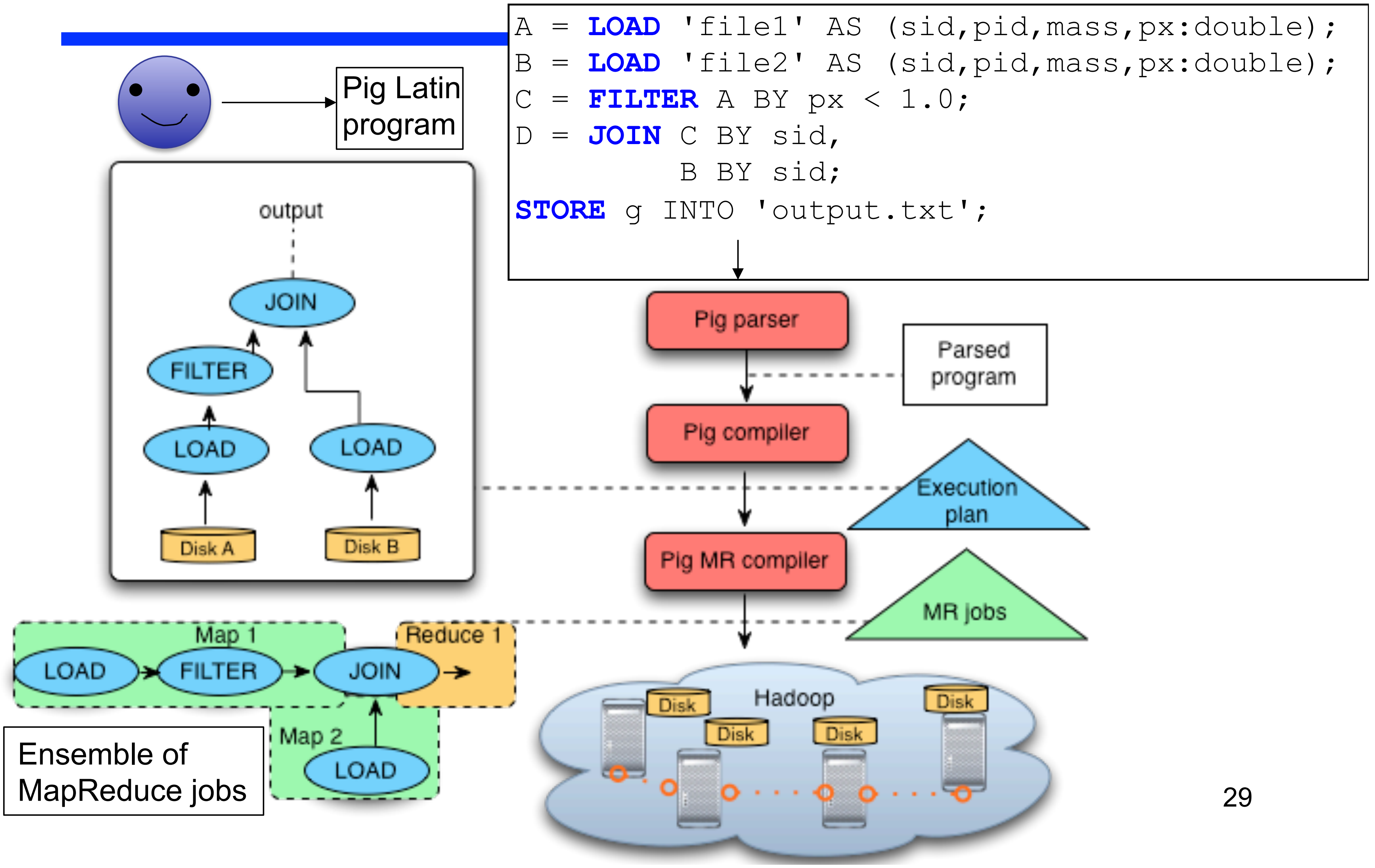
Backup tasks:

- *Straggler* = a machine that takes unusually long time to complete one of the last tasks. e.g.:
  - Bad disk forces frequent correctable errors (30MB/s → 1MB/s)
  - The cluster scheduler has scheduled other tasks on that machine
- Stragglers are a main reason for slowdown
- Solution*: pre-emptive backup execution of the last few remaining in-progress tasks*

Declarative Languages on MR

- PIG Latin (Yahoo!)
  - New language, like Relational Algebra
  - Open source
- HiveQL (Facebook)
  - SQL-like language
  - Open source
- SQL / Tenzing (Google)
  - SQL on MR
  - Proprietary

Example: Pig system



```
A = LOAD 'file1' AS (sid,pid,mass,px:double);
B = LOAD 'file2' AS (sid,pid,mass,px:double);
C = FILTER A BY px < 1.0;
D = JOIN C BY sid,
         B BY sid;
STORE g INTO 'output.txt';
```

MapReduce State

- Lots of extensions to address limitations
  - Capabilities to write DAGs of MapReduce jobs
  - Declarative languages
  - Ability to read from structured storage (e.g., indexes)
  - Etc.

- Most companies use both types of engines
- Increased integration of both engines

Parallel DBMS vs MapReduce

- Parallel DBMS
  - Relational data model and schema
  - Declarative query language: SQL
  - Many pre-defined operators: relational algebra
  - Can easily combine operators into complex queries
  - Query optimization, indexing, and physical tuning
  - Streams data from one operator to the next without blocking
  - Can do more than just run queries: Data management
    - Updates and transactions, constraints, security, etc.

Parallel DBMS vs MapReduce

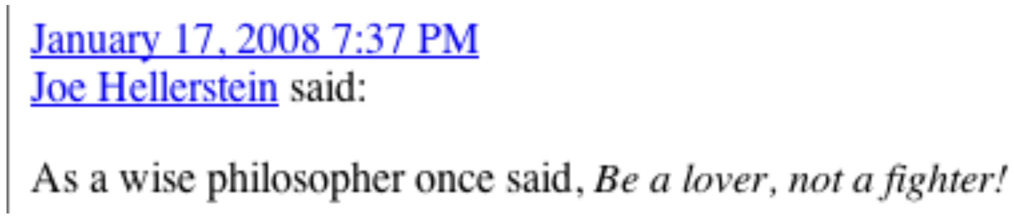
- MapReduce
  - Data model is a file with key-value pairs!
  - No need to “load data” before processing it
  - Easy to write user-defined operators
  - Can easily add nodes to the cluster (no need to even restart)
  - Uses less memory since processes one key-group at a time
  - Intra-query fault-tolerance thanks to results on disk
  - Intermediate results on disk also facilitate scheduling
  - Handles adverse conditions: e.g., stragglers
  - Arguably more scalable… but also needs more nodes!

Parallel DBMS vs MapReduce

- From DeWitt and Stonebraker article:
  - Lack of schema
    Lecture 2
  - No physical tuning
    - Indexes
    - Access methods
    Lectures 6-7
  - No novelty
    Lecture 1
    - map fn list → calls fn on each element in list, and returns a new list
    - fold fn list → passes each element in list to fn, fn computes an "aggregate" value
    - AKA group by and aggregate
  - Missing features as compared to DBMS
    - Updates and deletes
    - ETL tools
    Lecture 8

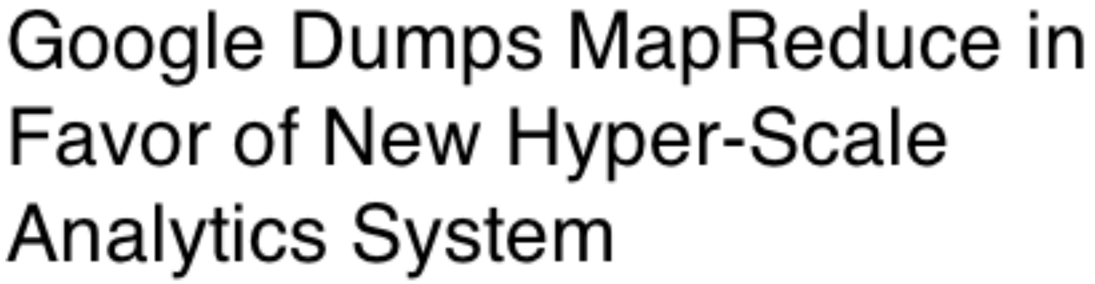
Parallel DBMS vs MapReduce

- Many technical similarities between the two systems
- At the end of the day, it's all about the users
  - They are the ones who need to deal with these tools



January 17, 2008 7:37 PM
Joe Hellerstein said:

As a wise philosopher once said, *Be a lover, not a fighter!*



Google Dumps MapReduce in Favor of New Hyper-Scale Analytics System

Spark

- [Zaharia et al, NSDI 2012]
- Open source implementation on top of Hadoop

- Spark = high-level programming model and implementation for large-scale parallel data processing

- Core idea:
  - Resilient Distributed Datasets (RDDs) as the basic data model

Resilient Distributed Datasets

- Primary abstraction in Spark
  - Immutable once constructed
  - Can be used to construct more RDDs
  - Each RDD traces lineage information of how it was computed
  - Iterate each element in RDD to perform computation
    - Compare this with Map Reduce

Creating RDDs

- Load from files (from local file system, HDFS [Hadoop File System], Amazon S3, etc)
- Generate from in-memory data structures (e.g., lists)
- Compute from an existing RDD

Examples

```
>>> rdd1 = sc.textFile("data.txt")

>>> list = [1, 2, 3, 4, 5]
>>> rdd2 = sc.parallelize(list, 2)
```

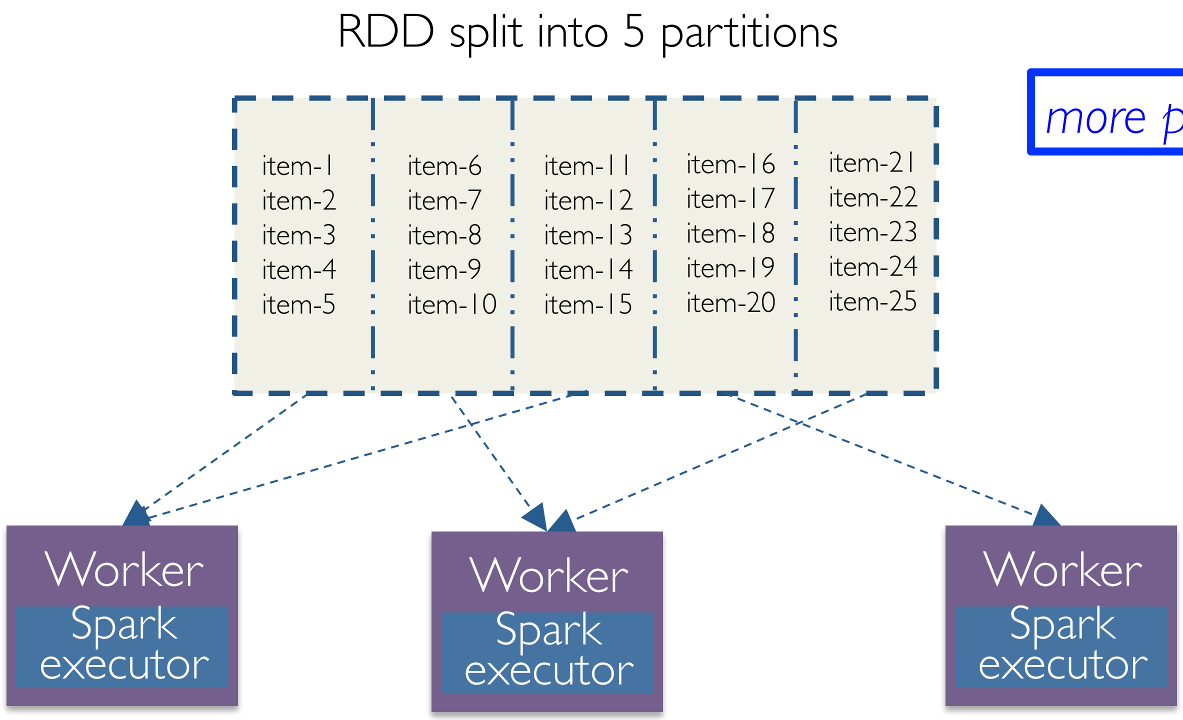
Spark Context object

Divide RDD into 2 *partitions*

Partitions

- Spark's unit of parallelism
  - An RDD divided into N partitions means that it can be *potentially* be operated in parallel by N different workers
  - Default value if unspecified (based on data size)

more partitions = more parallelism



Computing on RDDs

- Spark provides transformations on RDDs
  - Iterates over each element in RDD
- Examples:
  - rdd.map(fn)
    returns a new RDD by passing each element through fn
  - rdd.filter(fn)
    returns a new RDD by retaining those that passes fn
  - rdd.distinct()
    returns a new RDD with only distinct elements from source

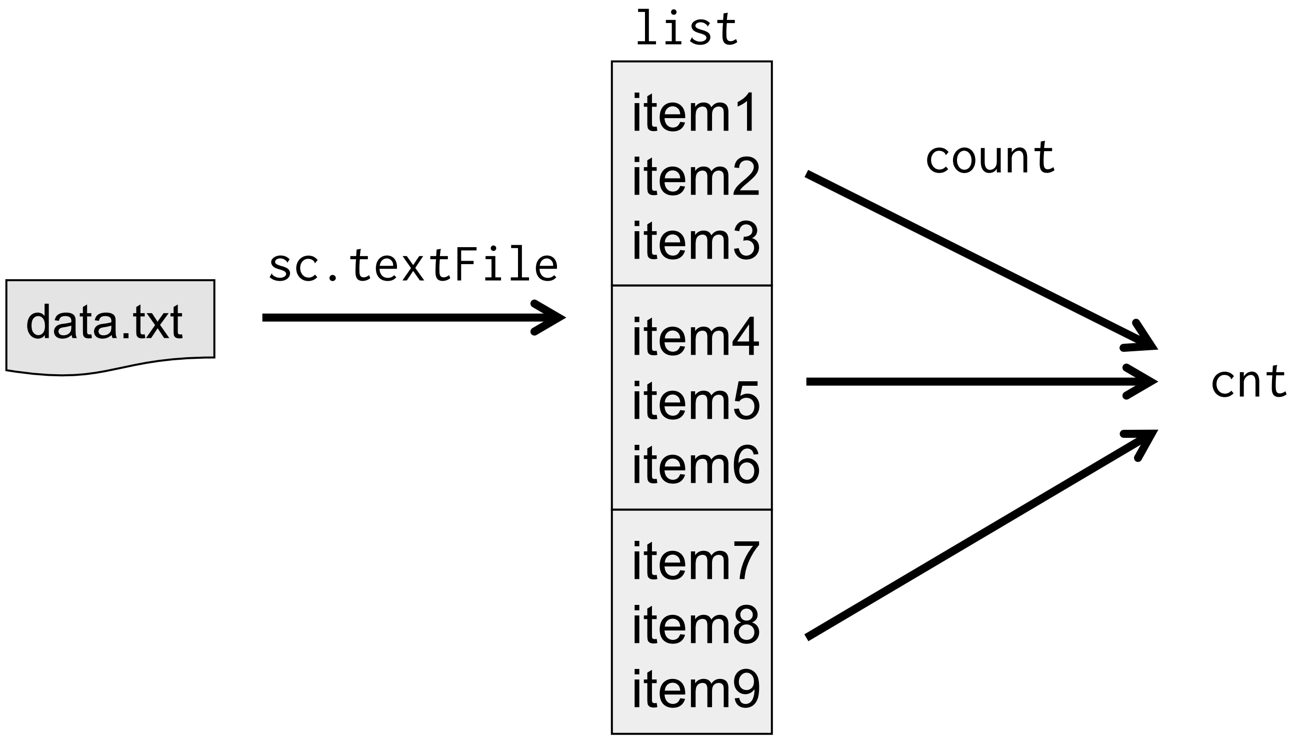
Computing on RDDs

- Spark provides actions to get values out of RDDs
  - Each one performs aggregations on a RDD

- Examples:
  - rdd.reduce(fn)
    computes aggregate on each element in rdd using fn
  - rdd.take(n)
    returns the first n elements from rdd
  - rdd.count()
    returns the number of elements in rdd

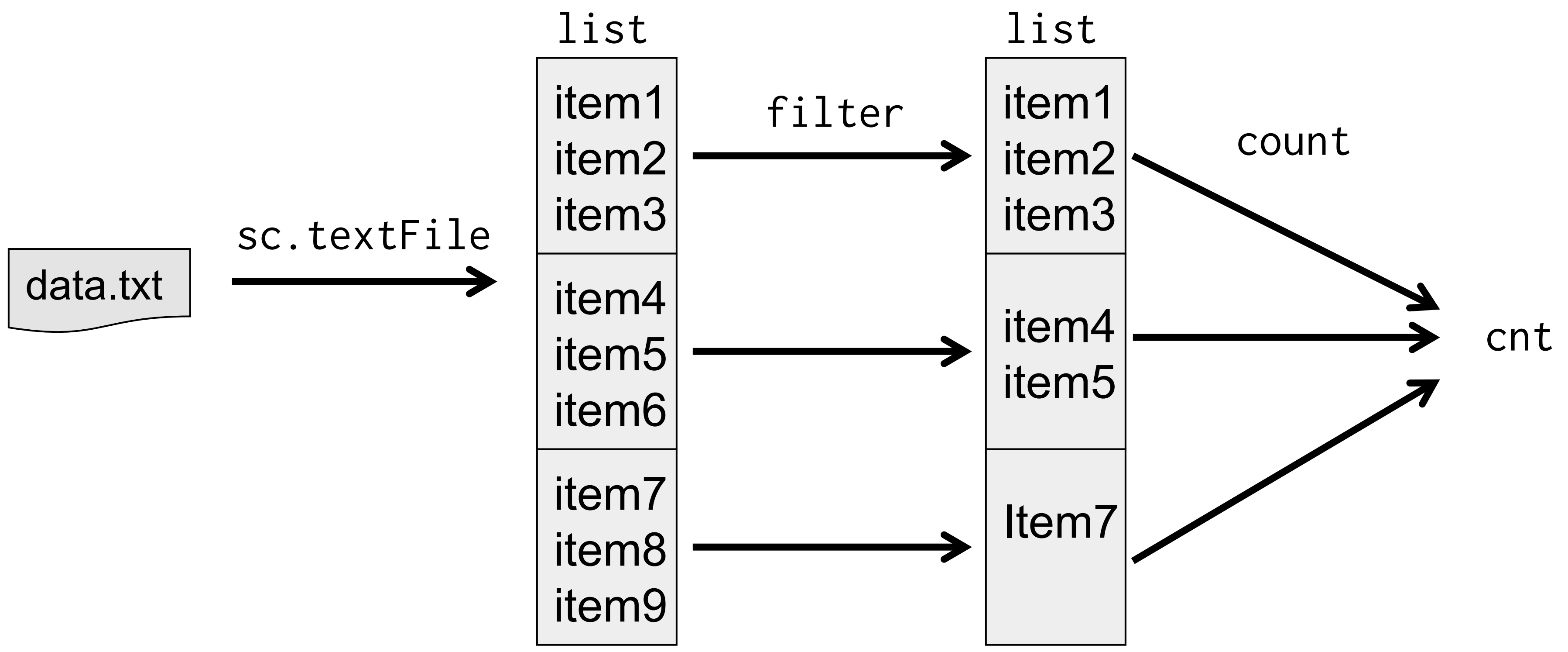
Example 1

```
list = sc.textFile(“data.txt”, 3)
cnt = list.count()
```



Example 2

```
list = sc.textFile(“data.txt”, 3)
filtered = list.filter(lambda a: a % 2 == 0)
cnt = filtered.count()
```



Lazy Evaluation

- Not all RDDs are computed immediately
- Spark instead remembers the set of transformations applied to the source dataset
  - Computations are applied when results are needed
  - This is known as **lazy evaluation**

- Advantages:
  - Optimizes across transformations
  - Recovers from failures
  - Kills slow workers and migrates jobs (recall the data skew problem)

Data Frames

- Data frame: collection of data organized into named columns
- Another data model besides RDD
- Example:

```
>>> users = sc.table("users")
>>> young = users[users.age < 21]
>>> young.groupBy("gender").count()
```

Spark Summary

- Programs structured around two data models:
  - RDDs
  - Data frames
- Emphasize on iteration over elements
  - Compare that with Map Reduce
- Lazy evaluation enables further optimization

Discussion

- We have seen three different programming models for analytics:
  - Writing queries (SQL)
  - Map Reduce
  - Spark
  - (there are many others, btw)

- Which one is better? Why?
- To what extent is each of these application dependent?