Large Scale Data Processing

Adaptation from

Magdalena Balazinska (Univ. of Washington)
Mining of Massive Datasets, by Rajaraman and Ullman
Alan Gates (Yahoo!)
Olston

Why Distributed Data Processing

- Hardware:
  - CPU speed does not increase ← hardware limitation
    too much data to process w/ a single machine
  - Instead: multicore
- Commodity clusters thousands of machines
  - Easy access to 1000 of nodes through cloud computing
  - Much cheaper than large mainframe ← commodity clusters replacing mainframes
- Big Data eg
  - Astronomy: high-resolution, high-frequency sky surveys
  - Medicine: digital records, MRI, ultrasound
  - Biology: sequencing data
  - User behavior data: click streams, search logs, …
    - Google and Facebook, but also Walmart and co…

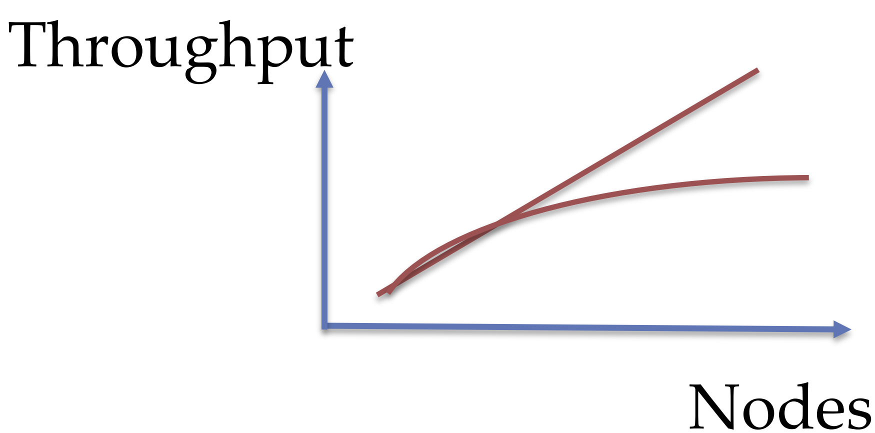
Distribution and Performance

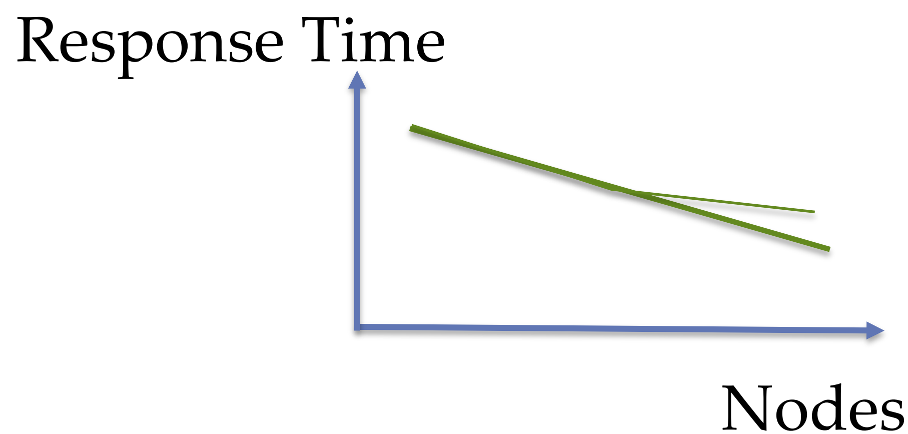
- Traditionally: scale-up
  - Improve performance by buying larger machine
- Distribution: scale-out multiple machines
  - Improve performance through parallel execution
- Performance metrics:
  - Throughput: transactions/queries per time unit # of requests
    time per transaction not important for this metric
    - The higher the better
    - Important for OLTP — online transaction processing
  - Response time: time for execution of an individual transaction/query
    time per transaction
    - The smaller, the better
    - Important for OLAP - online analysis processing

Speedup

more nodes or more processing capability

- Speedup (the data size remains the same)
  - More nodes → more throughput and/or lower response time





- Non-linear Speedup Startup costs
  - Coordination costs
  - Communication costs

overhead

  - Skew (equal distribution of load not possible)

in terms of data or computations

Scaleup

- Scaleup (the data size increases):
  - More nodes → have same throughput / same response time despite more data

- Non-linear Speedup and scaleup:
  - Data distribution overhead → distribute data between nodes
  - Non - parallelizable operations
    - aggregation
  - Communication costs – overhead, between nodes
  - Skew (equal distribution of data not possible)

Parallel Relational Database Systems

- A lot of DBS technology developed in 90s.
- Good understanding of distributed execution of relational algebra queries
- Both for
  - OLTP (online transaction processing): workload of short, update intensive queries, such as day-to-day banking, flight reservations etc.
  - OLAP (online analytical processing) / Decision support: workload of complex queries, mainly read-only
- Sophisticated and optimized operators (such as distributed join operators…)
- Expensive and specialized: Oracle, Teradata,…

Parallel Query Evaluation

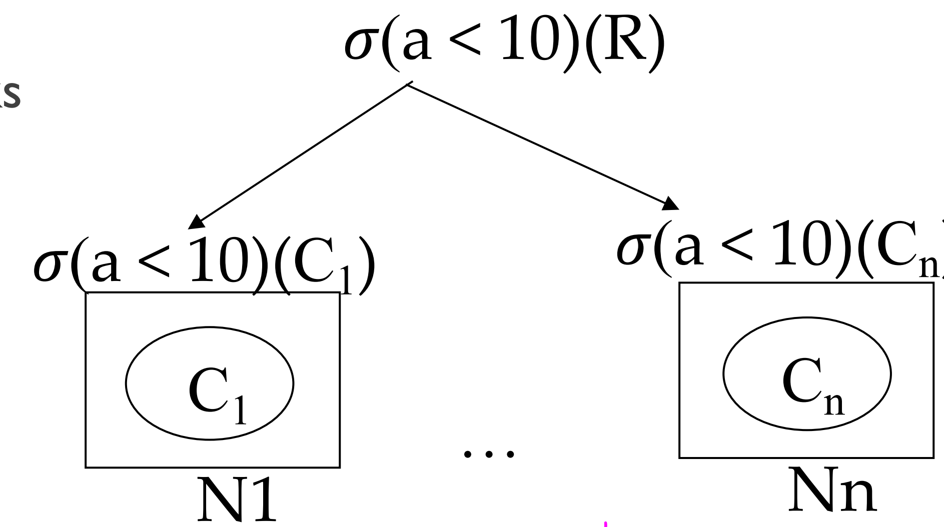
- Inter-query parallelism multiple queries running at same time
  - Different queries run in parallel on different processors; each query is executed sequentially
- Inter-operator parallelism single query → partition then join
  - Different operators within same execution tree run on different processors
  - Pipelining leads to parallelism
- Intra-operator parallelism
  - A single operator (e.g., scan, join) runs on many processors
  - Topic of this week

Horizontal Data Partitioning

- Data
  - Large table R($\underline{K}$, A, B C) ← relational model, partition data by rows
  - Key-value store KV($\underline{K}$,V) ← non-relational model hash-table stored in database
- Goal
  - partition into chunks $C_1, C_2, \ldots C_n$ of records stored at n nodes
- Hash partitioned on attribute X:
  - Record r goes to chunk i, according to hash function
  - Example hash-function: H = r.X mod n + 1
- Range partitioned on attribute X:
  - Partition range of X into: $-\infty = v_1 < v_2 < \ldots < v_{n-1} = \infty$
  - Record r goes to chunk i, if $v_{i-1} < r.v < v_i$

Example parallel operator: selection

- Execution path
  - Push selections to nodes with chunks
  - Execute locally at each node
  - Send result to coordinating node
  - Assemble result and return to user



replicate query on partitions

- Basics:
  - move and split of operators
  - Parallel operator execution
  - Transfer of records across nodes
  - Merge / post-processing at coordinating node
- Goals: Minimize CPU/IO/communication
  - Equal (non skewed) distribution of processing and I/O costs
    - Partition data equally ←
  - Keep communication costs low: execute locally, minimize transfer – minimize input/output

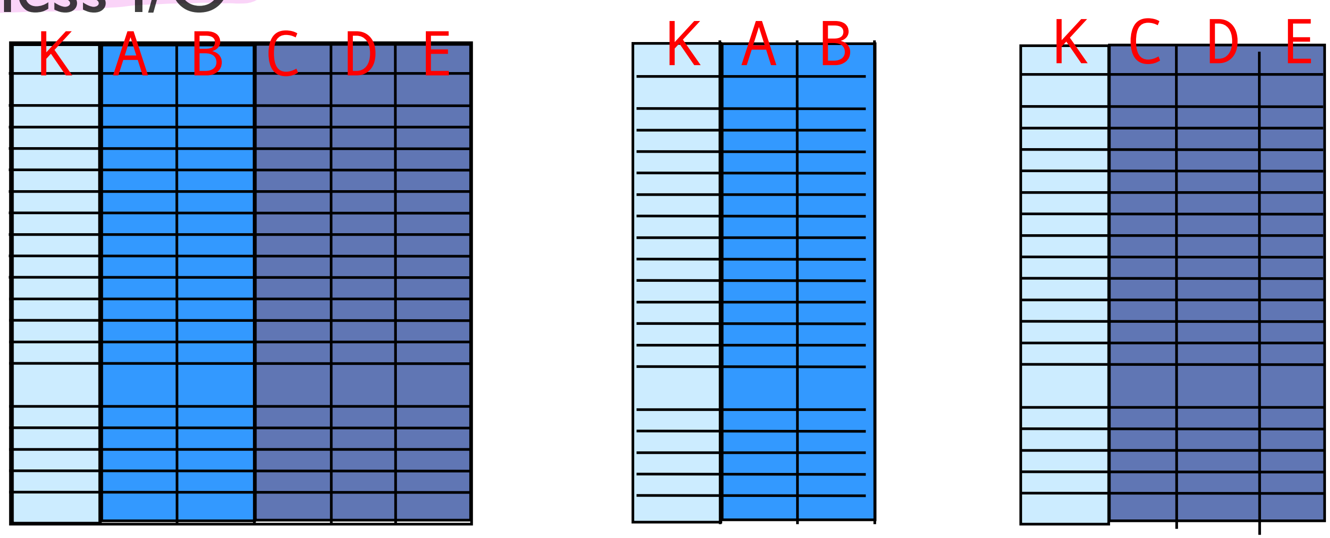
overhead

↳ send as little data as possible back and forth

Note: Vertical Data Partitioning

- Column-Stores
- Data: relations R(K, A, B, C,D,E) Very wide table → lots of columns
- Partition into RAB(K, A,B), RCDE(K, C,D,E)
- Query
  - *SELECT A from R where B > 50*
- *Query* only needs to access partition RAB
- Much less I/O

either only need to query one partition or can happen in parallel then join



Vertical Data Partitioning

- Why is the key replicated?

```
SELECT * FROM R
```

Equals

```
SELECT * FROM RAB, RCDE
WHERE RAB.K = RCDE.K
```

Map-reduce

— fundamental + basic framework for big data

- General-purpose distributed computing framework
- Can be applied to many types of queries; non-relational and relational
- Developed by Google; open-source version ***Hadoop*** developed by Yahoo led to quick success
- One initiative within the NoSQL movement

Data Processing at Massive Scale

- Massive Scale
  - Petabytes of data
  - 100s, 1000s, 10000s of servers
  - Many hours
- Failure becomes an issue
  - If medium-time-between failure is 1 year
  - Then 10000 servers have one failure / hour
  - Query execution must succeed even if individual nodes fail

need to handle failure / do failure recovery when it happens → need to gaurantee execution success even if there is a failure

Distributed Large-Scale File Systems

- Google DFS / Yahoo's Hadoop HDFS (sponsored by Yahoo)

  adoop istributed ile ystem

- Assumptions:
  - Files are large (terabytes….)
  - Files are rarely updated
- Main concepts

  file system distributed over nodes

  - Files are split into chunks, typically 64MBytes
    - (compare with 4K page size discussed so far)
  - Each chunk replicated for availability
  - Master node knows about location of chunks
    - Meta-repository (also replicated for fault-tolerance)

big chunks ⇒ less meta data

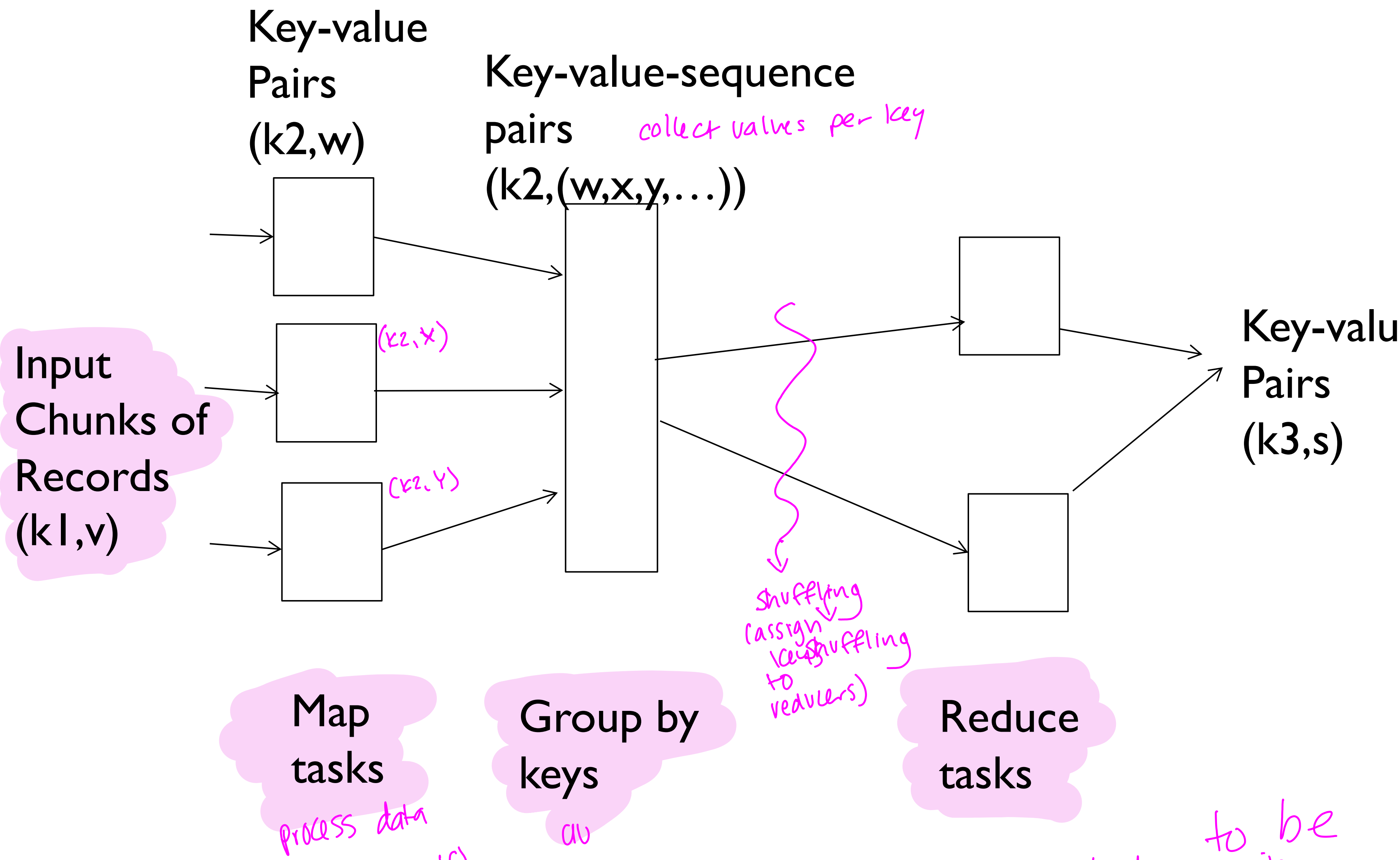
Map-reduce

- High-level programming model AND implementation for large-scale parallel data processing
- Programming model
  - Distribute data and each side read data records one by one (key-value pairs)
  - ***Map tasks:*** extract something interesting from records and output a new set of data records (key-value pairs)
  - Shuffle and sort (same key to same reduce task)
  - ***Reduce tasks:*** aggregate, summarize, filter
  - Write the results

get data + process

post process

Overview



framework needs data to be in (key,value)

Overview

- Input and output considered key/value pairs in order to be able to compose several map/reduce instances
- Keys and values themselves could be complex objects (including tuples).
- Map and Reduce functions are written by programmer
- Number of map tasks and reduce tasks given at start of program
- The rest done automatically (at least conceptually)

Example: Word Count

- Given: Document Set DS(K, documenttext)
- Output: For each word *w* occurring at least in one document of DS: indicate the number of occurrences of *w* in DS

Map Step

- Input Parameters from User
  - Number m of map tasks
  - Number r of reduce tasks
  - Data set = document set DS
- Map function written by User

m chunks

WordCountMap:

For each input key/value pair (dkey, dtext)

For each word w of dtext

Output key-value pair (w, 1)

each word as key
appearances as value
combine keys to get total count

- System splits input set into m partitions
- System creates m map tasks, gives each one partition
- Each map task executes map function on its partition
- Map step only completes once all map tasks are done

Shuffle and Reduce Steps

- System sorts map outputs by key and transforms all key/value pairs $(k, v_1), (k, v_2), \ldots (k, v_n)$ with same key k to one key/value-list pair $(k, (v_1, v_2, \ldots v_n))$
  - For Word count: all ('star', 1), ('star', 1), ('star', 1) … are transformed into one ('star', (1,1,1,….))
- System partitions output by key into *r* partitions
- Systems creates *r* reduce tasks and assigns each one partition
- Each reduce task executes user written reduce function

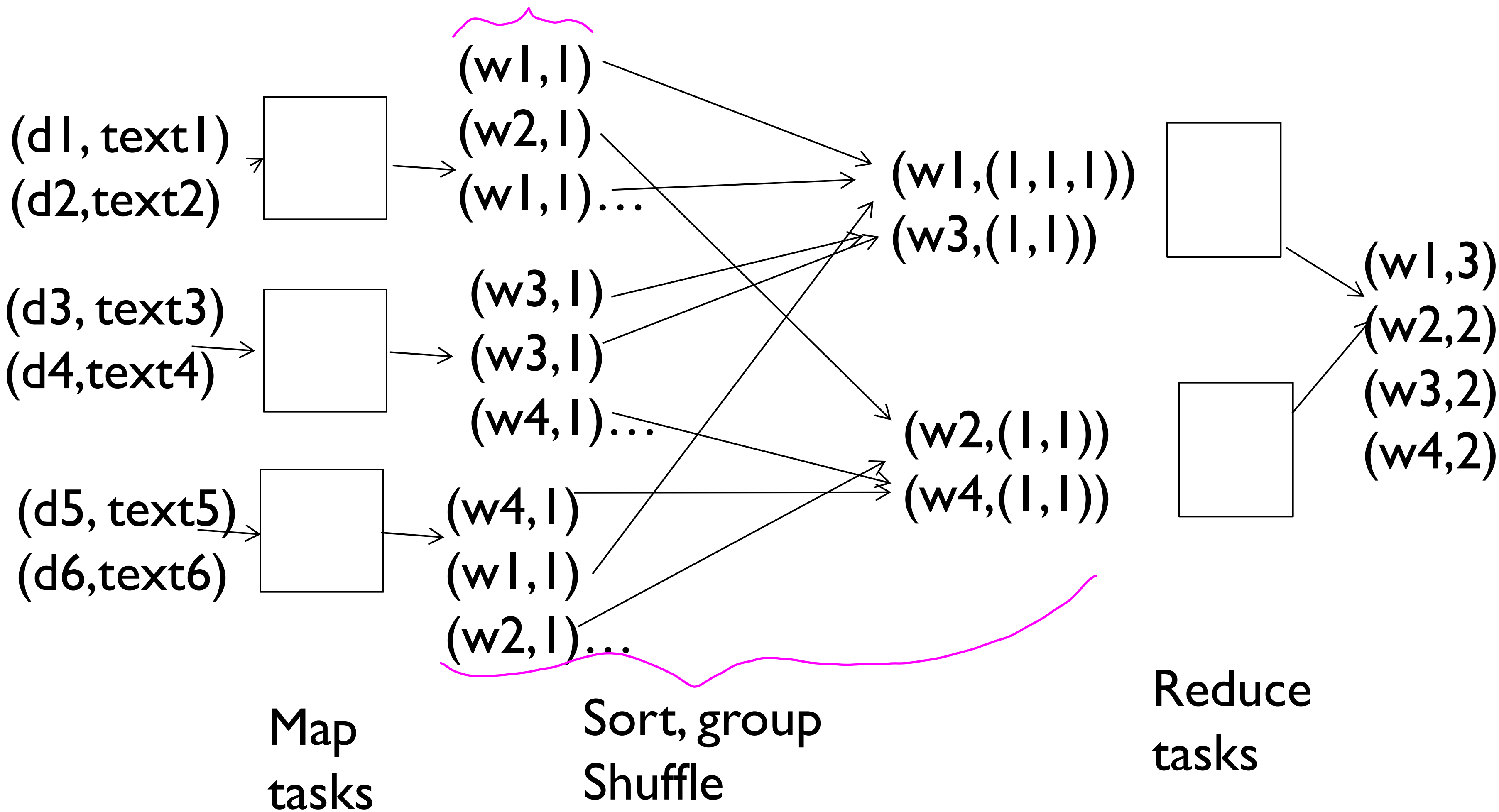
  WordCountReduce:

  For each input key/value-list pair $(k, (v_1, v_2, \ldots v_n))$

  Output (k, n)

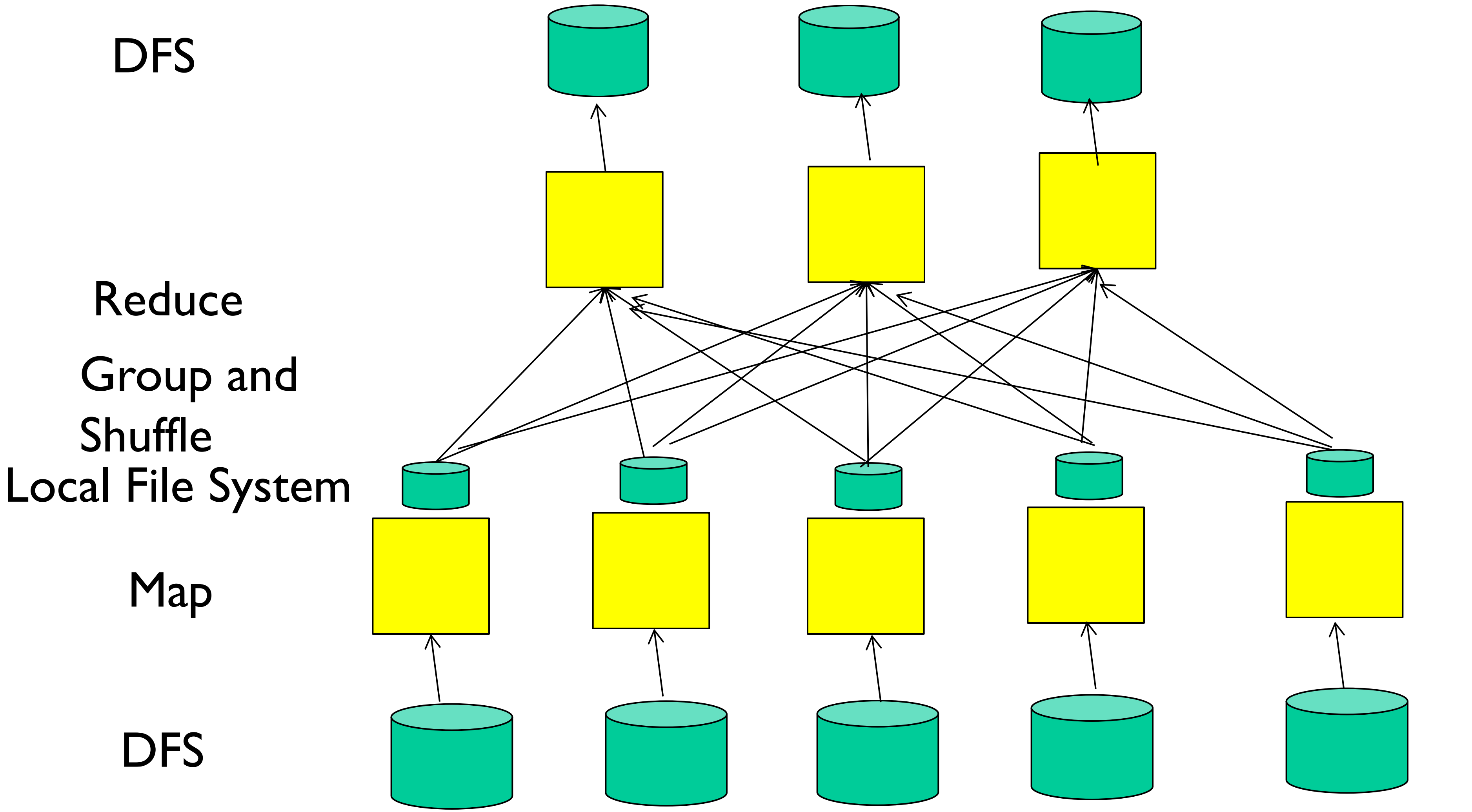
Example Execution

key value pairs



system has a certain # mappers + reducers

Once more as a tree



Phase Details

DFS

- Split into partitions
- At map tasks
  - Record reader
  - Map function
  - Possibly combine (explain later)
  - Write to local file

map location

- Group and shuffle
  - Group keys and aggregate value-lists
  - Copy from map location to reduce location
  - group keys and aggregate value-lists

reduce location

- At reduce tasks
  - Reduce function and write to file system

DFS

Combine

- Possible if reduce function commutative and associative
- Execute reduce function at each mapper on partial result of mapper
- Reduces data to be transferred by shuffle
- Example word count:
  - At each mapper: count the occurrences of each word for all documents read by this mapper
  - For each mapper, there is then only one key-value pair for each word and the value is the number of occurrences

Implementation

- There is one master node controlling execution
- Master partitions file into m partitions
- Master assigns workers (server processes) to m map tasks
- Workers executing map tasks write to local disk
- Master assigns workers to r reduce tasks
- Reduce workers implement group and shuffle (read from map disks) and execute reduce tasks

master node handles any failures

Failures

- Failures are detected by master
  - Failure of map task during map phase
    - master assigns new worker to map task
  - Failure of map task during reduce phase
    - Master assigns new worker to map task to redo (as data stored locally)
  - Failure of reduce task during reduce phase
    - Master assigns new worker to reduce task
- Straggler

  slow node
  - A machine that takes unusually long to complete one of its last tasks
    - Maybe some I/O problem, too many other tasks…
  - Solution: back execution of last few remaining in-progress tasks

System is not only for queries

Selection with Map/reduce

- Assume R(A, B, C, D) relation (no duplicates)
- Selection with condition c on R

① partition data into C_1, C_2

  - Map:
    - for each tuple t of R for which condition c holds, output (a,(b,c,d))

key value

  - Reduce:
    - $\boxed{\text{identity}}$, that is, output (a,(b,c)) just output no reducing to do

R C

SELECT * FROM Users WHERE experience = 10

key value

	uid	uname	experience	age
C_1	123	(Dora	2	13)
	132	(Bug	10	60)
C_2	267	(Sakura	10	15)
	111	(Cyphon	8	35)

Map

132	(Bug	10	60)
267	(Sakura	10	15)

Reduce

132	(Bug	10	60)
267	(Sakura	10	15)

Join with Map/reduce

- Natural Join R(A,B,C) with Q(C,D,E)
  - Map: Join column is key
    - For each tuple (a,b,c) of R, output (c, $\underbrace{\text{(R, (a,b))}}_{\text{value}}$) key
    - For each tuple (c,d,e) of Q, output (c, (Q, (d,e)))

    separate into key, values to join on the key
  - Group and shuffle will aggregate all key/value pairs with same c-value → combine values with same key
  - Reduce

    For each tuple (c, value-list)

    (e.g., value-list = (R, (a1,b1)), (R, (a2,b2)),…(Q,(d1,e1)),…))

    Rt = Qt = empty;

    for each v=(rel,tuple) in value-list

    if v.rel = R: insert tuple into Rt else insert tuple into Qt

    for v1 in Rt, for v2 in Qt, output(c,(v1,v2))

    Basically produces all combinations (c, (ai,bi,dj,ej))

distributes

```
SELECT *
FROM Users   u, GroupMembers g
WHERE u.uid = g.uid
```

users

uid	uname	experience	age
123	(Dora	2	13)
132	(Bug	10	60)
267	(Sakura	10	15)
111	(Cyphon	8	35)

group

uid	gid	stars
(123	G1)	2
(132	G1)	5
(132	G2)	3
(132	G3)	1
(123	G2)	4
(111	G4)	2

Map

(key, (R(a,b,c)))

key	value			
123	(U	(Dora	2	13))
132	(U	(Bug	10	60))
267	(U	(Sakura	10	15)
111	(U	(Cyphon	8	35)

key	value		
123	(GM	(G1	2))
132	(GM	(G1	5))
132	(GM	(G2	3))
132	(GM	(G3	1))
123	(GM	(G2	4))
111	(GM	(G4	2))

SELECT *
FROM Users u, GroupMembers g
WHERE u.uid = g.uid

123	(U	(Dora	2	13))
132	(U	(Bug	10	60))
267	(U	(Sakura	10	15))
111	(U	(Cyphon	8	35))

123	(GM	(G1	2))
132	(GM	(G1	5))
132	(GM	(G2	3))
132	(GM	(G3	1))
123	(GM	(G2	4))
111	(GM	(G4	2))

map outputs

Shuffle group by key (uid)

123	((U	(Dora	2	13)) ,	(GM	(G1	2)) ,	(GM	(G2	4)))			
132	((U	(Bug	10	60)) ,	(GM	(G1	5)) ,	(GM	(G2	3)) ,	(GM	(G3	1)))
267	((U	(Sakura	10	15))									
111	((U	(Cyphon	8	35) ,	(GM	(G4	2)))						

Shuffle

output

123	((U	(Dora	2	13))	(GM	(G1	2))	(GM	(G2	4)))				
132	((U	(Bug	10	60))	(GM	(G1	5))	(GM	(G2	3))	(GM	(G3	1)))
267	((U	(Sakura	10	15))										
111	((U	(Cyphon	8	35)	(GM	(G4	2)))							

no reduce

Reduce

123	(Dora	2	13	G1	2)
123	(Dora	2	13	G2	4)
132	(Bug	10	60	G1	5)
132	(Bug	10	60	G2	3)
132	(Bug	10	60	G3	1)
111	(Cyphon	8	35	G4	2)

U_T

Dora	2	13

GM_T

G1	2
G2	4

U_T

Bug	10	60

GM_T

G1	5
G2	3
G3	1

Projection with Map/reduce

- Projection on B,C of R
  - Map:
    - desired columns key
    - key
    - value place holder
    - for each tuple t=(a,(b,c)) of R, let t'=(b,c): output (t', 0)
  - There might now be duplicates, that is several (t', 0) tuples; the group function will aggregate them to → (t', 0, 0,… 0)) removes dups
  - Reduce:
    - once per key → distinct
    - for each tuple (t', (0,0,0…), output $\boxed{(t', 0)}$

SELECT DISTINCT uname, age FROM Users

partition

	uid	uname	experience	age
C_1	123	(Dora	2	13)
	132	(Bug	10	60)
C_2	267	(Sakura	10	15)
	111	(Dora	8	13)

Map

(Dora	13)	0
(Bug	60)	0
(Sakura	15)	0
(Dora	13)	0

Shuffle

(Dora	13)	(0,0)	…

Reduce

(Dora	13)	0	…

Group BY Map/reduce

- Grouping: SELECT b, max(c) GROUP BY b
  - Map:
    - for each tuple (a,(b,c)) of R , output (b, c)

      group as key
      output value as value
  - Group and shuffle will create for each value *b* a key/value-list (b, (c1, c2,…)) groups
  - Reduce:
    - for each (b, (c1, c2,…)) perform aggregation (e.g., c1+c2, …)

      to get max per group

whatever you're grouping by should be the key

SELECT experience, max(age) FROM Users
GROUP BY experience

	uid	uname	experience	age
C_1	123	(Dora	2	13)
	132	(Bug	10	60)
C_2	267	(Sakura	10	15)
	111	(Cyphon	8	35)

Map

2	13
10	60
10	15
8	35

Shuffle

2	13
10	(60,15)
8	35

Reduce

(picks max)

2	13
10	60
8	35