

Database System Internals
MapReduce

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

March 4, 2020 CSE 444 - Winter 2020 1

1

This lecture

Data model?	Relational
Scaleup goal?	OLAP
Architecture?	Shared-Nothing

March 4, 2020 CSE 444 - Winter 2020 9

9

This lecture

Data model?	Relational ↓ text/kv-pairs
Scaleup goal?	OLAP
Architecture?	Shared-Nothing

March 4, 2020 CSE 444 - Winter 2020 10

10

References

- **MapReduce: Simplified Data Processing on Large Clusters.** Jeffrey Dean and Sanjay Ghemawat. OSDI'04
- Mining of Massive Datasets, by Rajaraman and Ullman, <http://i.stanford.edu/~ullman/mmds.html>
 - Map-reduce (Section 20.2);
 - Chapter 2 (Sections 1,2,3 only)

March 4, 2020 CSE 444 - Winter 2020 11

11

Outline

- Review high-level MR ideas from 344
- Discuss implementation in greater detail

March 4, 2020 CSE 444 - Winter 2020 12

12

Map Reduce Review

- Google: [Dean 2004]
- Open source implementation: Hadoop
- MapReduce = high-level programming model and implementation for large-scale parallel data processing

March 4, 2020 CSE 444 - Winter 2020 13

13

MapReduce 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

March 4, 2020 CSE 444 - Winter 2020 14

14

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

March 4, 2020 CSE 444 - Winter 2020 15

15

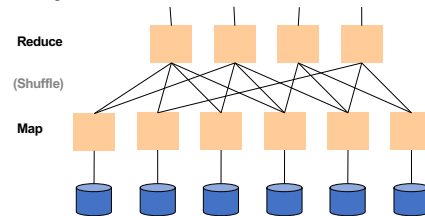
Data Storage: GFS/HDFS

- 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
 - HDFS: Hadoop File System
- Each data file is split into M partitions (64MB or more)
- Blocks are replicated & stored on random machines
- Files are append only

March 4, 2020 CSE 444 - Winter 2020 16

16

Observation: Your favorite parallel algorithm...



March 4, 2020 CSE 444 - Winter 2020 17

17

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

March 4, 2020 CSE 444 - Winter 2020 18

18

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

March 4, 2020 CSE 444 - Winter 2020 19

19

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

March 4, 2020 CSE 444 - Winter 2020 20

20

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

March 4, 2020 CSE 444 - Winter 2020 21

21

Example

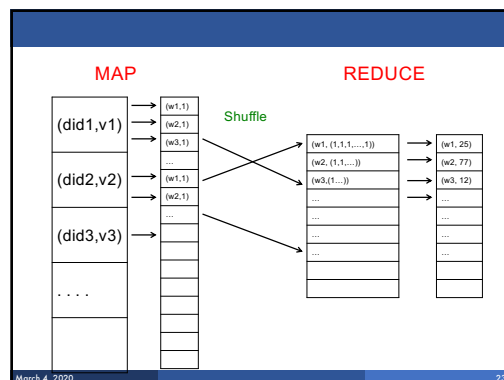
- Counting the number of occurrences of each word in a large collection of documents
- Each Document
 - The **key** = document id (**did**)
 - The **value** = set of words (**word**)

```
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));
```

March 4, 2020 CSE 444 - Winter 2020 22

22



March 4, 2020 CSE 444 - Winter 2020 23

23

Jobs vs. 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

March 4, 2020 CSE 444 - Winter 2020 24

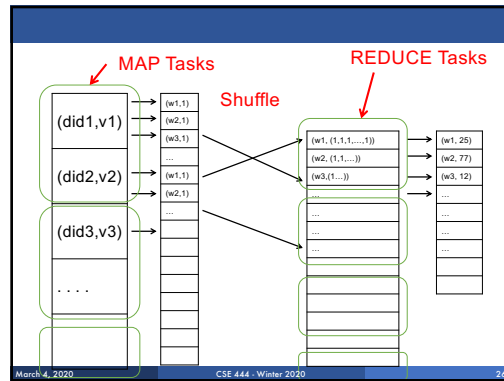
24

Workers

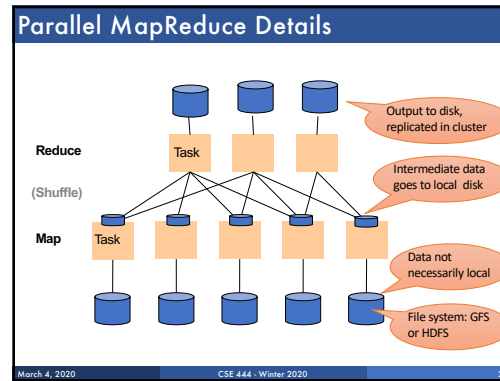
- A **worker** is a process that executes one task at a time
- Typically there is one worker per processor, hence 4 or 8 per node
- Often talk about "slots"
 - E.g., Each server has 2 map slots and 2 reduce slots

March 4, 2020 CSE 444 - Winter 2020 25

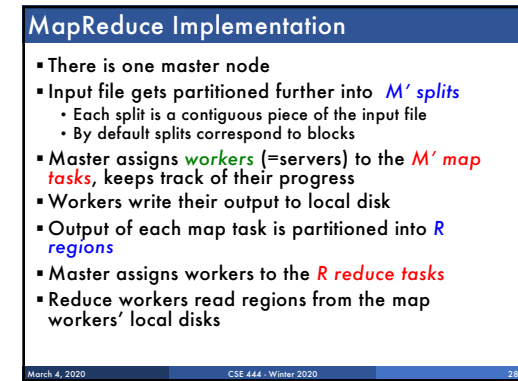
25



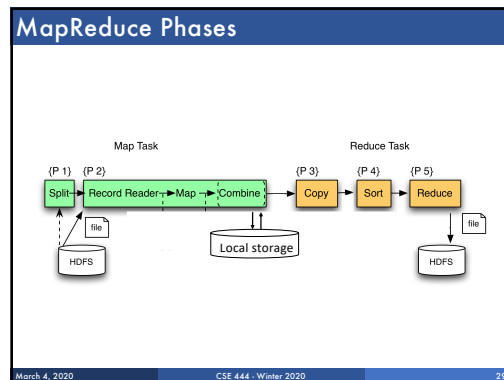
26



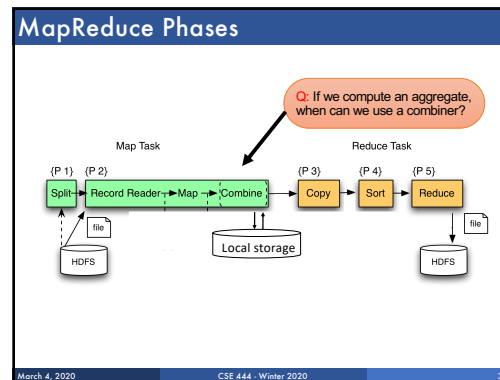
27



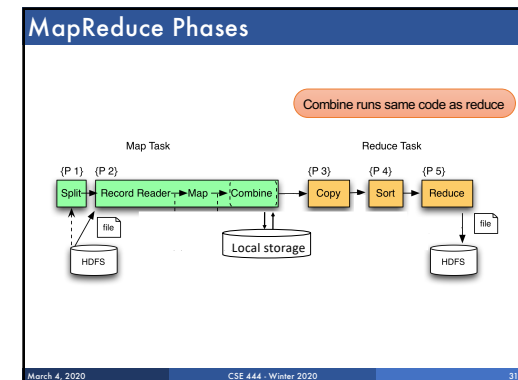
28



29



30



31

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
 - (≠ a parallel DBMS spreads ops across all nodes)

March 4, 2020 CSE 444 - Winter 2020 32

32

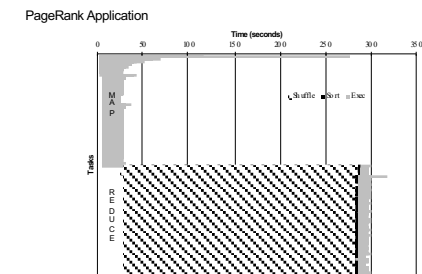
Interesting Implementation Details

- Backup tasks:
 - **Straggler** = a machine that takes unusually long time to complete one of the last tasks. Eg:
 - 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*

March 4, 2020 CSE 444 - Winter 2020 33

33

Skew



March 4, 2020 CSE 444 - Winter 2020 34

34

The State of MapReduce Systems

- 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 (MR and DBMS), with increased integration
- New systems emerged which improve over MapReduce: e.g. Spark

March 4, 2020 CSE 444 - Winter 2020 35

35

Declarative Languages on MR

- PIG Latin (Yahoo!)
 - Domain specific language, like Relational Algebra
 - Open source
- HiveQL (Facebook)
 - SQL-like language
 - Open source
- SQL / Tenzing (Google)
 - SQL on MR
 - Proprietary
 - Morphed into BigQuery

March 4, 2020 CSE 444 - Winter 2020 36

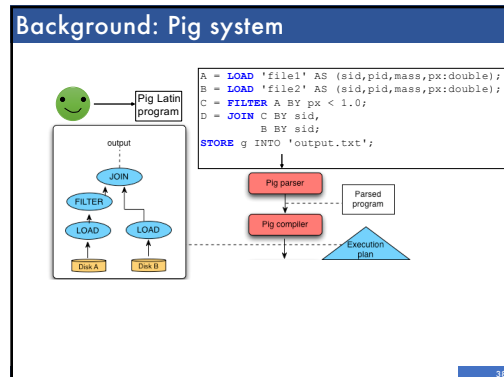
36

Relational Queries over MR

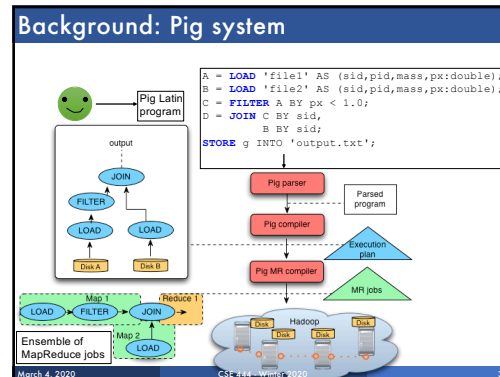
- Query → query plan
- Each operator → one MapReduce job
- Example: the Pig system

March 4, 2020 CSE 444 - Winter 2020 37

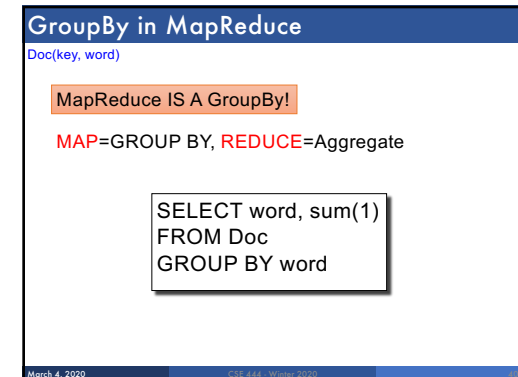
37



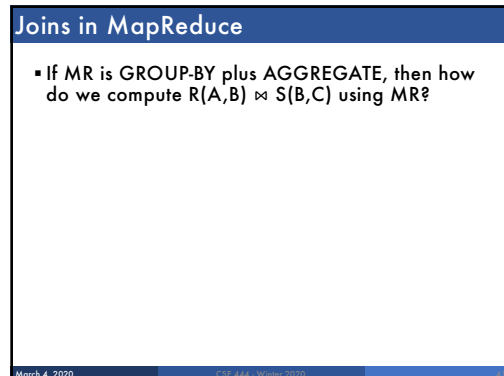
38



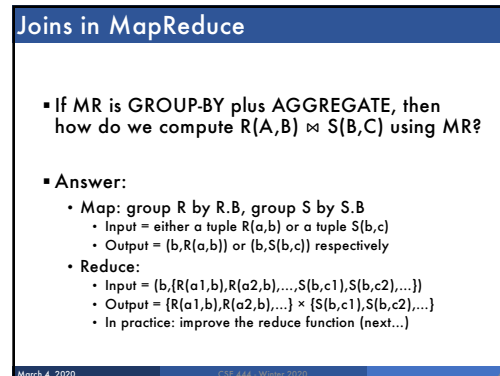
39



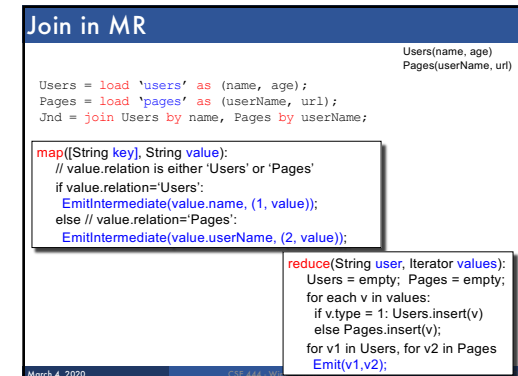
40



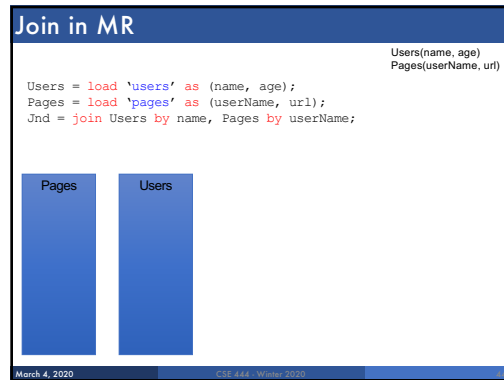
41



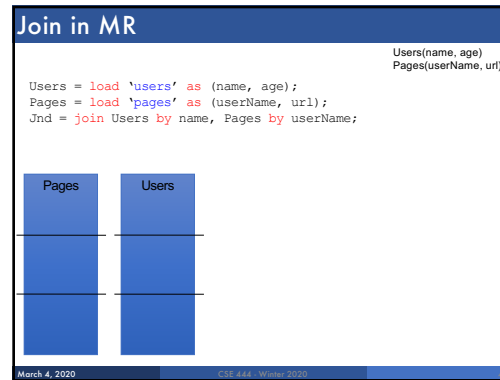
42



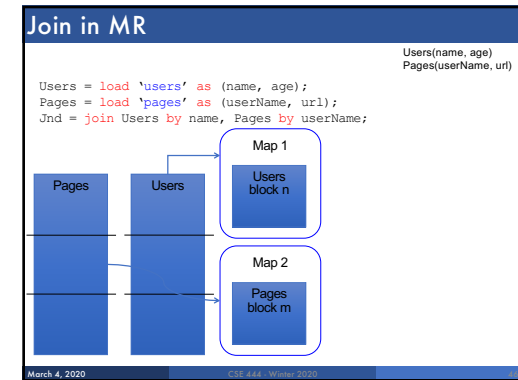
43



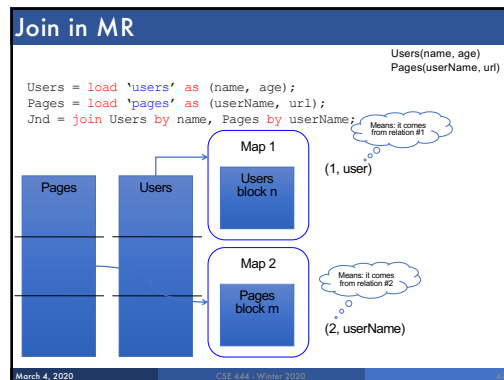
44



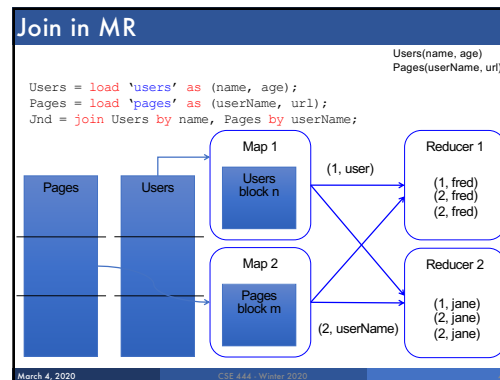
45



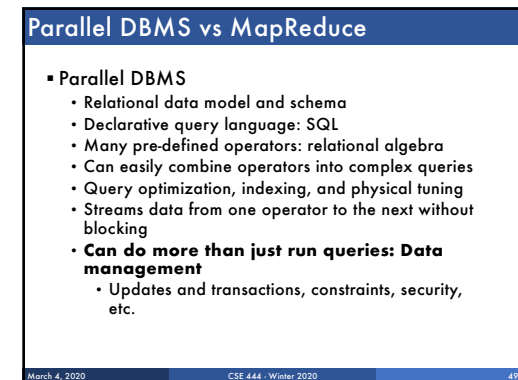
46



47



48



49

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**

Interesting historical reading:
MapReduce: A major step backwards by David DeWitt

March 4, 2020 CSE 444 - Winter 2020 50

50

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!**

March 4, 2020 CSE 444 - Winter 2020 51

51