UC San Diego

DSC 102
Systems for Scalable Analytics

Rod Albuyeh

Topic 4: Dataflow Systems

Chapter 2.2 of MLSys Book

Outline

- Beyond RDBMSs: A Brief History
- MapReduce/Hadoop Craze
- Spark and Dataflow Programming
- Scalable BGD with MapReduce/Spark
- Dataflow Systems vs Task-Parallel Systems

Q: *How can users be shielded*
from needing to think
about moving raw pages
between disk/RAM/network
to scale data-intensive programs?

Parallel RDBMSs

- Parallel RDBMSs are highly successful and widely used
- Typically shared-nothing data parallelism
- Optimized **runtime performance** + enterprise-grade features:
  - ANSI SQL & more
  - Business Intelligence (BI) dashboards/APIs
  - Transaction management; crash recovery
  - Indexes, auto-tuning, etc.

Q: *So, why did people need to go beyond parallel RDBMSs?*

Take CSE 132C for more on parallel RDBMSs

Beyond RDBMSs: A Brief History

- DB folks got blindsided by the rise of Web/Internet giants

- 4 new concerns of Web giants vs RDBMSs built for enterprises:
  - **Developability:** Custom data models and computations hard to program on SQL/RDBMSs; need for simpler APIs
  - **Fault Tolerance:** Need to scale to 1000s of machines; need for graceful handling of worker failure
  - **Elasticity:** Need to be able to easily upsize or downsize cluster size based on workload
  - **Cost:** Commercial RDBMSs licenses too costly; hired own software engineers to build custom new systems

A new breed of parallel data systems called **Dataflow Systems** jolted the DB folks from being complacent!

Outline

- Beyond RDBMSs: A Brief History
- MapReduce/Hadoop Craze
- Spark and Dataflow Programming
- Scalable BGD with MapReduce/Spark
- Dataflow Systems vs Task-Parallel Systems

What is MapReduce?

- A programming model for parallel programs on **sharded data** + **distributed system** architecture
- **Map** and **Reduce** are terms from functional PL; software/data/ML engineer implements logic of Map, Reduce
- System handles data distribution, parallelization, fault tolerance, etc. *under the hood*
- Created by Google to solve “simple” data workload: index, store, and search the Web!
- Google’s engineers started with MySQL! Abandoned it due to reasons listed earlier (developability, fault tolerance, elasticity, etc.)

https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

What is MapReduce?

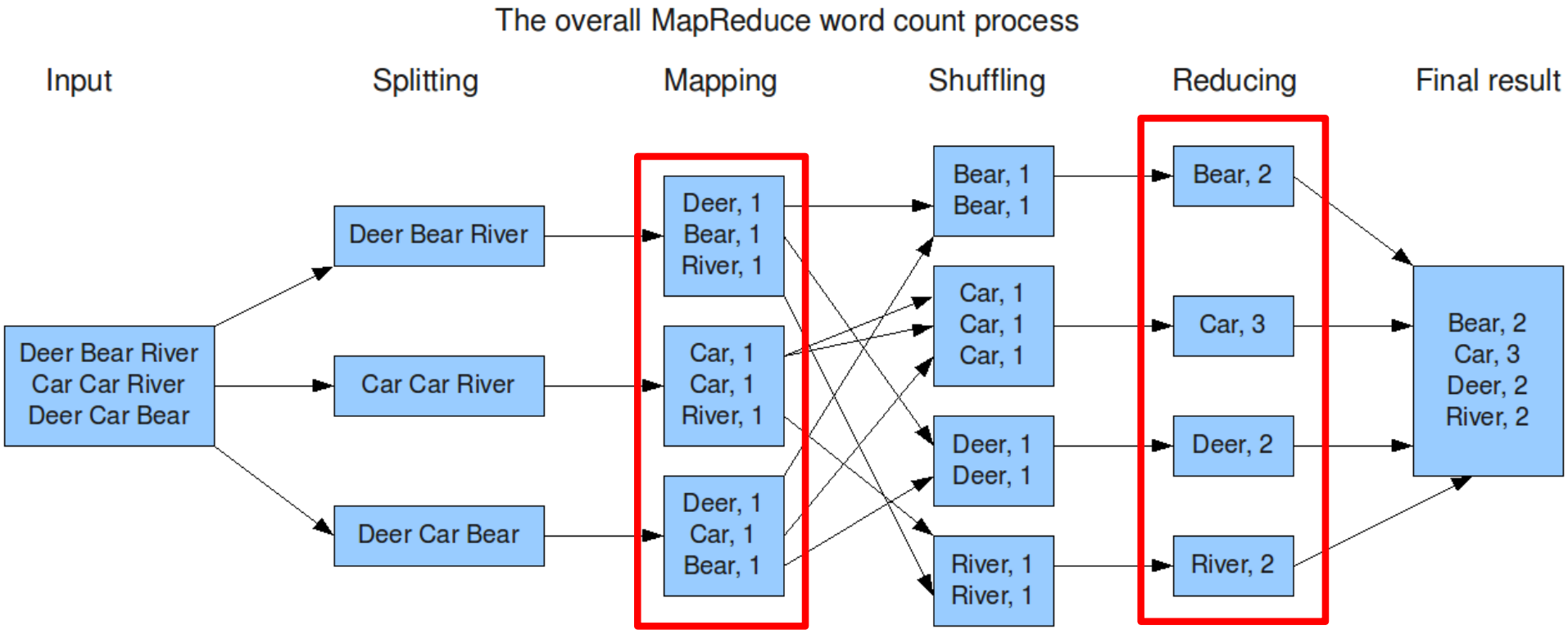
- **Standard example**: count word occurrences in a doc corpus
- **Input**: A set of text documents (say, webpages)
- **Output**: A dictionary of unique words and their counts

```
function map (String docname, String doctext) :
        for each word w in doctext :
                emit (w, 1)
function reduce (String word, Iterator partialCounts) :
        sum = 0
        for each pc in partialCounts :
                sum += pc
        emit (word, sum)
```

Part of MapReduce API

How MapReduce Works

Parallel flow of control and data during MapReduce execution:



Under the hood, each **Mapper** and **Reducer** is a separate process; Reducers face barrier synchronization (BSP)

Fault tolerance achieved using **data replication**

Benefits and Catch of MapReduce

- **Goal:** High-level *functional* ops to simplify data-intensive programs
- **Key Benefits:**
  - Map() and Reduce() are highly general; any data types/structures; great for ETL, text/multimedia
  - Native scalability, large cluster parallelism
  - System handles fault tolerance automatically
  - Decent FOSS stacks (Hadoop and later, Spark)
- **Catch:** Users must learn "art" of casting program as MapReduce
  - Map operates record-wise; Reduce aggregates globally
  - But MR libraries now available in many PLs: C/C++, Java, Python, R, Scala, etc.

Abstract Semantics of MapReduce

- **Map():** Process one "record" at a time *independently*
  - A record can physically *batch* multiple data examples/tuples
  - Dependencies across Mappers *not* allowed
  - *Emit* 1 or more key-value pairs as output(s)
  - Data types of input vs. output can be different
- **Reduce():** Gather all Map outputs across workers sharing same key into an Iterator (list)
  - Apply *aggregation* function on Iterator to get final output(s)
- **Input Split:**
  - Physical-level shard to batch many records to one file "block" (HDFS default: 128MB?)
  - User/application can create *custom* Input Splits

Emulate MapReduce in SQL?

Q: How would you do the word counting in RDBMS / in SQL?

- First step: **Transform** text docs into relations and load:

  Part of the ETL stage

  Suppose we pre-divide each doc into words w/ schema:

  DocWords (DocName, Word)

- Second step: a single, simple SQL query!

```
SELECT      Word, COUNT (*)
FROM        DocWords
GROUP BY    Word
[ORDER BY   Word]
```

Parallelism, scaling, etc. done by RDBMS under the hood

More MR Examples: Select Operation

- **Input Split**:
  - Shard table tuple-wise
- **Map():**
  - On tuple, apply selection condition;

    if satisfies, emit key-value (KV) pair

    with dummy key, entire tuple as value
- **Reduce():**
  - Not needed! No cross-shard aggregation here
- These kinds of MR jobs are called "**Map-only**" jobs

More MR Examples: Simple Agg.

- Suppose it is *algebraic* aggregate (SUM, AVG, MAX, etc.)
- **Input Split**:
  - Shard table tuple-wise
- **Map():**
  - On agg. attribute, compute incr. stats;

    emit pair with single global dummy key

    and incr. stats as value
- **Reduce():**
  - Since only one global dummy key, Iterator has *all* sufficient stats to unify into global agg.

More MR Examples: GROUP BY Agg.

- Assume it is *algebraic* aggregate (SUM, AVG, MAX, etc.)
- **Input Split**:
  - Shard table tuple-wise
- **Map():**
  - On agg. attribute, compute incr. stats;

    emit pair with *grouping attribute as key*

    and stats as value
- **Reduce():**
  - Iterator has all suff. stats *for a single group*;

    unify those to get result for that group
  - Different reducers will output different groups' results

More MR Examples: Matrix Sum of Squares

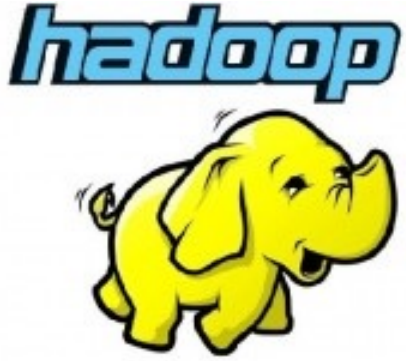
- Very similar to simple SQL aggregates
- **Input Split**:
  - Shard table tuple-wise
- **Map():**
  - On agg. attribute, compute incr. stats;

    emit pair with single global dummy key

  and stats as value
- **Reduce():**
  - Since only one global dummy key,

    Iterator has *all* sufficient stats to unify into global agg.

What is Hadoop then?



- FOSS system implementation with
  - MapReduce as programming model, and
  - HDFS as filesystem
- MR user API; input splits, data distribution, shuffling, and fault tolerances handled by Hadoop under the hood
- Exploded in popularity in 2010s: 100s of papers, 10s of products
- A “revolution” in scalable+parallel data processing that took the DB world by surprise
- But nowadays Hadoop largely supplanted by Spark

Note: Do not confuse MR for Hadoop or vice versa!