Big Data (DS-GA 1004) – Lecture 3 Finals Preparation Notes

Fully based on Week 3 Slides + Lecture Transcript + Expanded Knowledge

Spring 2025

1 Distributed Computation: Introduction to MapReduce

2 Announcements

- Everyone has an HPC (High-Performance Computing) account.
- Lab 3: Hands-on with MapReduce.
- HW1 is due tomorrow (02/11).
- Quiz during this week's lab.
- HW2 releases this week (due 02/27).

3 Historical Context: How Data Management Evolved

3.1 Timeline

- **1940s**: Vacuum tubes and plugboards (hardware dominated).
- **1950s**: Punchcards (software programs emerged).
- **1960s**: File systems, custom software per application/query.
- **1970s**: **Relational model** proposed (Codd, 1970), SQL standard.
- **1980s**: Relational DBMS become commodity technology.
- **1990s**: Web explosion; Distributed/parallel computing begins.
- **2000s**: **MapReduce introduced** (Google) for massive scale.
- **2010s**: Cloud computing, cluster management, dataframes.

4 Concept Review: RDBMS

- **Relations** standardize data structure.
- **SQL** standardizes data access.
- **DBMS** abstracts internal complexities.
- **ACID Transactions** guarantee data safety.

5 Confusions Resolved

5.1 Keys in Computer Science

- Think of **key** as a unique **identifier** (ID), unless specified otherwise.

5.2 Normalization

- **1NF**: Each field contains atomic (indivisible) values.
- **Higher NFs**: Ensure no partial or transitive dependencies.
- **Heuristic**: Break into multiple tables linked by primary/foreign keys.

5.3 Indexing

- Lookup tables to accelerate search.
- Variants include clustered vs non-clustered indexes.
- **SQL Syntax**: `CREATE INDEX idx_name ON table(column);`

6 Introduction to MapReduce

6.1 Motivation

- Processing N massive documents $\Rightarrow \Omega(N)$ time sequentially.
- Embarrassingly parallel problem $\Rightarrow$ process documents independently.

6.2 Google's Context

- Text indexing for entire web.
- Need to distribute workload efficiently & aggregate results.

7 MapReduce Framework

7.1 Core Idea

- Program consists of only two stages: **Map** and **Reduce**.
- **Map Phase**: Apply a function independently to data splits to produce (key, value) pairs.
- **Shuffle Phase**: Group all identical keys together.
- **Reduce Phase**: Aggregate values per key.

7.2 Example: Word Count

- **Mapper**: Emit (word, 1) for each word in document.
- **Shuffle**: Group identical words.
- **Reducer**: Sum counts for each word.

8 Functional Programming Basis

8.1 Mapping

- `map(f, [x1, x2, ..., xn])` $\rightarrow [f(x1), f(x2), ..., f(xn)]$

8.2 Reducing

- `reduce(g, [x1, x2, ..., xn])` recursively applies function g to combine values.

9 Important Properties

9.1 Associativity

- $(a \oplus b) \oplus c = a \oplus (b \oplus c) \Rightarrow$ Order of reduction doesn't matter.

9.2 Determinism

- Same input yields the same output.
- No side effects.

9.3 Scalability

- More machines $\Rightarrow$ Linear speed-up $\approx$ possible.

10 Working with MapReduce: Practical Tips

- Prefer integers or strings for keys (floating point issues).
- Design simple `map` and `reduce` functions.
- Use combiners to reduce data transfer when possible.

10.1 Combiner Example

- Local reduction before shuffle phase.
- Word count example: sum local counts.

11 Challenges and Drawbacks

11.1 Key Skew

- Uneven distribution of keys $\Rightarrow$ Some reducers overloaded.

11.2 Not Suitable For

- Iterative algorithms (e.g., gradient descent).
- Interactive applications (e.g., real-time exploration).

12 Critiques of MapReduce (Stonebraker et al.)

- Too low-level $\rightarrow$ No schema awareness.
- Lacks indexes, views, integrity constraints.
- Not novel: Earlier systems had partitioning & aggregation.
- Not compatible with traditional DBMS ecosystem.

13 Why was MapReduce so impactful?

- Simple abstraction; easy to learn and deploy.
- Fits many one-shot batch-processing needs.
- Democratized distributed programming.

14 Expanded Knowledge: Legacy and Beyond

14.1 Hadoop

- Open-source implementation of MapReduce.
- Built atop HDFS (Hadoop Distributed File System).

14.2 Modern Successors

- Apache Spark (supports iterative algorithms, faster in-memory computation).
- Flink, Dask for advanced distributed workflows.

14.3 Design Philosophy

- Divide and Conquer ("Divide et Impera").
- Stateless computation where possible.
- Embrace locality to minimize communication.

15 Summary

- MapReduce enabled scalable parallel computation.
- Great for batch processing, bad for iterative ML.
- Concepts of mapping, shuffling, and reducing remain foundational.

16 Next Week

- Hadoop Distributed File System (HDFS): How to manage distributed storage.
- Hands-on with HDFS.