SE446: Big Data Engineering

Week 3A: MapReduce Concepts

Prof. Anis Koubaa

Alfaisal University - College of Engineering

Spring 2026

Today's Agenda

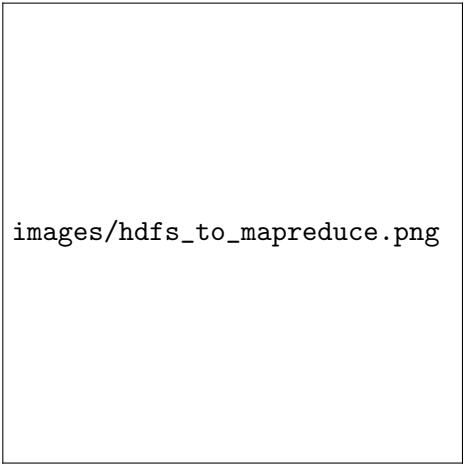
Recap: Why HDFS Alone Is Not Enough

HDFS provides:

- ✔ Distributed storage
- ✔ Fault tolerance
- ✔ Scalability

But how do we process data?

- Read 10 TB from HDFS?
- Analyze on one machine?
- ✖ Bottleneck!



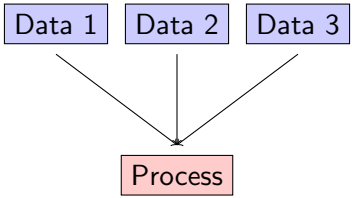
The Big Idea: Move Computation to Data

Traditional Approach

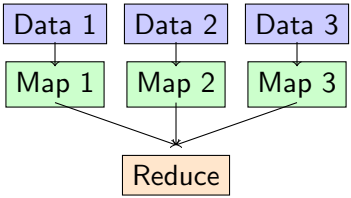
Move data to computation → Network bottleneck

MapReduce Approach

Move computation to data → Process locally, combine results



Traditional



MapReduce

MapReduce: Origin Story

- **2004**: Google publishes landmark paper
- **Problem**: Index the entire web (billions of pages)
- **Solution**: Simple programming model for distributed processing
- **2006**: Doug Cutting implements open-source version $\rightarrow$ Hadoop

Key Insight

Many data processing tasks follow the same pattern:

1. Process each record independently (Map)
2. Group by some key
3. Aggregate groups (Reduce)

MapReduce in One Slide

$$\textbf{map}(\text{key, value}) \rightarrow \text{list(key', value')}$$

$$\textbf{reduce}(\text{key', list(value')}) \rightarrow \text{list(value'')}$$

Map Phase:

- Process each input record
- Emit (key, value) pairs
- Runs in parallel across nodes

Reduce Phase:

- Receive all values for a key
- Aggregate/combine values
- Produce final output

Classic Example: Word Count

Input: 3 documents with text
Goal: Count occurrences of each word

Mapper (Python):

```
def mapper(doc_id, text):
    for word in text.split():
        emit(word, 1)
```

Input: "Hello World Hello"
Output:

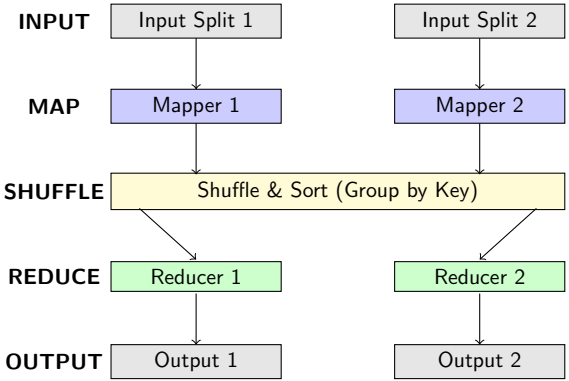
- (Hello, 1)
- (World, 1)
- (Hello, 1)

Reducer (Python):

```
def reducer(word, counts):
    emit(word, sum(counts))
```

Input: (Hello, [1, 1])
Output: (Hello, 2)

Data Flow: Step by Step



Real-World Example: Chicago Crime Data

Dataset: Chicago crime reports (columns):

- ID, Case Number, Date
- **Primary Type** (THEFT, ASSAULT, BATTERY, ...)
- Description, Location Description
- Arrest (True/False)
- District, Ward, Community Area

Question: How many crimes of each type occurred?

Crime Count: Mapper

```
def crime_mapper(record):
    """
    Input: One crime record (dictionary)
    Output: (crime_type, 1)
    """
    crime_type = record['Primary Type']
    return (crime_type, 1)

# Example
record = {'Primary Type': 'THEFT', 'District': 11, ...}
crime_mapper(record)  # Output: ('THEFT', 1)
```

Key Point: Mapper processes ONE record at a time, emits (key, value)

Crime Count: Reducer

```
def crime_reducer(crime_type, counts):
    """
    Input: crime_type (key), list of counts (values)
    Output: (crime_type, total_count)
    """
    total = sum(counts)
    return (crime_type, total)

# Example
crime_reducer('THEFT', [1, 1, 1, 1, 1])
# Output: ('THEFT', 5)
```

Key Point: Reducer receives ALL values for ONE key

Complete Data Flow

Phase	Input	Operation	Output
Input	CSV file	Split by lines	Records
Map	{"Type": "THEFT"}	Extract type, emit 1	("THEFT", 1)
Map	{"Type": "ASSAULT"}	Extract type, emit 1	("ASSAULT", 1)
Map	{"Type": "THEFT"}	Extract type, emit 1	("THEFT", 1)
Shuffle	All map outputs	Group by key	"THEFT" $\rightarrow$ [1, 1] "ASSAULT" $\rightarrow$ [1]
Reduce	"THEFT", [1, 1]	Sum	("THEFT", 2)
Reduce	"ASSAULT", [1]	Sum	("ASSAULT", 1)

The Shuffle Phase

Often Overlooked but Critical!

Shuffle is the **most expensive** phase - involves network transfer

What happens during Shuffle:

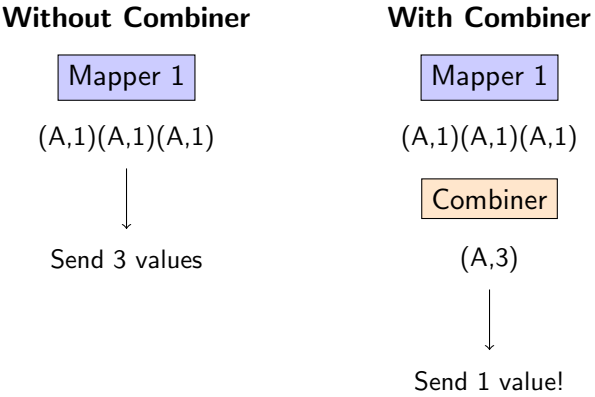
1. Mapper outputs are **partitioned** by key
2. Data is **sorted** by key
3. Data is **transferred** to appropriate reducer
4. Values for same key are **grouped** together

Example:

- Mapper 1 emits: (A, 1), (B, 2), (A, 3)
- Mapper 2 emits: (B, 4), (A, 5)
- After shuffle: A $\rightarrow$ [1, 3, 5], B $\rightarrow$ [2, 4]

Combiners: Local Aggregation

Problem: Shuffle transfers lots of data
Solution: Combine locally before shuffle!



Note: Combiner must be **associative** and **commutative** (like sum, max, min)

Partitioner: Key Distribution

Default: Hash partitioning (hash(key) % num_reducers)

Why customize?

- Ensure related keys go to same reducer
- Balance load across reducers

Example: Analyzing crimes by year

- Key: "2023-THEFT", "2023-ASSAULT", "2024-THEFT"
- Partitioner: Extract year, send same year to same reducer
- Result: Each reducer handles one year's data

MapReduce: Good vs Bad Use Cases

✅ Good for:

- Batch processing
- Large datasets (TB+)
- Embarrassingly parallel tasks
- Aggregations, counts, sums
- Log analysis
- ETL pipelines

❌ Bad for:

- Real-time queries
- Iterative algorithms
- Small datasets
- Interactive analysis
- Graph processing
- Machine learning

Rule of Thumb

If you need results in ¡ 1 minute, MapReduce is probably wrong choice

MapReduce Limitations

1. **High Latency**: Minutes to hours for jobs
2. **Disk I/O**: Intermediate results written to disk
3. **Only Two Phases**: Complex workflows need chaining
4. **No Iteration**: Each job reads from scratch

The Evolution

MapReduce limitations led to:

- **Spark**: In-memory processing, DAG execution
- **Flink**: True streaming
- **Hive**: SQL on MapReduce

We'll cover these in later weeks!

Key Takeaways

1. **MapReduce** = programming model for distributed processing
2. **Map**: Process records in parallel $\rightarrow$ emit (key, value)
3. **Shuffle**: Group by key (automatic, expensive)
4. **Reduce**: Aggregate values per key $\rightarrow$ final result
5. **Data locality**: Computation moves to data, not vice versa
6. **Combiner**: Local aggregation to reduce shuffle

Think in Key-Value Pairs!

Every MapReduce problem starts with:
"What should my key be? What should my value be?"

Next Session: Hands-On Practice

Week 3B: Implementing MapReduce in Python

- Implement mappers and reducers
- Complete crime analysis exercises
- Start Milestone 2

Pre-class preparation:

- Watch: Python MapReduce Tutorial
- Clone team repo and pull latest
- Review Chicago crime dataset