SE446: Big Data Engineering

Week 3B: Implementing MapReduce in Python

Prof. Anis Koubaa

Alfaisal University - College of Engineering

Spring 2026

Today's Agenda

Quick Recap

Input Data ⟶ MAP ⟶ SHUFFLE ⟶ REDUCE ⟶ Outp

Map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$
Process each record, emit pairs

Reduce: $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$
Aggregate all values for each key

Our Simple MapReduce Framework

```python
from collections import defaultdict

def map_reduce(data, mapper, reducer):
    """
    Simple MapReduce implementation in Python
    """
    # MAP PHASE
    mapped = []
    for record in data:
        result = mapper(record)
        if result:
            mapped.append(result)

    # SHUFFLE PHASE (group by key)
    shuffled = defaultdict(list)
    for key, value in mapped:
        shuffled[key].append(value)

    # REDUCE PHASE
    results = []
    for key, values in shuffled.items():
        result = reducer(key, values)
```

Using the Framework: Word Count

```python
# Sample data
documents = [
    "Hello World Hello",
    "World of Big Data",
    "Hello Big Data World"
]

# Flatten to words
words = [word for doc in documents for word in doc.split()]

# Mapper: each word -> (word, 1)
def word_mapper(word):
    return (word.lower(), 1)

# Reducer: sum all counts
def count_reducer(word, counts):
    return (word, sum(counts))
```

Loading Chicago Crime Data

```python
import pandas as pd

# Load crime data
url = "https://raw.githubusercontent.com/alfaisal-se446/data/main/
    chicago_crimes_sample.csv"
crimes = pd.read_csv(url)

print(f"Loaded {len(crimes)} crime records")
print(f"Columns: {list(crimes.columns)}")

# Preview
crimes.head()
```

Key Columns:

- Primary Type: THEFT, ASSAULT, BATTERY, etc.
- District: Police district number
- Arrest: True/False

Exercise 1: Count Crimes by Type

```python
# Convert DataFrame to list of dictionaries
crime_records = crimes.to_dict('records')

# TODO: Implement mapper
def crime_type_mapper(record):
    """
    Input: crime record dictionary
    Output: (crime_type, 1)
    """
    crime_type = record['Primary Type']
    return (crime_type, 1)

# TODO: Implement reducer
def count_reducer(crime_type, counts):
    """Sum all counts"""
    return (crime_type, sum(counts))

# Run MapReduce
```

Exercise 2: Count Crimes per District

```python
# Mapper: extract district
def district_mapper(record):
    district = record['District']
    return (district, 1)

# Same reducer (count)
results = map_reduce(crime_records, district_mapper, count_reducer)

# Sort by count (descending)
sorted_districts = sorted(results, key=lambda x: x[1], reverse=True)

print("Top 5 Districts by Crime Count:")
for district, count in sorted_districts[:5]:
    print(f"  District {district}: {count} crimes")
```

Exercise 3: Filter - Crimes with Arrests

```python
def arrest_mapper(record):
    """
    Only emit crimes where arrest was made
    """
    if record['Arrest'] == True:
        return (record['Primary Type'], 1)
    return None  # Filter out - no arrest

# Run MapReduce
arrest_counts = map_reduce(crime_records, arrest_mapper,
   count_reducer)

print("Crimes with Arrests:")
for crime_type, count in sorted(arrest_counts, key=lambda x: x[1],
   reverse=True)[:5]:
    print(f"  {crime_type}: {count} arrests")
```

Exercise 4: Aggregation - Arrest Rate by Type

```python
def arrest_stats_mapper(record):
    """Emit (crime_type, (arrested, total))"""
    crime_type = record['Primary Type']
    arrested = 1 if record['Arrest'] else 0
    return (crime_type, (arrested, 1))

def arrest_rate_reducer(crime_type, values):
    """Calculate arrest rate"""
    total_arrests = sum(v[0] for v in values)
    total_crimes = sum(v[1] for v in values)
    rate = (total_arrests / total_crimes) * 100
    return (crime_type, round(rate, 1))

results = map_reduce(crime_records, arrest_stats_mapper,
   arrest_rate_reducer)

print("Arrest Rates by Crime Type:")
for crime_type, rate in sorted(results, key=lambda x: x[1], reverse=
```

Chaining MapReduce Jobs

Problem: Find the top 5 crime types

Solution: Two MapReduce jobs

1. **Job 1**: Count crimes by type
   - Map: (record) → (type, 1)
   - Reduce: (type, [1,1,1...]) → (type, count)
2. **Job 2**: Find top 5
   - Map: (type, count) → (1, (type, count))
   - Reduce: (1, [(type, count)...]) → sorted top 5

Note

In real Hadoop, this means two separate jobs reading/writing to HDFS. In Spark, this is optimized with in-memory chaining.

Multi-Stage Example

```python
# Stage 1: Count by type
crime_counts = map_reduce(crime_records, crime_type_mapper, count_reducer)

# Stage 2: Find top 5
def top_mapper(item):
    """Send all to same reducer with dummy key"""
    crime_type, count = item
    return (1, (crime_type, count))  # Key=1 sends all to one reducer

def top_reducer(key, values):
    """Sort and take top 5"""
    sorted_values = sorted(values, key=lambda x: x[1], reverse=True)
    return (key, sorted_values[:5])

top_5 = map_reduce(crime_counts, top_mapper, top_reducer)

print("Top 5 Crime Types:")
for crime_type, count in top_5[0][1]:
    print(f"  {crime_type}: {count}")
```

MapReduce Design Patterns

Pattern	Map Output	Reduce Operation
Counting	(key, 1)	sum(values)
Sum	(key, value)	sum(values)
Average	(key, (value, 1))	sum(v)/sum(c)
Max/Min	(key, value)	max/min(values)
Filter	(key, value) or None	pass through
Distinct	(value, None)	emit key only
Inverted Index	(word, doc_id)	list of doc_ids

Pattern: Computing Average

```python
# Goal: Average crimes per district

# Mapper: emit (district, (count, 1))
def avg_mapper(record):
    return (record['District'], (1, 1))  # (crime_count, 1)

# Reducer: sum crimes, sum districts, divide
def avg_reducer(district, values):
    total_crimes = sum(v[0] for v in values)
    total_records = sum(v[1] for v in values)
    # Note: In this case, same as count since we emit 1 per record
    return (district, total_crimes)

# For true average (e.g., avg salary per department):
# mapper: (dept, (salary, 1))
# reducer: sum(salaries) / count
```

Common Mistakes

1. **Forgetting to handle None**
   - Mapper can return None to filter
   - Framework must check for None
2. **Wrong key choice**
   - Key determines grouping
   - Choose key based on what you want to aggregate
3. **Non-hashable keys**
   - Keys must be hashable (strings, numbers, tuples)
   - Lists, dicts cannot be keys
4. **Type mismatches**
   - Ensure values are consistent types
   - sum([1, 1, '1']) will fail!

Debugging Tips

```python
# Add debugging to mapper
def debug_mapper(record):
    result = crime_type_mapper(record)
    print(f"Mapper: {record['ID']} -> {result}")
    return result

# Test with small sample first
sample = crime_records[:5]
test_results = map_reduce(sample, debug_mapper, count_reducer)

# Verify shuffle output
def debug_map_reduce(data, mapper, reducer):
    # ... map phase ...
    print(f"After map: {len(mapped)} pairs")
    # ... shuffle phase ...
    print(f"After shuffle: {len(shuffled)} keys")
    for key in list(shuffled.keys())[:3]:
        print(f"  {key}: {len(shuffled[key])} values")
```

Lab 02: MapReduce Hands-On

Today's Lab Tasks:

1. Implement word count on a text file
2. Count crimes by type
3. Find district with most crimes
4. Calculate arrest rate by crime type
5. (Bonus) Find top 5 crime locations

Deliverables:

- Complete lab notebook
- Commit to team GitHub repo
- Answer ExamGPT questions

Summary

What You Learned Today

- Implement MapReduce in Python
- Apply mapper/reducer patterns to real data
- Chain multiple MapReduce stages
- Debug common MapReduce issues

Key Takeaway:

Think: **What is my key? What is my value?**

Next Week: Milestone 2 work session + MapReduce on Hadoop cluster