
UMD DATA605 - Big Data Systems

# 8.4: Map Reduce Algorithms

**Instructor**: Dr. GP Saggese, gsaggese@umd.edu

# MapReduce: Applications

- **Intuition**
  - Break massive jobs into independent map tasks
  - Aggregate via shuffle
  - Combine in reduce
- **Major classes of applications**
  - Text processing and search
    - E.g., tokenization, inverted index, log analysis
  - Large data transforms
    - E.g., ETL pipelines, joins, global sort, deduplication across petabytes
  - Data mining and machine learning
    - E.g., Counting co occurrences, feature extraction, $k$-means iterations
  - Graph and link analysis
    - E.g., PageRank, connected components
- **Typical outputs**, e.g.,
  - Counts
  - Aggregates
  - Reorganized datasets for downstream systems

# Cost Measures for Distributed Algorithms

- What matter is the real dollar cost not just $O(\cdot)$
  - **Total cost** $\approx$ CPU + storage + network
  - **Communication cost**
    - Total I/O across all processes, e.g., shuffling $1\,\mathrm{TB}$
  - **Elapsed communication cost**
    - Max I/O along the critical path
  - **Elapsed computation cost**
    - Wall clock with $p$ workers, sensitive to skew and stragglers
- **Dominant term heuristic**
  - *"If one cost dominates, ignore the others for first order reasoning"*
- **Practical note**
  - *"Adding more machines trades $ for time and may not fix skew"*

# Total Cost Model for MapReduce

- **Total cost**

$$C_{total} = C_{compute} + C_{io} + C_{network} + C_{storage}$$

- **Notation**
  - $|I|$ input GB, $|S|$ shuffle GB, $|O|$ output GB
  - $p_m$ mappers, $p_r$ reducers
  - $T_m$ map hours, $T_r$ reduce hours
  - $c_{vm}$ = \$/VM hour, $c_{io}$ = \$/GB I/O, $c_{sh}$ = \$/GB shuffle
- **Compute**

$$C_{compute} = c_{vm}\,(p_m T_m + p_r T_r)$$

  where $T_m$, $T_r$ include skew and stragglers
  - Skew effects = a heavy key or hotspot inflates the heaviest task and the critical path
- **I/O**

$$C_{io} = c_{io}\,(|I| + 2|S| + |O|)$$

  - Shuffle often dominates when $\sum_i |S_i| \gg |I|, |O|$

# Total Cost Model for MapReduce

- **Notation**
  - $|I|$ input GB, $|S|$ shuffle GB, $|O|$ output GB
  - $c_{sh}$ = \$/GB shuffle $c_{eg}$ = \$/GB egress, $c_{st}$ = \$/GB hour
  - $R$ HDFS replication
- **Network**

$$C_{network} = c_{sh}|S| + c_{eg}|O|_{eg}$$

  where $|O|_{eg}$ is data leaving the provider
- **Storage**

$$C_{stor} = c_{st} R(|I| + |O|)$$

- Plug in variables and unit prices to get $C_{total}$
- Apply the dominant term heuristic to prioritize optimization
- Tuning levers
  - Use Combiners
  - Compression
  - Better partitioning
  - Early filtering to reduce $\sum_i |S_i|$ and stragglers

# Inverted Index using MapReduce

- **Goal**: build a mapping from words to the list of documents they appear in
  - Map phase:
    - Input: `(docID, content)`
    - Emit: `(word, docID)` for each word in content
  - Reduce phase:
    - Input: `(word, [docID_1, docID_2, ...])`
    - Emit: `(word, list of unique docIDs)`
- Useful in search engines and information retrieval
- Requires tokenization and normalization of content
- Deduplication of document IDs in reducer

# Join Operations using MapReduce

- Goal: join two datasets based on a common key
- Types of joins:
  - Inner join, left/right outer join, full outer join
- Map phase:
  - Tag records by source and emit key as join attribute
- Reduce phase:
  - Input: (key, [records from both sources])
  - Perform join logic in reducer
- Example: joining employee records with department data by deptID
- Optimization strategies:
  - Replicated join for small datasets
  - Reduce-side join for large datasets
- Careful data tagging and partitioning required

# Sorting and Grouping in MapReduce

- Goal: organize data by keys or values for further analysis
- Map phase:
  - Emit data with key as sort/group criterion
- Shuffle and sort phase automatically sorts by key
- Reduce phase:
  - Receives sorted keys and can perform grouped aggregation
- Total sort:
  - Requires partitioning and sampling for global order
- Partial sort:
  - Sort within each partition
- Example: sort sales data by date or group by product ID
- Often used as a preprocessing step for reporting

# Graph Processing with MapReduce

- Many graph algorithms are iterative
- PageRank:
  - Compute importance score of web pages
- Map phase:
  - Emit contributions to neighboring nodes
- Reduce phase:
  - Sum contributions to update PageRank score
- Requires multiple MapReduce rounds for convergence
- Graph is represented as adjacency lists
- Each iteration refines the scores
- Can be implemented using chained MapReduce jobs or systems like Apache Giraph

# Statistical Aggregation and Log Analysis

- Statistical Aggregation:
  - Map: emit relevant quantities (e.g., value, 1 for count)
  - Reduce: compute sums, averages, variances
  - Example: $(\text{sensorID}, \text{temperature}) \rightarrow$ compute average temperature per sensor
- Log Analysis:
  - Map: parse logs, extract fields (timestamp, IP, status)
  - Emit: $(\text{key}, \text{value})$ for desired metrics
  - Reduce: aggregate (e.g., count errors, hits per IP)
  - Example: $(\text{status code}, 1) \rightarrow$ count occurrences
- Useful for monitoring, alerting, and trend analysis
- Can handle large-scale logs from distributed systems