CSCI 5105
Instructor: Abhishek Chandra

Today
- Data-intensive Computing
- MapReduce
  - Programming Model
  - System Model
- Other Data-intensive Computing Models

Data-Intensive Computing
- Big Data: Large quantities of data being generated
  - Commercial, social, scientific
  - E.g.: Google, Facebook, LHC, ...
- Goal: Analyze and compute on this data
- Problems:
  - Scale: PB’s of data, millions of files, 1000’s of nodes, millions of users
  - Cost: Using special purpose hardware may be too expensive
  - Reliability: Failures are common due to no. of machines

Common issues for Data-intensive Computing
- Data placement
  - How to partition the data across nodes
- Task scheduling
  - Where to execute computation tasks
- Fault tolerance
  - How to handle node/task failures
MapReduce
- Simple data-parallel programming model and framework
  - Designed for scalability and fault-tolerance
  - Uses commodity hardware clusters
- History
  - Pioneered by Google. E.g. application: Index construction for Google Search
  - Popularized by open-source Hadoop project

MapReduce Programming Model
- Data: Sequence of key-value records
- Map function: converts input key-value pairs to intermediate key-value pairs
  \((K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})\)
- Reduce function: converts intermediate key-value pairs to output key-value pairs
  \((K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})\)

Example: Word Count
mapper(filename, text) {
  foreach word in text.split():
    output(word, 1)
}

reducer(word, list(count)){
  output(word, sum(count))
}
Examples
- What would be MapReduce programs for the following?
- Grep:
  - Input: (Text files, word), Output: matching lines
- Reverse Web-link graph:
  - Input: (Web pages), Output: (<URL, source URLs>)
- Inverted Index:
  - Input: (Text documents), Output: (<word, documentIDs>)

MapReduce Stages
- Push: Input split into large chunks and placed on local disks of cluster nodes
- Map: Chunks are served to mapper processes
  - Prefer mapper that has data locally
  - Mappers save outputs to local disk before serving them to reducers
- Reduce: Reducers execute reduce tasks when map phase complete

Data Partitioning/Shuffling
- Goal: Divide intermediate key space across reducers
  - k reduce tasks => k partitions (simple hash fn)
  - E.g.: k=3, keys: {1,...,6} => partitions: {1,2}, {3,4}, {5,6}
- Shuffle: Send keys to the relevant reducers
  - All-to-all communication
- Combine: Local aggregation function for repeated keys produced by same map

MapReduce: System Components
- Distributed File System
  - Combines cluster’s local storage into a single namespace
  - Uses replication, provides locality information
    - E.g.: GFS, HDFS
- Cluster Manager (JobTracker)
  - Manages cluster resources and job scheduling
  - Schedules tasks near data
- Local Agent (TaskTracker)
  - Per-node agent
  - Manage tasks
Fault Tolerance

- Task re-execution: Retry task(s) on another node
  - On task or node failure
  - OK for a mapper?
  - OK for a reducer?
- Speculative execution: Launch copy of task on another node
  - To handle stragglers (slow tasks)
  - Use result from first task to finish

Other Data-intensive Computing Models

- Dryad: More general than Mapreduce
  - Job is a general DAG
  - Can use different communication mechanisms
- Spark: Distributed in-memory computation
  - Resilient Distributed Datasets (RDD) abstraction
  - Partitions data across the memory of multiple nodes
  - Well-suited for iterative processing

Other Data-intensive Computing Models (Contd.)

- Graph processing: Suited for iterative graph computations
  - Typically vertex-centric computations
  - Optimized for typical graph characteristics
- Stream computing: Operate on continuous data
  - Both input and output are data streams
  - DAG of operators applied on records as they come in