CSCI 5105

Instructor: Abhishek Chandra

Today

- Data-intensive computing
  - Mapreduce
  - Other 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

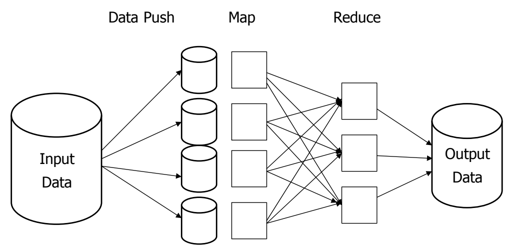
Data-intensive Computing: Issues

- 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

MapReduce Computation



MapReduce Programming Model

- Data: Sequence of key-value records

  list(K_{in}, V_{in})

- Map function: converts input key-value pairs to intermediate key-value pairs

  (K_{in}, V_{in}) ➔ list(K_{inter}, V_{inter})

- Reduce function: converts intermediate key-value pairs to output key-value pairs

  (K_{inter}, list(V_{inter})) ➔ list(K_{out}, V_{out})

Examples

- Wordcount:
  - Count the number of occurrences of each word in a set of text files
- Inverted Index:
  - Find the set of files containing each word

MapReduce Execution

- Map: Input data chunks processed by mappers
  - Mappers save outputs to local disk before serving them to reducers
- Shuffle: Send intermediate data to reducers
  - Intermediate key space partitioned across reducers
  - All-to-all communication
- Reduce: Execute reduce function on intermediate data
- Combine: Local aggregation function for repeated keys produced by same map

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

Resource Scheduling

- Each machine runs a certain number of mapper and reducer processes
- Locality-aware scheduling:
  - For each map task, prefer machine that has data locally
  - If not machine-local, then rack-local

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

- General computing frameworks:
  - Dryad
  - Spark
- Graph Processing
- Stream computing

13

Dryad

- More general than Mapreduce
- Job is a general DAG
  - Vertices: functions or operators
  - Edges: Dataflow
- Parallelism at each stage:
  - Each vertex can be replicated (data partitioning)
- Can use different communication mechanisms
  - Files, sockets, pipes, shared memory, etc.

14

Spark

- Distributed in-memory computation
  - Partitions data across the memory of multiple nodes
- Resilient Distributed Datasets (RDD) abstraction
  - Partitioned collection of records
  - Maintains data lineage: set of transformations applied to other datasets
- Well-suited for iterative and interactive processing

15

Graph Processing

- Suited for iterative graph computations
- GAS execution model
  - Gather, Apply, Scatter
  - Typically vertex-centric computations
  - Uses message passing abstraction
- Graph partitioning across machines
  - Could be optimized for typical graph characteristics
- Examples: Pregel, GraphLab

16

Stream Computing

- Operate on continuous data
- Both input and output are data streams
- DAG of operators applied on records as they come
- Latency is important metric
- Examples: Storm, Flink

17