CS744: Big Data Systems Notes

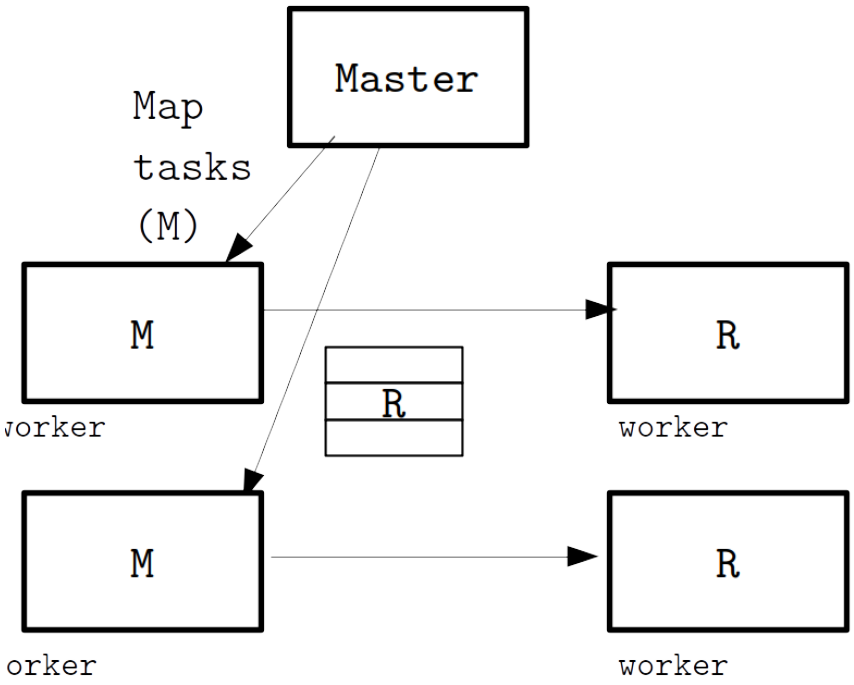
Jack Truskowski

February 18, 2019

Contents

1 MapReduce (2.4.19)

- Programming model
- Execution
- Runtime issues
- M-R library handles execution and run-time issues
  - Transparent to programmers



1.1 Operators

1. Map
   - Input = (key,value) –> (key, <v>)

2. Reduce
    - Operates share a key
    - (key,value) is sorted and values passed to reducer

1.2 Failures and Slowdowns

- Handled by the master

1.2.1 Possible failures

1. Map / Reduce
    - Worker fails, some maps and some reduces completed
    - Reduce data is already written to HDFS, doesn't need to be re-computed
    - Maps must be re-executed to recover intermediate data, since it hasn't been written to HDFS

2 Spark (2.6.19)

- Programming model

2.1 RDDs

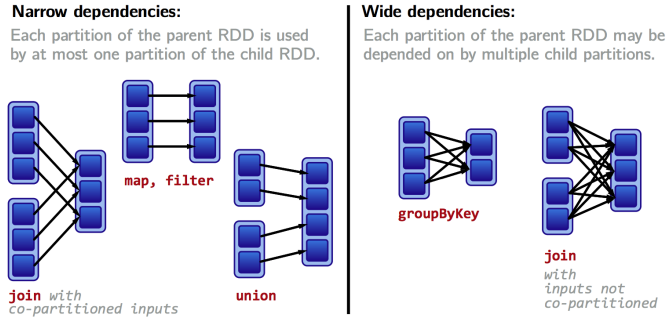
- Partitioned collection of records
- SQL, D-Streams, Graphx
- Intermediate data stored in memory
- Low overhead fault tolerance achieved through lineage

2.2 Benefits

1. Speed up iterative computations
2. Load datasets into memory
    - can't be done in MapReduce
3. Higher level programs

Verbatim RDD -> transformations -> action

- `Persist` (deserialized, serialized, on-disk)
  - RDDs only exist logically unless `persist` is called
    - Only then materialized (unless wide dependencies)



  - `REL` (reliable flag): checkpoint to disk or other memory locations
- Partitioning
- Lazy computation

2.3 Example: PageRank

- General process:
  1. Gather
  2. Applies
  3. Scatter

3 Spark + Job Manager (2.8.19)

3.1 Job Manager

- Scheduling: resources, ready, location
  1. How important is the task? (critical path) -> modeling, use some tasks to estimate others

3.2 Stragglers

1. Rate of progress -> progress report
    - Compare report with model
    - 50% slower -> this task is a straggler
    - Detect and react to stragglers early
        - No early detection of stragglers in `MapReduce`
2. Worthwhile to react to stragglers?
    - Does cloning lower overall resource use?

4 Cluster Schedulers – Mechanism (2.11.19)

4.1 Mechanisms

- YARN: one-level resource allocator
- Mesos: two-level pessimistic scheduler
- Omega/Borg: optimistic scheduler
- Fair allocation - delay scheduling

4.2 YARN

- Statistical-multiplexing
    - Work-conserving allocation discipline
        * No resources go to waste
- *Node Manager (NM)*: local available resources
- *Resource Manager (RM)*: global state
- *Application Master (AM)*: requesting for resources
- De-coupled from programming model
    - Gang scheduling, Message Passing Interface (MPI)
        * Revoking vs. admission control
- Late binding (kill-and-restart, e.g.)

4.3 Mesos

- Two-level scheduler
- Resource offers
  - Slaves report what resources are available
  - Mesos offers these resources to individual tasks
    - Frameworks can accept or deny offers (maybe they are not data-local)
- May kill tasks that run for too long to allow better sharing of resources

4.3.1 Weaknesses

- Applications cannot request specific resources, must wait until they get offered what they want

5 Omega Scheduler (2.13.19)

- HFS (delay scheduling)
- DRF: instantaneous fairness

1. all-or-nothing
2. priority

5.1 Dominant Resource Fairness

1. Slots -> rigid
2. Tasks are uniform
3. Equal number of slots is fair

6 (2.18.19)

- Resource variabilities (Clarenet, QOOP) | Batch analytics
- WAN bandwidth -> **Geo distributed analytics**
- Compute resources:

- Spot market
- Small clusters

6.1 Geo Distributed Analytics

6.1.1 Constraints

- Bandwidth varies over links (logical link)
- Control Plane
  - Scheduler
  - Job Manager
  - Parameter server
  - Partitioning/replication/coordination of the above
- Latency
  - Staleness
  - Heartbeats
- Legal & Privacy Issues
  - Multi-party computation

6.1.2 Clarinet

- Mutually-trusting data centers
- Batch computation – multiple queries
- Ignores legal & privacy issues
- Ignores partitioning scheduler or job manager

1. Big ideas
   (a) Query plans in a network-aware manner

   `Query => Query Optimizer (n/w aware) => Scheduler =>`

   - Placement of tasks
   - Scheduling, or "when"