Variability in Performance with Hadoop's Map/Reduce

Charles Reiss

CS262B

Motivation / Goal

- Anecdotal Observations: “I don’t know how long my [MR] job will take.”
- This project examines variation in the “easiest” case:
  - One job at a time
  - Mapper and reducer functions are trivial
- Variation observed should be the fault of the framework
- Important to scaling

Methodology

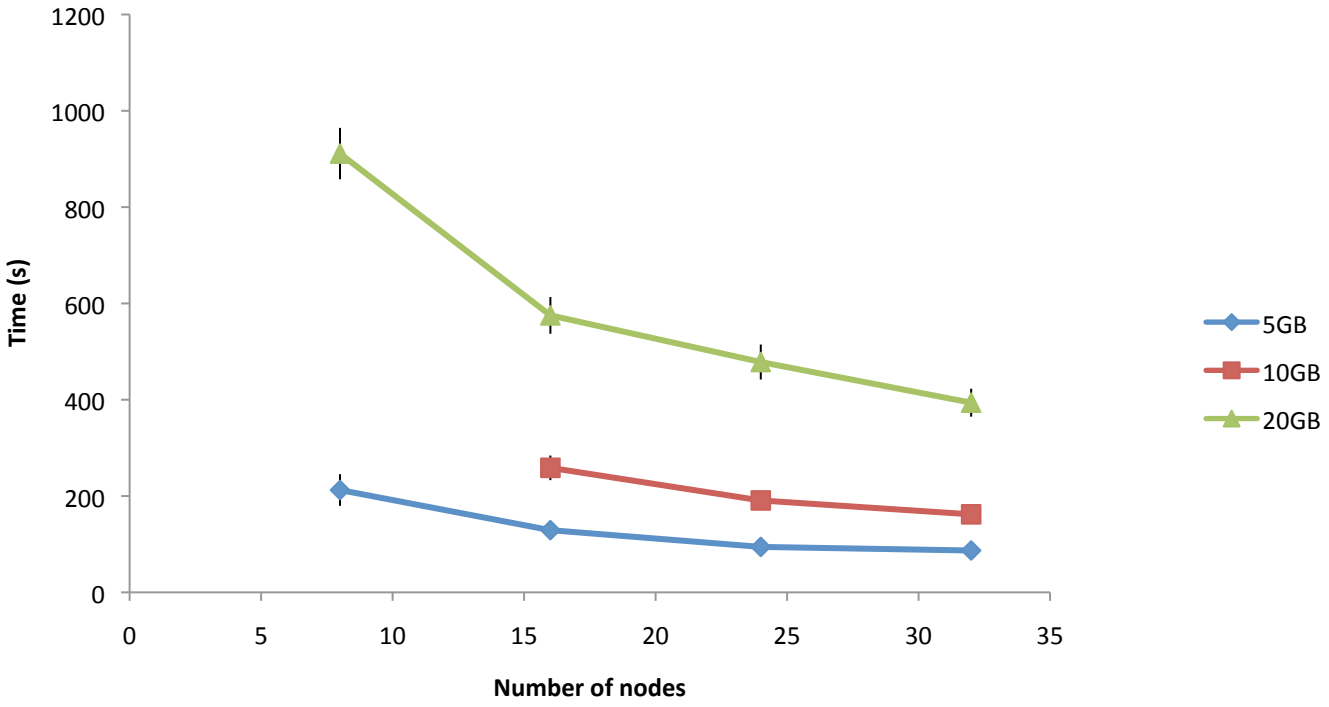
- Instrumented possibly skewed sort jobs:
  - Synthetic data (80B records)
    - Usually chosen from a uniform distribution
  - Mapper = identity function
  - Reducer = identity function
  - 1 map task per DFS block
  - 1 reduce task per map task
  - Partitioned by ranges based on a sample from the same uniform distribution

Measurements

- Already captured by Hadoop:
  - Job, task completion times and data sizes
  - Map task locality, reduce task location
- Added instrumentation to capture:
  - Map output (per reduce task) size, transfer time
  - HDFS read sizes, times
- Varying the environment to reduce jitter
  - One map/reduce slot/machine (2 cores) vs. default 2
  - One map output read per reduce task vs. default 4
  - HDFS on a RAM disk

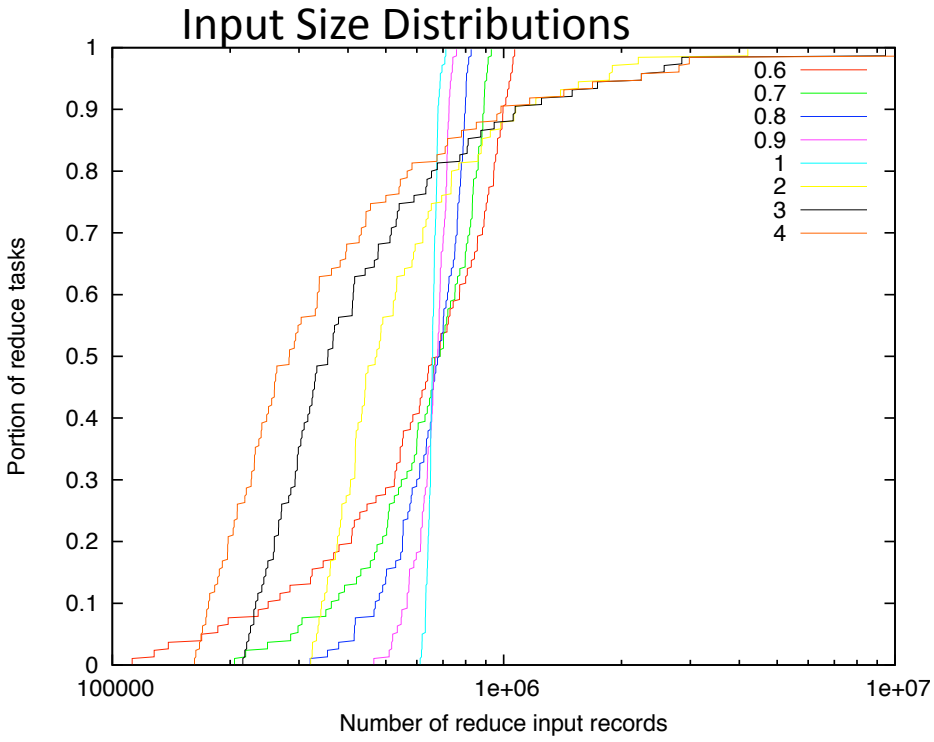
Overall Time Breakdown

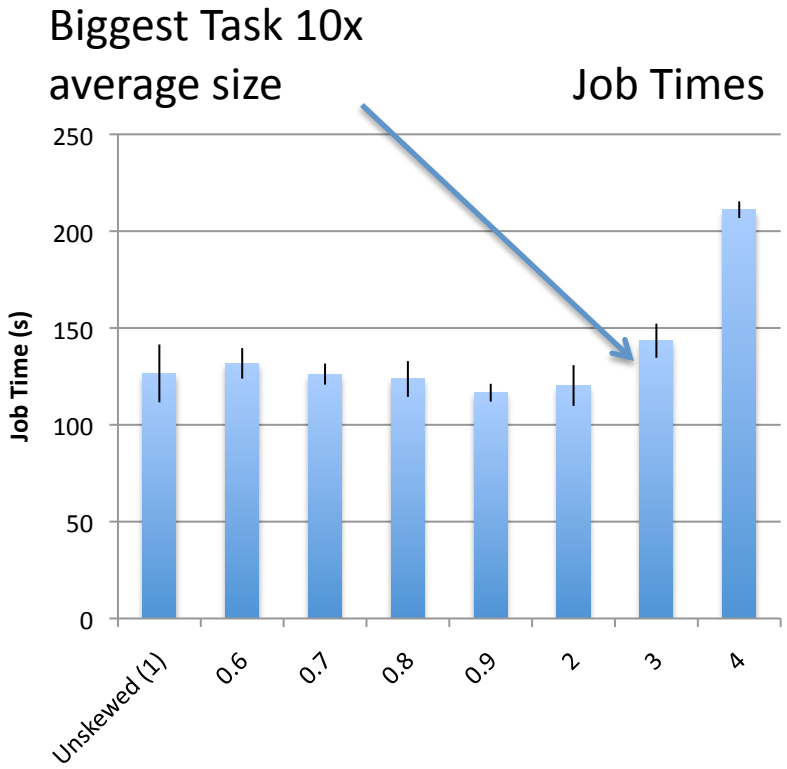
- Seems pretty good for well-balanced sort jobs



Reduce Task Balance

- With more than one wave of reduce tasks and little computation in them, only blatant imbalances have a significant effect…

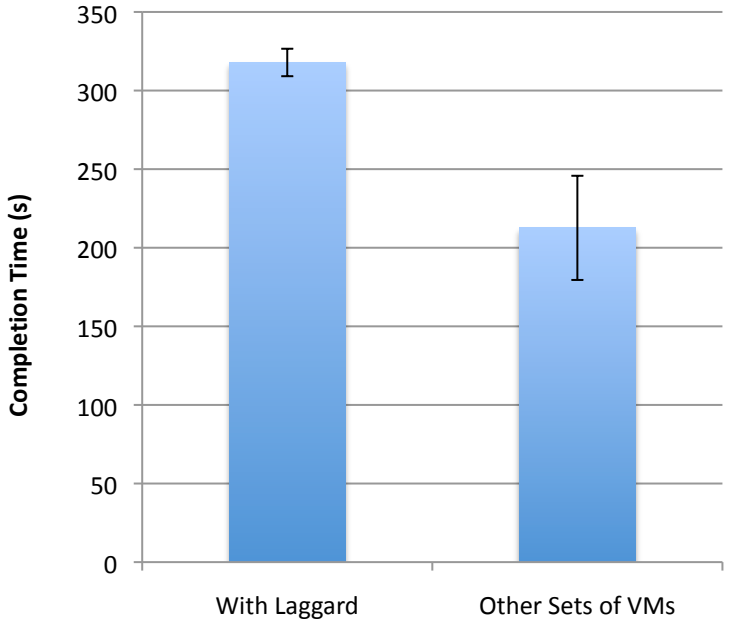




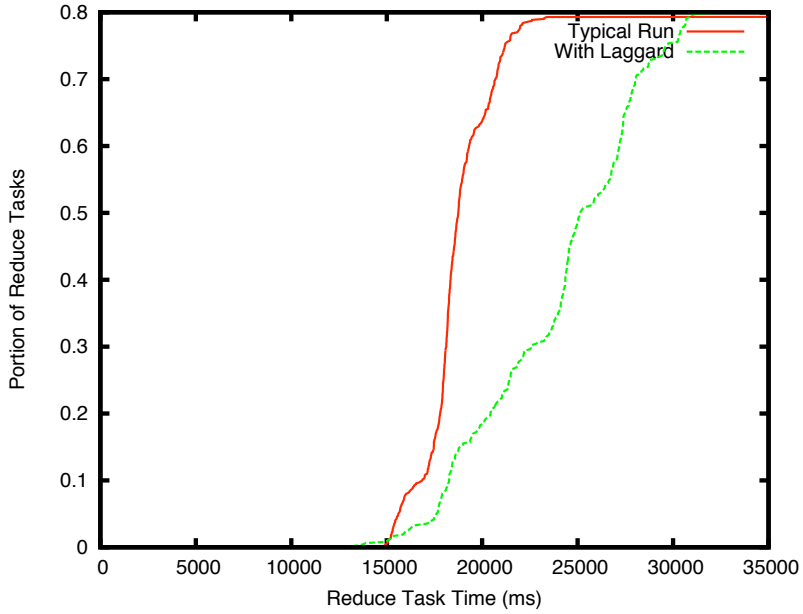
Map Output Laggards (Problem)

- Rarely on EC2 and more frequently with shared cluster machines, we noticed that some sets of machines do all jobs much slower:

Overall Job Time (with std. dev.)

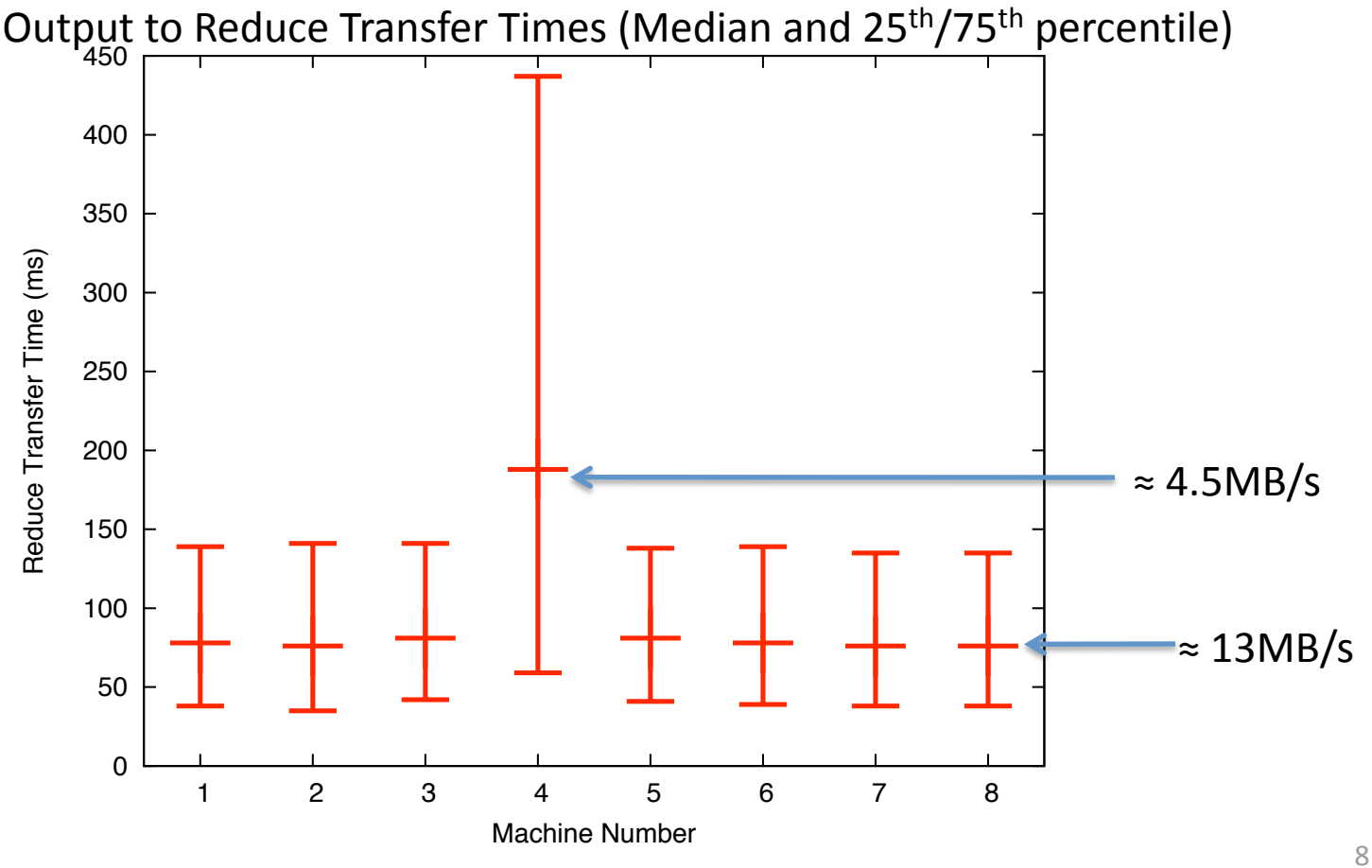


Reduce Task Times (CDF)



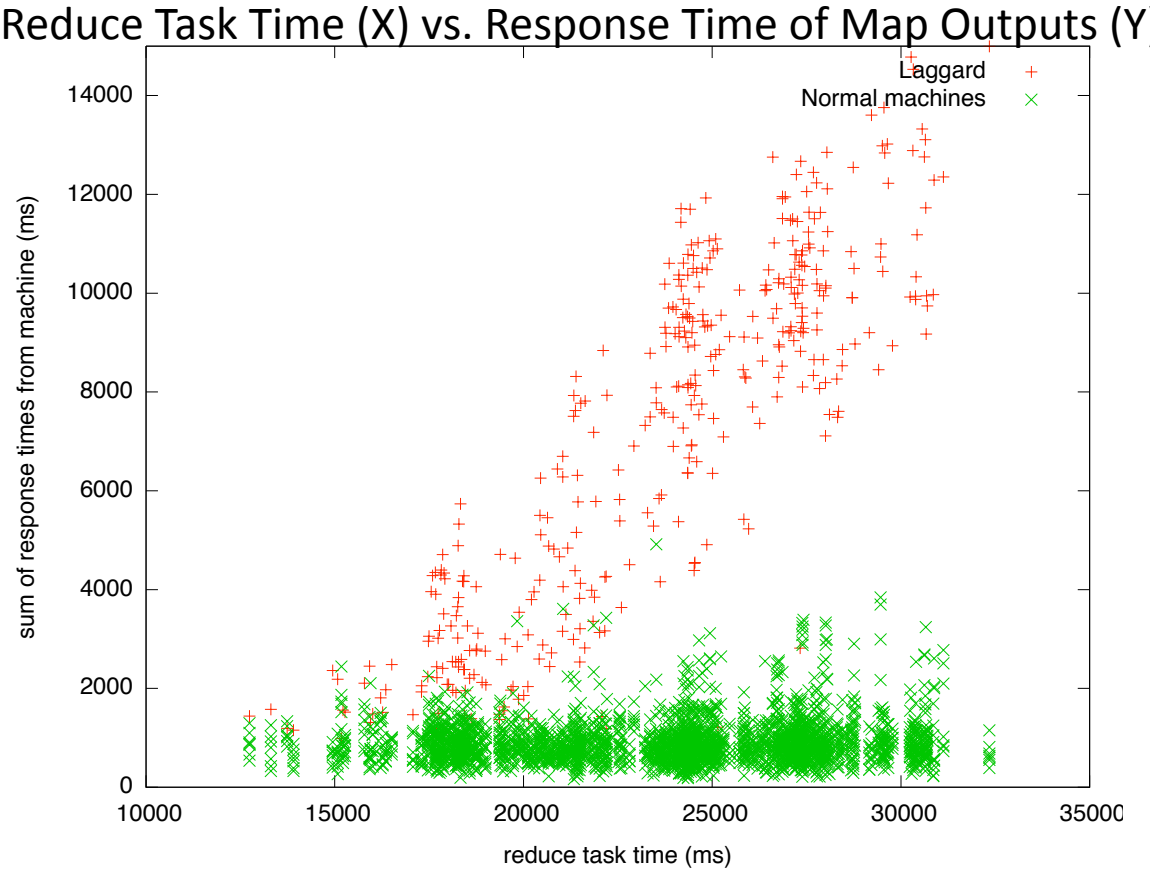
Map Output Laggards (Cause)

- Only one machine is actually slow:



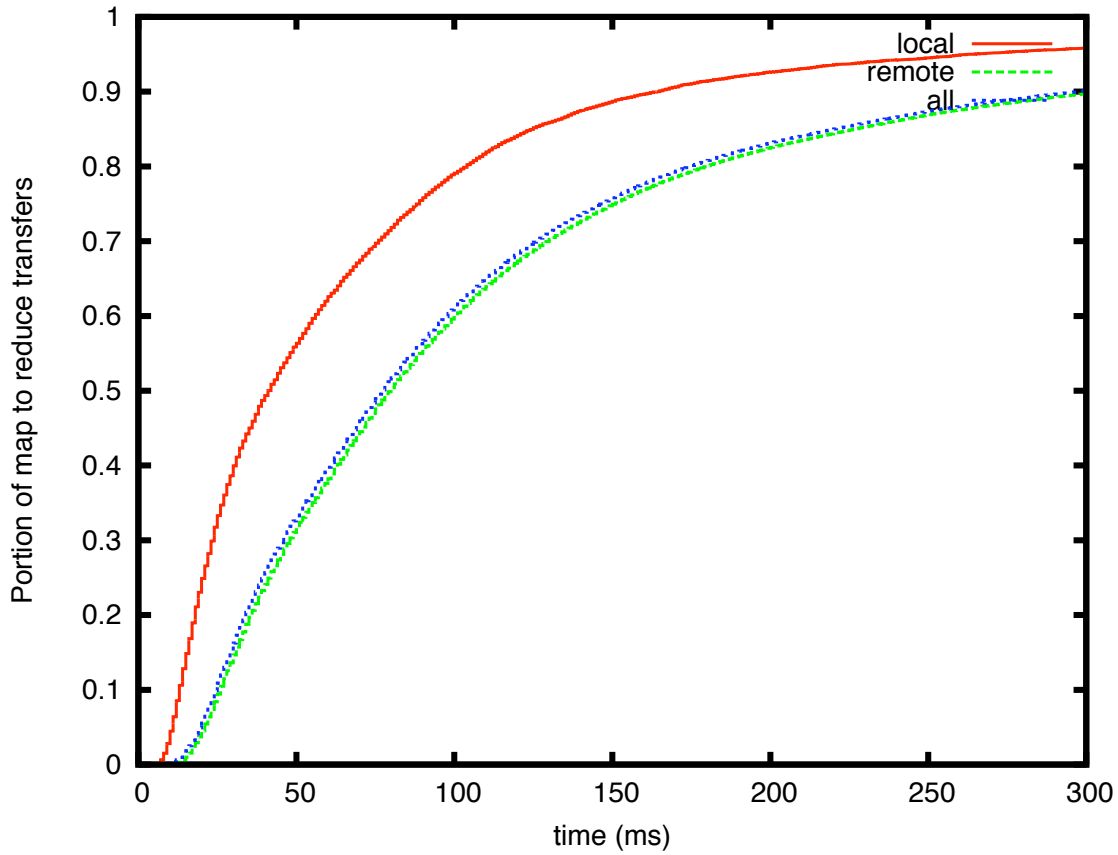
Map Output Laggards (Cause)

- …but that machine's response time determines how long almost every reduce task takes:



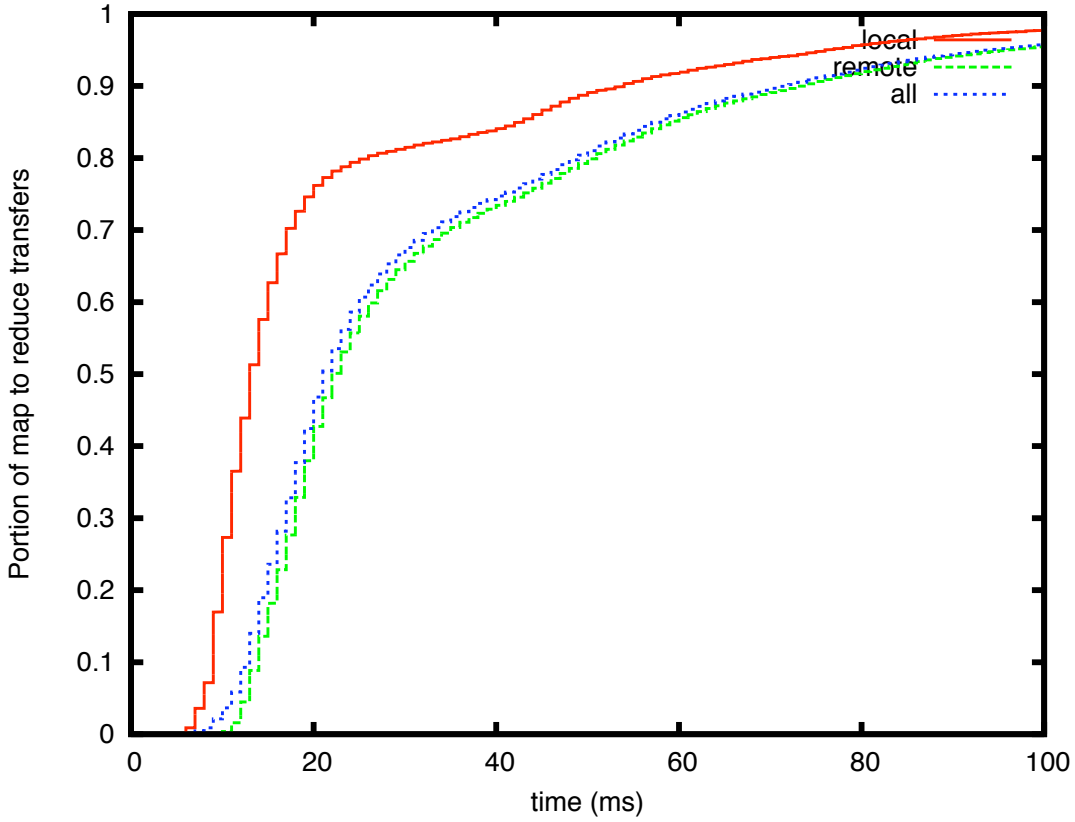
Map Outputs in the Normal Case

- Fetching map outputs in the normal case:



Map Outputs in the Normal Case

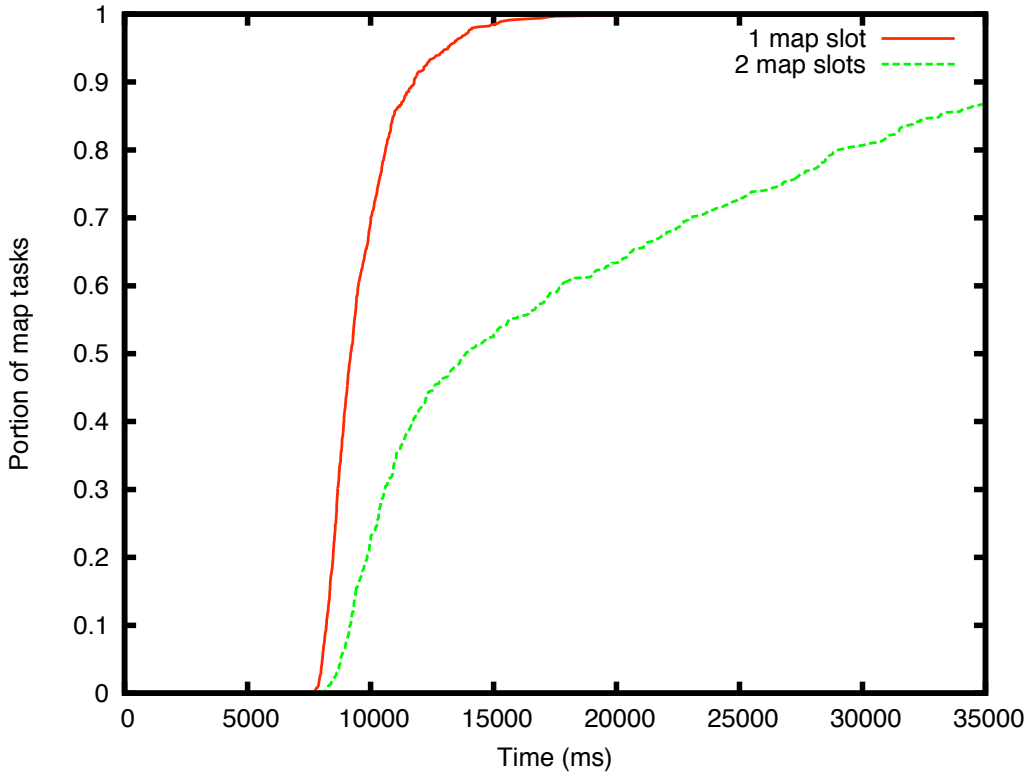
- Large tail not just from parallel fetches
- One fetch per machine:



Multiplexing Machines

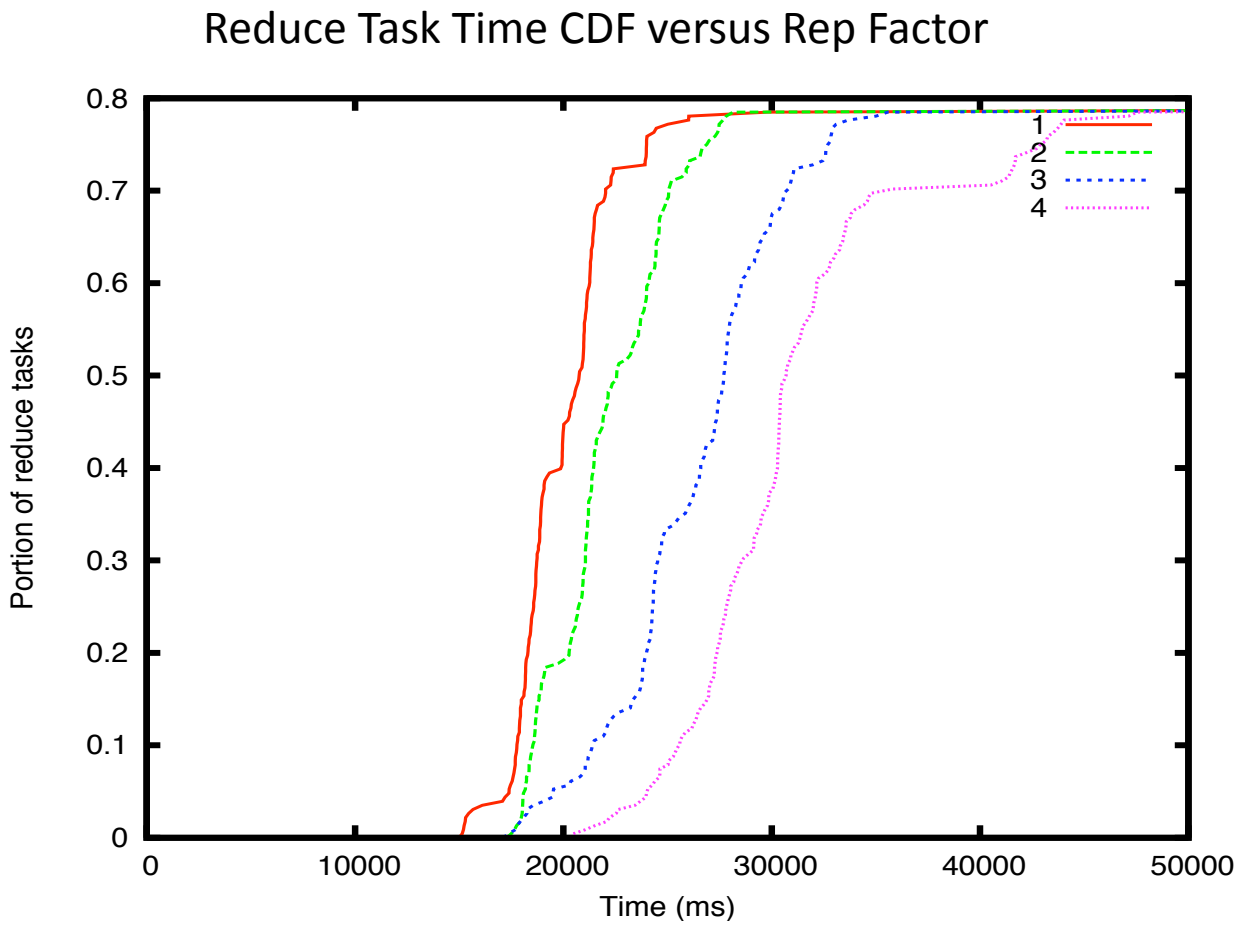
- Multiple map tasks on the same machine interfere substantially even with separate cores

CDF of Map Task Times (5GB sort job, 16 machines, 2 cores/VM)



Output Replication

- Output I/O not overlapped with computation with our trivial reduce function



Conclusions / Future Directions

- The Shuffle Step
  - Keep the pipeline full
    - Apply copy-phase splitting (see Matei's scheduling work)
    - Push instead of pull
  - Stop making lots of small data transfers
    - Consolidate map outputs destined for the same machine
      - For larger installations, consolidate per rack?
- Reduce skew
  - Presently, fixed overheads seem to dominate over load balance concerns
- Map task speeds
  - Future MR schedulers may need to account for the multiplexing penalty
- Future work
  - Cross-job/task interference
  - HDFS performance when a job is running

Questions?