Performance Variability in Hadoop's Map Reduce

Charles Reiss (Advisor: Randy Katz)

MOTIVATION / METHODOLOGY

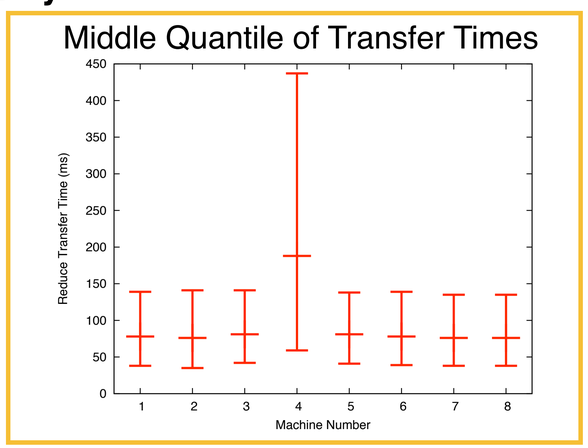
Goal: Characterize performance inconsistency in MapReduce frameworks that lead to poor interactive performance and scaling.

Data source:

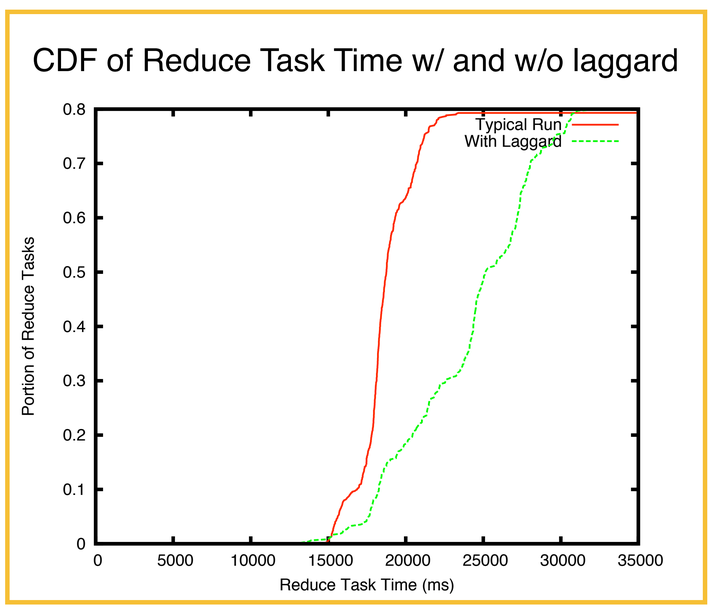
- **Sort jobs using identity map and reduce functions**
  - **Job is mostly framework overhead**

SHUFFLE STEP LAGGARDS

A single slow machine (observed "naturally" on EC2) does not send its map outputs quickly:

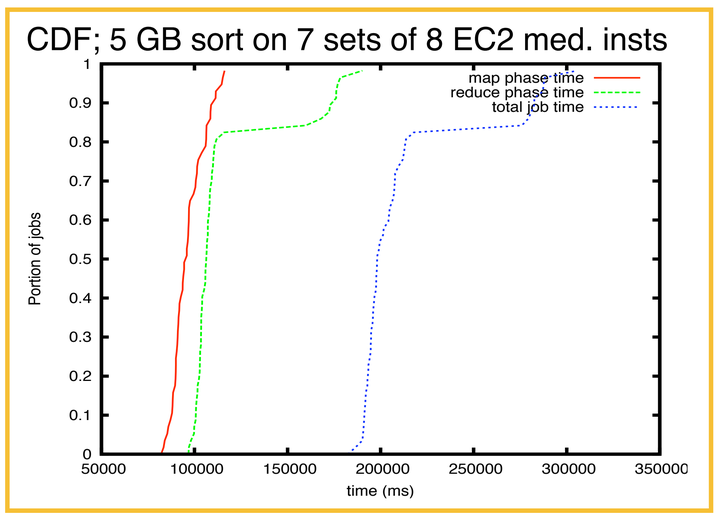


This makes *all* reduce tasks in the job slow:



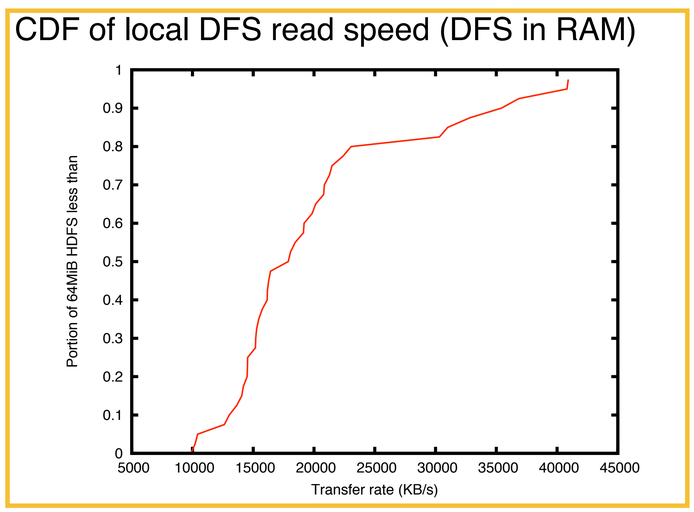
OVERALL JOB BREAKDOWN

Overall synthetic job time is relatively consistent (<10% change in completion time) on a particular set of nodes, except for the occasional set of runs with laggard nodes:



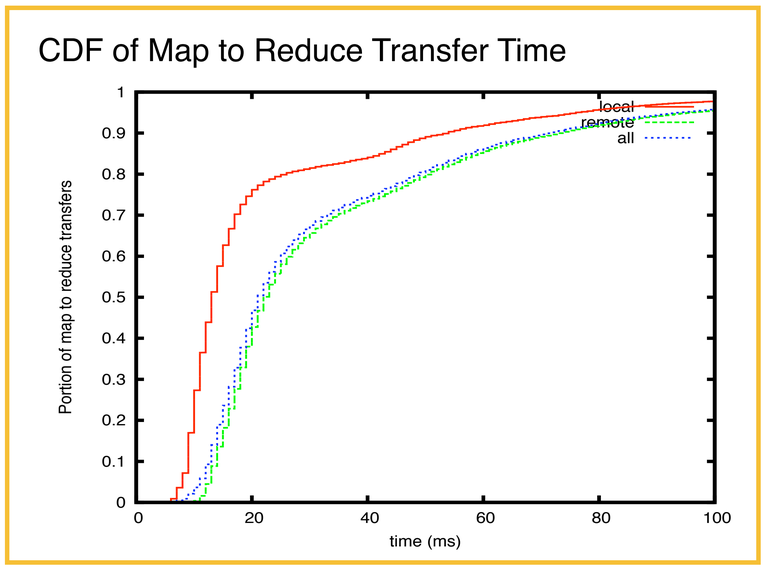
MAP READS

With a trivial mapper, reading the data from the distributed filesystem (DFS) dominated the map phase. This transfer is long enough to make setup overhead negligible but even local reads from a RAM disk-backed DFS are not consistent when the FS is loaded by the job:



SHUFFLE STEP CONNECITON OVERHEAD

Hadoop's reducers "pull" from each mapper. Even after accounting for locality and performing only one transfer at a time (instead of 8), the time taken to do this is inconsistent (for approx 900KB: median ~50ms, but 90th percentile ~300ms), despite consistent, small data sizes:



PROPOSED WORKAROUNDS

- **Make scheduler aware of time that will be necessary to pull map outputs when reducers are running**
- **Make scheduler account aware that task slots on the same machine aren't independent.**
- **Before running reduce tasks:**
  - **Consolidate map outputs so they can be transferred al at once;**
  - **Move map outputs to where reducers are expected to run**
- **Separate scheduling of data transfer part of tasks and data processing part of tasks**

OTHER FUTURE WORK

Empirical data:
-Analyze Yahoo! Traces to estimate real effects of these overheads, job configuration mistakes