Improving MapReduce Performance in Heterogeneous Environments

Matei Zaharia, Andy Konwinski, Anthony Joseph, Randy Katz, Ion Stoica

University of California at Berkeley
Motivation

1. MapReduce becoming popular
   – Open-source implementation, Hadoop, used by Yahoo!, Facebook, Last.fm, ...
   – Scale: 20 PB/day at Google, O(10,000) nodes at Yahoo, 3000 jobs/day at Facebook
2. Utility computing services like Amazon Elastic Compute Cloud (EC2) provide cheap on-demand computing
   - Price: 10 cents / VM / hour
   - Scale: thousands of VMs
   - Caveat: less control over performance
Results

• Main challenge for Hadoop on EC2 was performance heterogeneity, which breaks task scheduler assumptions

• Designed new LATE scheduler that can cut response time in half
Outline

1. MapReduce background
2. The problem of heterogeneity
3. LATE: a heterogeneity-aware scheduler
What is MapReduce?

- Programming model to split computations into independent parallel tasks
- Hides the complexity of fault tolerance
  - At 10,000’s of nodes, some will fail every day

1. Nodes fail $\rightarrow$ re-run tasks
2. Nodes slow (stragglers) $\rightarrow$ run backup tasks

How to do this in heterogeneous environment?
Heterogeneity in Virtualized Environments

• VM technology isolates CPU and memory, but disk and network are shared
  – Full bandwidth when no contention
  – Equal shares when there is contention

• \(2.5x\) performance difference
Backup Tasks in Hadoop’s Default Scheduler

- Start primary tasks, then look for backups to launch as nodes become free
- Tasks report “progress score” from 0 to 1
- Launch backup if progress < avgProgress – 0.2
Problems in Heterogeneous Environment

1. *Too many* backups, thrashing shared resources like network bandwidth
2. *Wrong* tasks backed up
3. Backups may be placed on *slow nodes*
4. Breaks when tasks start at different times

- Example: ~80% of reduces backed up, most losing to originals; network thrashed
Idea: Progress Rates

• Instead of using progress values, compute progress rates, and back up tasks that are “far enough” below the mean

• **Problem:** can still select the wrong tasks
Progress Rate Example

Node 1: 1 task/min
Node 2: 3x slower
Node 3: 1.9x slower
What if the job had 5 tasks?

Node 1

Node 2

time left: 1 min

Node 3

time left: 1.8 min

2 min

Node 2 is slowest, but should back up Node 3’s task!
Our Scheduler: LATE

• Insight: back up the task with the largest estimated finish time
  – “Longest Approximate Time to End”
  – Look forward instead of looking backward

• Sanity thresholds:
  – Cap number of backup tasks
  – Launch backups on fast nodes
  – Only back up tasks that are sufficiently slow
LATE Details

• Estimating finish times:

\[
\text{progress rate} = \frac{\text{progress score}}{\text{execution time}}
\]

\[
\text{estimated time left} = \frac{1 - \text{progress score}}{\text{progress rate}}
\]

• Threshold values:
  – 10% cap on backups, 25\textsuperscript{th} percentiles for slow node/task
  – Validated by sensitivity analysis
LATE Example

![Diagram showing progress and time for nodes]

- **Node 1**: Progress = 5.3%
- **Node 2**: Progress = 66%
- **Node 3**:...

Estimated time left:
- Node 1: \( \frac{1-0.66}{\frac{1}{3}} = 1 \) min
- Node 2: \( \frac{1-0.05}{\frac{1}{1.9}} = 1.8 \) min

LATE correctly picks Node 3.
Evaluation

• Environments:
  – EC2 (3 job types, 200-250 nodes)
  – Small local testbed
• Self-contention through VM placement
• Stragglers through background processes
EC2 Sort with Stragglers

- Average 58% speedup over native, 220% over no backups
- 93% max speedup over native
EC2 Sort without Stragglers

- Average 27% speedup over native, 31% over no backups
Conclusion

• Heterogeneity is a challenge for parallel apps, and is growing more important

• Lessons:
  – Back up tasks which hurt response time most
  – Mind shared resources

• 2x improvement using simple algorithm
EC2 Grep and Wordcount

Grep

- 36% gain over native
- 57% gain over no backups

WordCount

- 8.5% gain over native
- 179% gain over no backups