A. Arpaci-Dusseau **Department of Computer Science**
CS739: Distributed Systems **University of Wisconsin, Madison**

MapReduce: Simplified Data Processing on Large Clusters - OSDI'04

1 Introduction

- What is the motivation for this work? (What is their computing environment? What does their programming model hide? What don't they hide,and what is the advantage of this?)

2 Programming Model

- At a high level, what does their programming model look like? What type of applications can be easily expressed as MapReduce computations?

3 Implementation

- What are the execution steps when starting up and running a MapReduce application?
- What does the value of M influence? (What are the tensions between making M too large or too small?) What is the purpose of the partition function? How should one choose R? Is it necessary to sort the data before passing it to the reducers? What is the final state of the system?
- What is their system robust to? What are the pros and cons of using a single master?
- What happens if a worker fails? What tasks need to be restarted? What would be the implication of using GFS instead of local disks for intermediate results? What guarantees can they make in the presence of failures? (What do they do if there is a deterministic failure caused by the data?)
- What techniques do they use to help performance?
- How do they ensure each worker has roughly the same amount of work?

4 Performance

- What is the scale of the system used in these experiments? For grep, how is the work split between mappers and reducers? Does anything stand out regarding performance? What is causing this problem?

- How does grep differ from the average job stats shown in Table 1? Does having a single master seem to be a good decision?
- How is the sort constructed? What code can run concurrently? (see Figure 3) How is the MapReduce sort different than a sort that is optimized for *best-case* performance in a cluster (e.g., NOWSort)? What are the relative advantages of each approach?

5 Conclusions

- Very difficult to run parallel, I/O-intensive applications: have to worry about failure, stragglers, locality, and load balancing. MapReduce addresses those concerns in very reasonable manner by restricting the programming model and exposing the need to keep the computation near the data. Created a very usuable system for 1000 nodes.
- Do you want experience using Hadoop (open-source map-reduce)?

Improving MapReduce Performance in Heterogeneous Environments – OSDI'08

- What class of MapReduce jobs are they focusing on? What specific aspect of the system are they improving?
- In the Hadoop implementation of MapReduce, why is speculative execution performed? How do they identify stragglers?
- The authors identify 6 assumptions of the Hadoop scheduler. How do each of those assumptions impact the Hadoop scheduler?
- The LATE scheduler speculatively executes the task they estimate will finish farthest into the future; what fundamental assumption does this make?
- How do the authors estimate time left in the LATE scheduler? What is the purpose of SpeculativeCap? Of SlowNodeThreshold? Of SlowTaskThreshold?
- How do you think the authors should have evaluated the LATE scheduler?