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

The Google File System - SOSP'03

1 Introduction

- What is the motivation for this work? What are their assumptions? What do you think is most impressive about their goals?

2 Design Overview

- What is the overall architecture of their system? Does GFS provide functional homogeneity? Is a master the right design decision?
- What are the interactions between the nodes on a read operation? Is the master likely to be a bottleneck for reads?
- What data structures are kept on the master? Which data structures are kept persistently? What data is not persistent? How does the master get this information about a crash? What are the pros and cons of keeping all of that meta-data in main memory? (What is the size of the meta-data on the master??)
- GFS makes the design decision to not explicitly cache files on either the clients or the chunkservers. Does this seem like a good decision?
- GFS makes the design decision to use fixed sized chunks of 64 MB? What factors argue for large chunk sizes? What factors argue for small chunk sizes? How does 64 MB interact with Map-Reduce applications? Does 64 MB seem reasonable?

3 Consistency Model

- GFS specializes its consistency model to its application domain. To understand Table 1: What does it mean for replicas to be *consistent*? What does it mean to be *defined*? What is the difference between a *write*? and a *record append*? How do the different states occur? Why or why not are all of these states acceptable?
- Is it ever possible for a client to read an inconsistent (i.e., stale) replica? Do you think this is acceptable?

4 System Interactions

- What happens when a client wants to do a write? Why is it helpful to have a primary? Is the use of leases appropriate here? How does the replica-update protocol achieve decent performance while ensuring that replicas are kept consistent?
- Why might the write protocol lead to inconsistent regions? why undefined regions?
- How is the protocol for record appends different than ordinary writes? (Why must the the primary sometimes pad the previous chunk?) Why might this protocol lead to some inconsistent entries? How do applications deal with this model?

5 Master Operation

- How does the master organize the file namespace? What is the advantage of their approach compared to a traditional Unix directory structure?
- *Where* replicas are placed is an important factor for both reliability and performance. What is the GFS policy for placing replicas?
- What happens when a file is deleted? How is the physical space on disk actually freed? Do you think this is a better approach than having the master explicitly tell the chunkservers to delete the space?
- What is the role of *chunk version numbers* in the protocol?

6 Fault Tolerance and Diagnosis

- As discussed so far, what is the single point of failure in the system? How do they improve availability in GFS?
- How do they address the concern of data integrity? Given that they have multiple replicas, why don't they just compare the data across replicas and vote? Is their approach ever inefficient?

7 Measurements

- What do you think of their throughput results in Figure 3?
- What do you think about the performance they report in Table 3?
- What do you think of the data presented in Tables 4 and 5?

8 Conclusions

- Contributions: Handling node failures so well (making location of chunks soft state and using checksums for all data), pushing some complexity into map reduce framework (tuning to application semantics with appends), simplifying system to use a single master that can handle all metadata in memory.