MOTIVATION

1. Description of Google’s environment

2. Your worst nightmare is real: you are trapped in an endless cycle of 6.033 design projects. The latest is number 64. Even worse, it’s due in five minutes.
   --See the blackboard for the problem.
   --Assume you are operating in the Google environment described above.
   --Do not use MapReduce as you think about this problem.

UNDERSTANDING MAPREDUCE

3. Overview of MapReduce computational model

   --Are there computations that can’t be conveniently expressed in this model?

4. Overview of MapReduce implementation

   --Why does the Map worker write its data to the local disk?

5. Fault tolerance

   --How does MapReduce tolerate worker faults?
   --What about master faults?

6. Performance

   --What has the biggest effect on performance?
   --How did the authors solve that problem?
   --How did they know it would be a problem?

REFLECTION

7. In Chapter 6, we hear that often it’s easier to solve a performance problem by waiting until surrounding technology (processors, disks, memory, other software) improves than it is to be clever with one’s own problem. Will MapReduce be rendered obsolete in the future by faster, bigger, better computers?

8. This system appears to be a pragmatic response to a real problem. Let’s review some of the ways pragmatism trumps idealism in MapReduce.

NOTE: Concluding thoughts on reverse....
CONCLUDING THOUGHTS

[To some extent, these thoughts are opinion; we may disagree.]

MapReduce is a great example of the following:

A. Layering, modularity, and abstraction
   --MapReduce factors out a common, useful function
   --MapReduce breaks a complex problem into manageable sub-pieces for *both*:
     --Users of MapReduce
       --The authors of MapReduce (the map worker, the reduce worker, etc. are relatively simple to *implement*)
     --MapReduce lets programmers who have no particular experience dealing with a difficult environment -- namely, vast amounts of data, lots of parallelism, hardware and software faults, distributed computations -- nonetheless operate successfully in that environment

B. Performance tuning
   --Authors do not over-optimize but do solve their problem

C. Designing with a goal in mind
   --Solution is quite tailored to problem
   --Authors display what I would call "well-considered pragmatism": some corners cut, but those seem to be the right corners