Recitation 16 — MapReduce

Overview

- Distributed system that supports a specific computation model: mapping and reducing.
    - View a document as a set of <key, value> pairs
    - Process those pairs with some map function to get intermediate values, which are then processed with some reduce function.
- There are often multiple ways to apply this model to a single problem.

Basic functionality

Figure 1 in the paper is a good one for understanding this

- Controller ("master" in paper) has an organizational/control role, like in GFS
- Controller assigns map tasks to workers, considering the locality of data in its assignments
    - The input data is replicated by GFS
- Workers map, write intermediate data to files
- As mappers finish, controller assigns reduce jobs, tells reduce workers where to access the intermediate data

Handling failure

- GFS is in place so that if a machine fails, input data is still available
- Partial executions of map and reduce are fine; controller can reassign the task
    - Controller pings workers to detect failures (essentially just like how the HELLO protocol for routing works)
    - Being able to re-execute tasks also helps with performance: can just re-execute the tasks of workers who are straggling.
- Paper claims that controller failure is unlikely (the argument would be that, while the failure of *some* machine is high, the failure of a *specific* machine — the controller — is low)

Discussion

- Is MapReduce simple?
- What trade-offs do we make in this system?
- What *can't* MapReduce be used for?