Design Document

Dependencies

- libthrift 0.9.3 + transitives for rpc
- slf4j 1.7.12 for logging
- metrics-core 3.1.0 for logging
- rxjava 1.1.3 for frp and async

Java 8

This uses Java 8 for reducing boilerplate. Since Java 8 isn't default on lab machines, in my Makefile I have it set up to find the Java 8 compiler and runtime. It queries OpenDNS for the machine's external IP, converts it to a hostname, and checks if it ends in 'cselabs.umn.edu.'.

Thrift

I use Thrift to generate three sets of files: the addresses, the storage service, and the coordinator service. The addresses are shared between both services. All interprocess and some intraprocess communication is done with Thrift generated structs and exceptions.

Metrics

I use metrics-core from Dropwizard to measure how long a read or write request takes. It generates stats on the timers and writes them to a csv file.

RxJava

I use RxJava to do functional reactive programming. This heps me get the logic and error handling correct for complex tasks (like sending concurrent tasks to multiple nodes and logging and retrying errors).

Common

ThriftClient

Thrift requires a fair amount of boilerplate on the client side to make a transport, a protocol, a client, and then close them. It also throws the checked exception TException on everything. So ThriftClient and ThriftFunction handle the boilerplate for every use case (including retries). I can pass a lambda of just the work I need done.

Data

Data is the class that encapsulates access to the `work/input`, `work/intermediate`, and `work/output` folders.

Metrics

Metrics encapsulates setting up and reporting metrics.

Master

Master code is in `mapreduce.master`. It runs a thrift service called MasterService. It has 5 classes:

- Master is the main class that launches a MasterServer and WorkerPool.
- MasterParameters parses command line arguments.
- MasterServer binds the socket and launches a thrift service.
- MasterHandler implements the thrift service.
    - It sets up clean meters for a each job.
    - It splits the data into chunks.
    - It sends sort requests to nodes from the worker pool.
    - Every time it gets enough chunks sorted or partially merged, it will send another merge request out till everything is fully merged.
    - Once it has a fully merged chunk, it will expand it from the condensed form.
    - For all sort and merge requests, it handles proactive failure handling by sending the requests to multiple workers, and retrying any requests that fail.
- WorkerPool keeps a list of workers and whether they're alive. It will heartbeat every worker each second, and update their liveness. A worker is considered alive if it responded to the previous heartbeat. The pool will return a random living worker when asked.

Worker

Worker code is in `mapreduce.worker`. It runs a thrift service called WorkerService and has 4 classes:

- Worker is the main class that launches a WorkerServer and metrics reporting thread.
- WorkerParameters parses command line arguments.
- WorkerServer binds the socket and launches a thrift service.
- WorkerHandler implements the thrift service.
    - It responds to heartbeat checks.
    - It times how long each sort and merge takes.
    - It randomly (according to parameter) will throw a TaskFailException on some requests.
    - When sorting, instead of writing back a full sorted version of the input like `0 0 2 3 3 3 4 ...`, it will write a condensed version `2 0 1 3 1 ...`. This saves on disk space (though it doesn't solve all the quota issues.
    - Also when merging, it takes in and produces condensed version files. The master expands it back to the output at the end of the mergesort.

Client

- Client is the application. It reads input line by line and passes it to the Commander.
- Commander is the class that runs various commands, like checking on storages, getting info, and doing reads and writes.

Orchestrator

If I may be so boastful, Orchestrator is a *brilliant* piece of engineering.
It doesn't use Thrift. What it does is parse a config of servers desired, and then launch those servers as desired. It's a mix between a CM system such as Ansible and a server manager like Terraform, specialized to this assignment.

It ensures that every server is shut down, every process waited on, and every port is unbound. And to top it all off, it grabs the logs from all servers and downloads them to `logs/`.