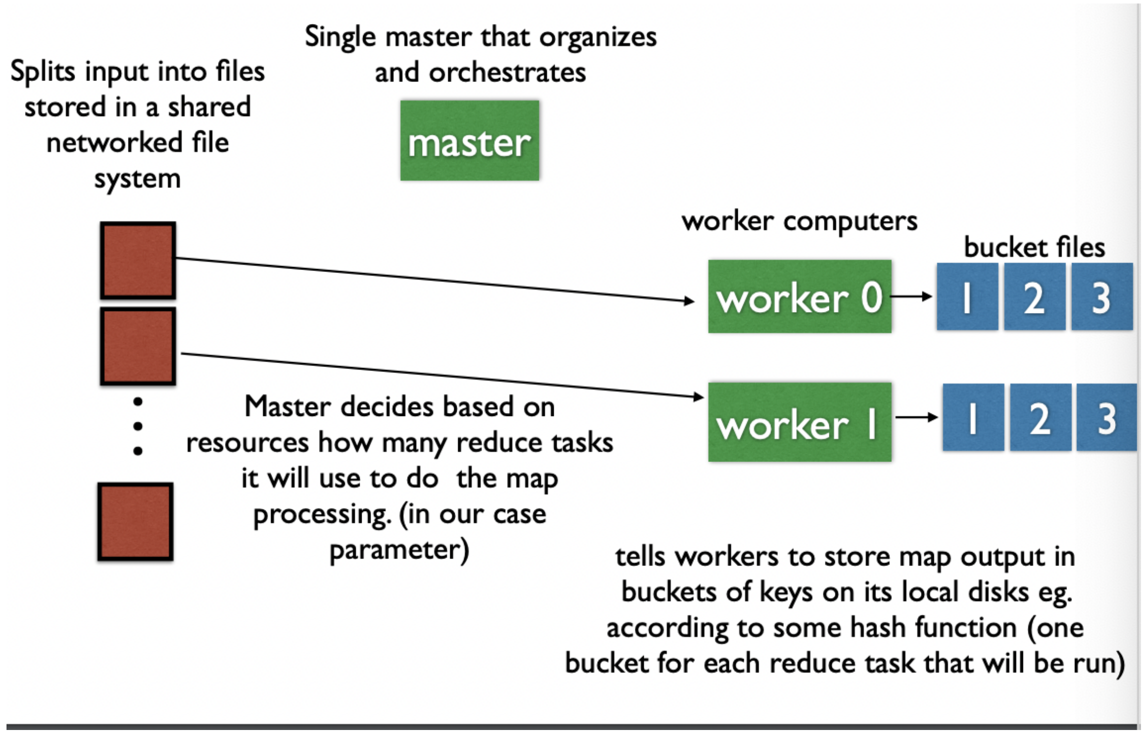
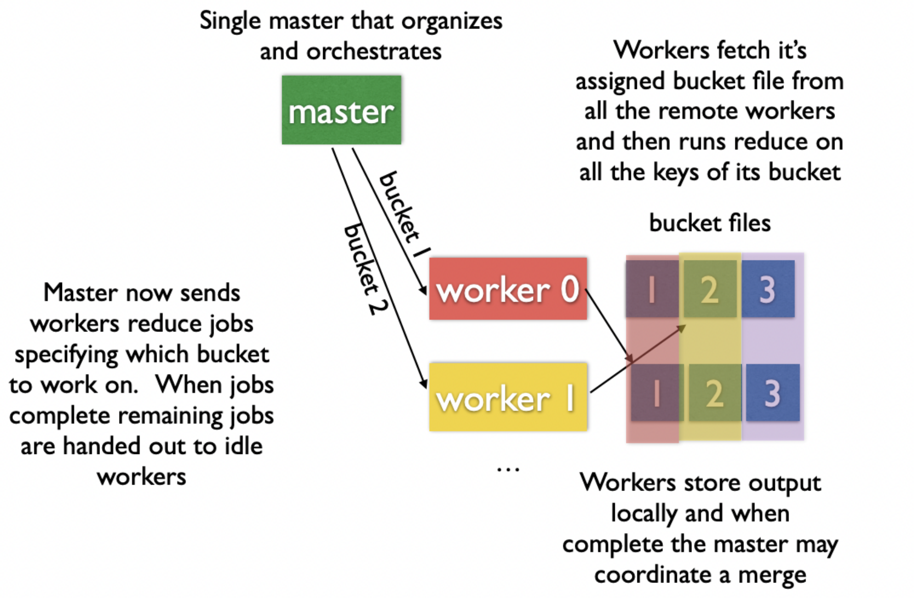
CAS CS 350
InClass Note 3

MapReduce

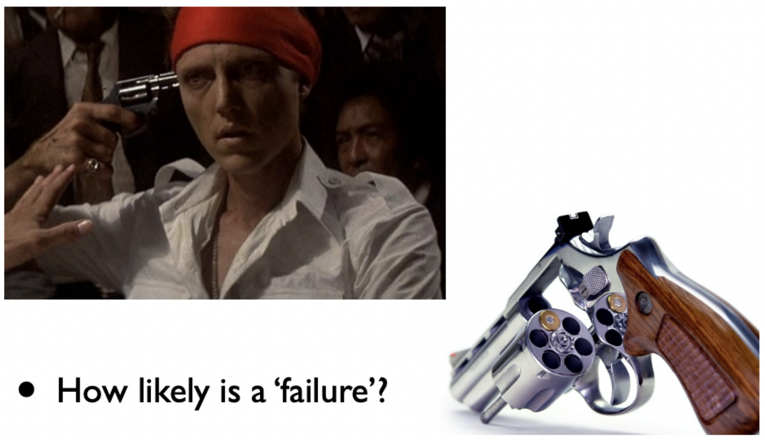
1. Overview
    a. MapReduce → programming model and a system that enables users to paralyze communications
    b. It is useful
        - with respect to other systems, it was more efficient when it was first published
        - It handles failures transparently (programmer does not need to bother about failure and can focus on the logical program which increases productivity)
        - Very simple API (easy to learn) → just implement two functions (Map and Reduce)
    c. Between the map and reduce phase, there is a logical barrier (reduce function only works when all map functions are completed) since reduce function needs to include all the keys within the map function → for correctness
2. MapReduce: Implementation
    a. 
Splits input into files stored in a shared networked file system
Single master that organizes and orchestrates
master
worker computers
bucket files
worker 0
1
2
3
worker 1
1
2
3
Master decides based on resources how many reduce tasks it will use to do the map processing. (in our case parameter)
tells workers to store map output in buckets of keys on its local disks eg. according to some hash function (one bucket for each reduce task that will be run)
    b. The nodes in the mapReduce are called workers
    c. We can use resources for both map and reduce phase
    d. Ex) 4 machines → 4 slots (use all machine)

e. Special node (master or coordinator) → schedules to tasks to workers and knows everything (knows where the workers are, which workers are busy or not, where the data are)
f. When jobs start, the input files are stored in distributed file system (in this case, Google System), and intermediate data are stored on local disks of the mapper
g. Coordinator always tries to assign map tasks in the same machine that data exists since if you work in other machine, it requires movement of data
h. Coordinator also knows the users → it tells the mappers how many buckets (blue files) the mappers need to produce
i. Blue files are partitions of the data
j. Intermediate key value pairs are partition by key using hash function (function that takes string of arbitrary length and produces string of fixed length, the value of the hash function should be same but not necessarily unique)
k. When partition key value pairs with only three bucket files → split into three parts → Apply hash function to the input key and take modulo of the output of the hash function (integer in this case)
l. All workers should use the same hash function to partition the keys (because the same key in worker 0 and worker 1 should be placed in the same bucket files) Ex) If key A is stored in bucket file 2 in worker 0, key A should be stored in bucket file 2 in worker 1 → possible by the usage of the same hash function
m. All partition (all buckets in the same number) will be processed by the same reducer in the next phase

n. 
Single master that organizes and orchestrates
master
bucket 1
bucket 2
worker 0
worker 1
...
Workers fetch it's assigned bucket file from all the remote workers and then runs reduce on all the keys of its bucket
bucket files
1
2
3
1
2
3
Master now sends workers reduce jobs specifying which bucket to work on. When jobs complete remaining jobs are handed out to idle workers
Workers store output locally and when complete the master may coordinate a merge

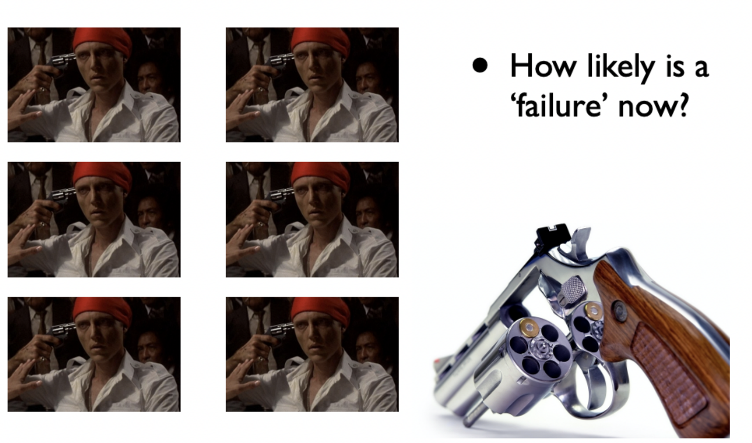
o. Coordinator assigns each workers the task
p. Each reducer → RPC → complete the reduce phase → output file is created
q. After reducer creates output files, the coordinator may also merge all the outputs

3. So why is this good?
    a. Lots of parallelism
        - Maps have each worker doing a independent access to a file and attendant processing
        - Reduce tasks are also independent and parallel
    b. Straggler (machine that is slow and delays the rest of the system) and failure handling
        - Handles stragglers (by running/assigning the task on two different workers and get the fastest one completed) and failures (the workers are stateless → it means they do not need any state but requires function only, if there is a failure, they can simply report to the coordinator and continue to work and ignore errors) transparently
4. Failures
    a. Coordinator failure (has state) can be a problem, not machine (stateless) failure
    b. If coordinator is important, why do they not talk about coordinator failures in the paper?
        -
    c. How likely is a 'failure'?



- How likely is a 'failure'?

    d.
    e. ⅙

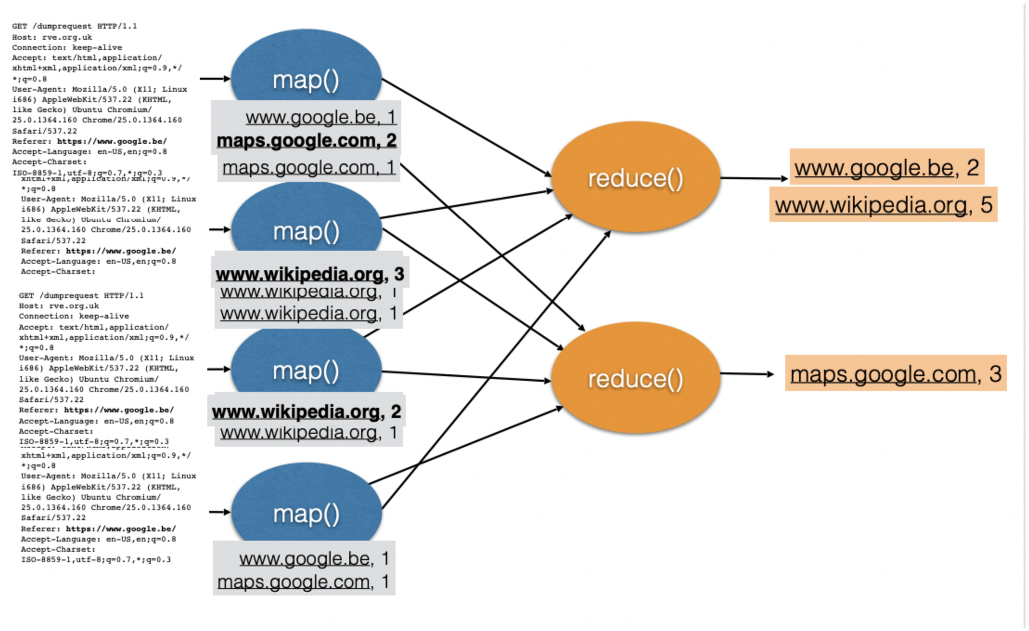


- How likely is a ‘failure’ now?

    f.
    g. Failures at scale are the norm

5. Stragglers and Failures
    a. Slow worker
    b. CPU is broken (eg. produces bad results)
    c. Worker or master might crash
    d. Network outage
    e. Bugs
6. Failure-handling Methods
    a. RETRY: If you can detect something went wrong retry the failing job (works good with workers since they are stateless)
    b. Coordinator continues to send question to the workers (heartbeat) to check the work status and if there is no answer from workers (the worker is dead), the work is sent to other workers → but there can exist duplicates of data
    c. How would you know the cpu is producing a bad result? → need more sophisticated protocol
    d. Replicate: Life is too short and machines are cheap always run job two times (n) (problem with this is that there exists duplicates, which could be troublesome)
    e. Replace/Repair
7. When doesn’t work well? (When mapReduce is a bad place to use)
    a. Small data
        - Overhead of scaling out to many machines
    b. Small updates in large datasets
        - Well-suited for bulk processing
    c. Iterative computations (model training in machine learning)
        - Multiple stages
    d. Computations where Mappers and Reducers need to choose input
    e. MapReduce does not work well with functions that take more than one input (joint in data → one data and secondary data)
8. Data & computation skew

a. Data Skew → non uniform distribution of a data set among workers
   - Use good partition function (each worker gets equal amount of data)
   - Combiner function → a way to do partial merging before the final aggregation by the reducer



```
GET /dumprequest HTTP/1.1
Host: rve.org.uk
Connection: keep-alive
Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.22 (KHTML,
like Gecko) Ubuntu Chromium/
25.0.1364.160 Chrome/25.0.1364.160
Safari/537.22
Referer: https://www.google.be/
Accept-Language: en-US,en;q=0.8
Accept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

```
xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.22 (KHTML,
like Gecko) Ubuntu Chromium/
25.0.1364.160 Chrome/25.0.1364.160
Safari/537.22
Referer: https://www.google.be/
Accept-Language: en-US,en;q=0.8
Accept-Charset:
```

```
GET /dumprequest HTTP/1.1
Host: rve.org.uk
Connection: keep-alive
Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.22 (KHTML,
like Gecko) Ubuntu Chromium/
25.0.1364.160 Chrome/25.0.1364.160
Safari/537.22
Referer: https://www.google.be/
Accept-Language: en-US,en;q=0.8
Accept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

```
xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.22 (KHTML,
like Gecko) Ubuntu Chromium/
25.0.1364.160 Chrome/25.0.1364.160
Safari/537.22
Referer: https://www.google.be/
Accept-Language: en-US,en;q=0.8
Accept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

   -
   - Similar to reduce function, but is used in the mapper function

b. Computation skew
   - Good scheduling strategy
   - Assign tasks to idle (or least-loaded) workers
   - Assign tasks to workers based on workload type and available resources

9.