CS425 MP4 - Group 31
bmjain2 (Bhavana Jain) and dipayan2 (Dipayan Mukherjee), Language : Go

We have built a Map-Reduce system using components from MP2( membership protocol) and MP3( Distributed File System). We have a centralized master server which runs the maple and the juice commands in the system, and maintains the file structure of the whole system and keeps track of the progress of the **maple**(Map) and **juice**(Reduce) tasks. Our system can handle failure of the worker nodes during any of the above phases (map, juice).

**1.DataStructure**
- WorkerNodes : The nodes chosen for the specific maple or juice task
- **Maple**
    - *node2mapleJob* : Keeps track of the maple jobs assigned to nodes, which basically tracks which files are being assigned for maple job to nodes. Also tracks the keys the node has generated and which it will be aggregating. This data structure is used for failure handing
    - *keyMapleIdMap* : Keeps track of the files in which the key is present
    - *keyStaus*: Keeps track of the status of a given key. It has three states (ONGOING, DONE,FAILED). This tells us about completion status of the mapleTask.
    - *mapleBarrier*: Tells us whether the file processing and key extraction stage of the maple has been done or not
    - *keyTimeStamp*: Tells us when the job for this key was submitted, helps us detect soft failures of the system

- **Juice**
    - *node2mapleJob* **:** Keeps track of the juice jobs (per key jobs) to nodes, and helps us during failure handling
    - *juiceCompMap* : Tells us about the status of the juiceID task, i.e for each intermediate files, what is the status. Tells us about the completion status.
    - *juiceTimestamp*: Tells us when the juice task was submitted, helps with the soft failures of the system

**2. Components of the Design:**
- **Maple Task:**
    When a maple task is started:
    - A) the master check all the files that matches the input prefix and adds them in a list
    - B) Master assigns each file an mapleID and assigns the task to nodes , and keeps track of how many files has been processed
    - C) Worker node on receiving a file for processing , runs the mapleExe and gets all the keys , and updates to the master

D) Master upon receiving all the keys from all the worker nodes, start the process of aggregating the key. It assigns a key to a node for aggregating and sends them the location of nodes containing the relevant files for the key and keeps track of the status of the key
E) Worker : For key aggregation, it fetches the relevant files from the node as directed by master and upon receiving the files aggregates it in a single file and updates master about the status of the Key, and adds it in the sdfs file system
F) Master on receiving all the keys from all nodes after aggregation finishes the MAPLE job

- **Juice Task:**
    A) Master : Upon receiving the Juice task finds all the relevant files for juice task
    B) Master : Assigns worker node each file to process(which are key specific ).
    C) Worker : On receiving juice task runs JuiceExe on the key-specific file and sends it output to master
    D) Maste : Upon receiving updates from all the Juice tasks finishes the juice task

## 3. Handling Failures:
**Maple :**

A) If failure happens before all files have processes their files with MapleExe master selects another node as worker and assigns the files for the failed nodes to it
B) Otherwise, master check if there is any required keyfile the crashed node handles which hasn't been aggregated yet,
    a) then it runs the complete maple task
    b) Otherwise : It simply reassigns the keys for aggregation to other nodes
C) Silent Failure:
    If a key aggregation for a key hasn't been received after a timeout, master reassigns the key to other nodes

**Juice :**
A) If a node fails then, another node is selected to run its juiceExe
B) If a keyJuice times out, master reassigns the key to other nodes

## 4. System Issues:

a) Limit on open connections or open files : While designing we encountered an issue where too much parallelism was causing the system to crash owing to huge open file descriptors, we mitigated that by using limited tokens for opening files or connections.
b) Too much context switch: When we spawned too many goroutines we observed that the system keeps on waiting and no tasks are completed, hence we kept a check on numbers of token for parallelism in the code

**5. Measurements:**

## Comparison of average runtime for wordcount (total keys=3117)
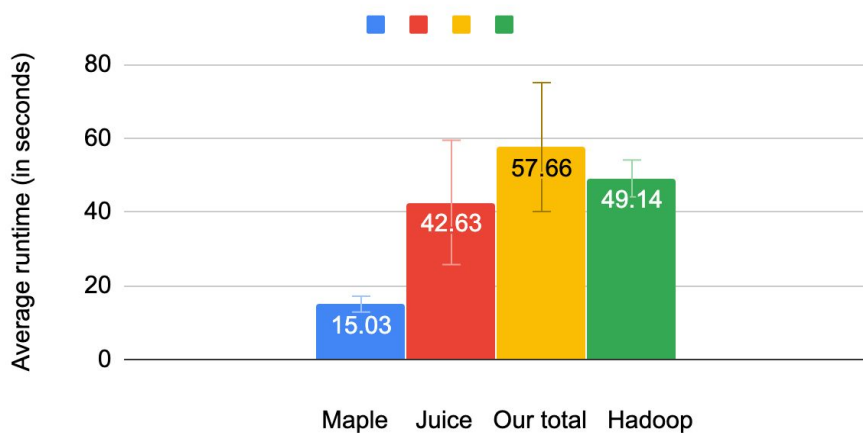


## Comparison of average runtime for URL frequency