Lab3 Analysis

Q1)

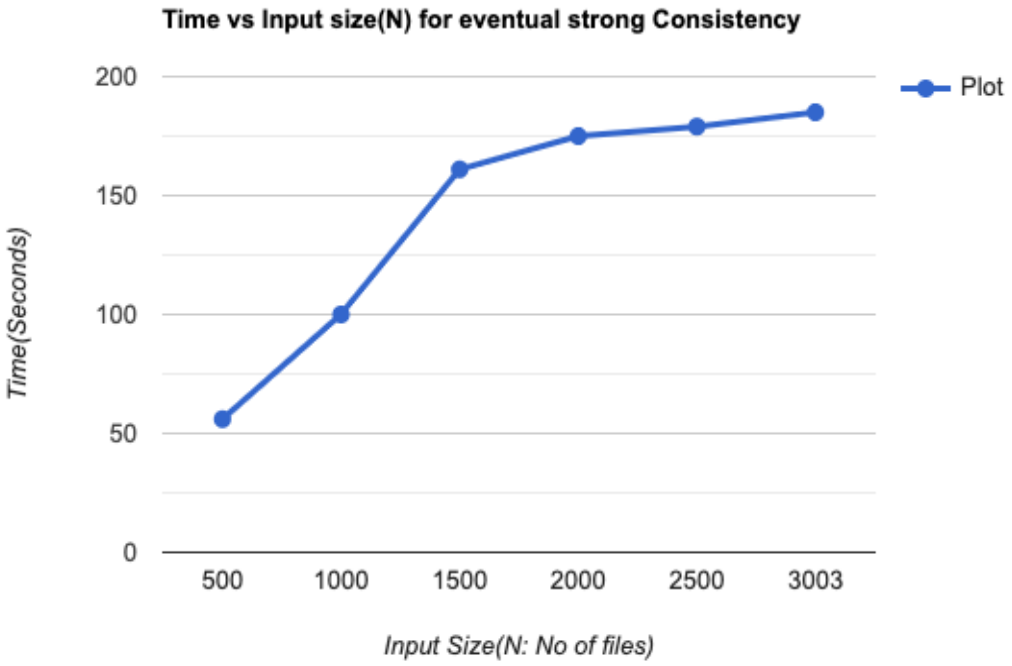
celery worker failure:

On celery worker failure, all the tasks which were held by the worker are appended back to the rabbitmq and assign failed tasks to some other workers in the next iteration that is done by using a flag "**acks_late"** and "**CELERY_TASK_REJECT_ON_WORKER_LOST=True".**
As a worker crashes the exception is handled by the flag and the task is again put back into rabbitmq queue for other workers to pick up.

Redis instance failure and Redis network partitions:

Before doing any operation on any Redis instance, we are checking if redis is available or not by doing **'ping'**.
We have used the set- dataset to maintain **CRDT** behavior.
As we get the word count of each file in **'map'** task, the rds function is called to save the word count of that file in all three Redis instances with set name as filename and word counts as key value pairs. And before adding that filename set we are checking in redis if the set with that filename doesn't already exist.
While doing so, we are expecting to get success for at least 2 redis instances, else we are retrying to save that set on all three instances. As we are checking filename before saving so there is no issue of double counting of words.
When the **'get_top_ten_words**()' is called and the parameter **heal=True** is passed, then we check the keys count in all three redis, which ever instance has lesser keys we assume that that must be the partitioned one and we take the intersection of union of keys of two up redis instances with the keys of down instance. By doing so we get the missing keys by down instance of redis due to partition. And we get the word count of those missing keys and save it in the redis which was down.This maintains all three redis instances consistency even after partition.
So, once all this is done, we now get that all the keys are the same in all the redis instances, so we merge all the sets in redis in for all the instances. If we get some error due to the **watch()** method, we redo the merging of sets for that instance. And only if the merges in all the instances are successful we return from function.
And the top 10 word counts are returned from a set with filename, because all the instances will have a single set with all the merged word count and single filename.
When the partition is there, we get **'redis timeout issue'** so those tasks are retried 3 times, even after 3 tries, tasks fail, we send them back to the queue.
As the code is done it handles both redis instance failure and redis network partition.

Q2)



Q3)

In the case of full partition in which all 3 VM are unable to talk to Redis instances, we are using **max retry** limit after that limit. If we are unable to talk to Redis instances then we send back all tasks to rabbitmq and this process repeats again.

Q4)

If we were to implement ‘Consistent Redis’, the same approach as we have done for ‘Available redis’ would have worked, Just a new logic needed to implement while merging the sets in all three redis instances.
In Available redis we are not focused on the consistency during the partition. We are not merging the sets.
But for Consistent Redis, we will periodically check if all keys (filename) are in common in all three redis and merge those in a single key and delete the rest of the keys.Here, we will introduce one extra set(**filestate**) for tracking the state of other keys if they are being merged or not. If the entry of the key(filename) is in that set then we know that the file is merged. For periodic check we will implement a celery **beat** worker. When the partition happens, we will still check the common keys between all three redis, but one redis won’t respond but we still want the consistency. We will merge the keys from the rest of the two working instances, but won’t delete the merged keys and put the key(filename) of those keys in the filestate set. So when the partition is down and all redis are accessible, the keys which were missing from the down redis will get added and the final merge will be done and all the redis will be in the same state.And then keys from the filestate set will be deleted and their respective sets are also deleted. Now, If a user needs top 10 keys at any time, whether during the partition or not, she will receive a consistent output.

Q5)

Same approach as mentioned in the anser of q4 can be used here for majority-minority partition. The state of both the partitioned grouped redis instances will be synced with each other. When the partition is lifted, all the redis instances can communicate with each other and the eventual consistency will be achieved.