COL733: Fundamentals of Cloud Computing
Lab-3: Network partition

Dipanshu Patidar(2018CS50405)
Nikita Bhamu(2018CS50413)

AvailableRedis:

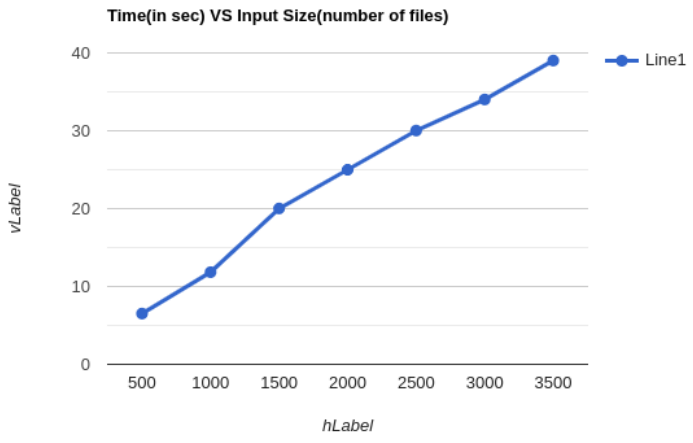
1)Give reasons for the correctness of your implementation. Cover all failure scenarios in your analysis: celery worker failure, Redis instance failure, and Redis network partitions.

In RabbitMQ, we create a task for workers to handle one file per job. After the worker has finished processing the file, we generate a dictionary of words and their counts, which we then convert to a string using the json dumps method. We then generate a tuple consisting of the filename, the word count dictionary in string format, which we then encode in string, and add this value to all three redis servers operating on three distinct nodes. Only after it writes to at least two redis servers can we declare our task is completed; otherwise, it keeps attempting.

- Correctness against celery worker failure: Since I am dumping a filename, value pair in redis set, My task is idempotent. That is if celery worker fails which means job is reschedule by RabbitMQ which means another worker will process the same job which means it will again write filename, value pair in redis set. Now if it was earlier not written then it will be written now else no problem it will again write with no effect/change in previous value. Our implementation will also handle stragglers. It will also handle network partition between RabbitMQ and celery worker. All these are possible because we designed idempotent tasks
- Correctness against Redis instance failure:If a worker can write to two redis instances, it will return and select the next job. If there are two redis instances accessible for the worker, it will successfully return in the event of a redis instance failure; otherwise, it will continue to wait and try to write to at least two redis instances. Once the operation has successfully written a file, it must be present in at least two places. And if at least one redis fails during read time, we can still read all successful writes; otherwise, some successful writes are lost.
- Correctness against Redis network partitions:As previously stated, as long as a worker can write to two redis instances, it will return and choose the next job. If we have two available redis for the worker in the case of a redis network partition, it will successfully return; otherwise, it will keep waiting and trying to write to at least two redis instances. Once the operation has successfully written a file, it must be present in at least two places. When the partition heals, we perform an algorithm that restores consistency to all redis instances.

2)Plot the time taken to achieve strong eventual consistency vs the input size (N) after the network partition is healed.

Number of files	Time(in seconds)
500	6.522
1000	11.838
1500	20.025
2000	24.998
2500	30.032
3000	34.024
3500	39.024



3)What happens if there is a full partition, i.e., all 3 VMs can’t talk to Redis instance on each other?
In the scenario described above, all workers on a given node can only communicate with the redis instance that is executing on that same node. That means that each worker can only write to one redis instance at a time, i.e. each worker can only write to one redis instance at a time => write fails because it must write to at least two instances for a successful write => it will attempt again => our software is waiting for the partition to heal. During that time, no work would be completed. Resources are being wasted.

ConsistentRedis:

4)If you were instead using a consistent Redis, which implementation performs better in the absence of network partitions?
Consistent Redis would perform better in the absence of network partition. This is due to the fact that no partition => writes are always successful for consistent redis because all instances are available, and it is also successful for available redis. However, throughout read time, redis has an eventual consistent algorithm that runs from time to time and ensures eventual consistency. Even if no network partition exists and all redis instances are consistent, available redis will still perform the algorithm and cross-check to ensure that all redis instances are consistent. Consistent redis, on the other hand, does not have an eventual consistency method because it always ensures consistency by rejecting writes that are not written to all instances. In the absence of a network split, there will be no reason for writes to fail, hence consistent redis will outperform available redis.
5)Which implementation is expected to perform better when there is a majority-minority partition, i.e, only 2 Redis instances can talk to each other but 1 can’t talk to any of the other 2?
Because no node is linked to every other node during such a majority-minority partition, all writes in consistent redis would fail, resulting in a waste of resources. Workers are rushing and failing to complete tasks, and work is not being completed.
In the case of available redis, all workers on nodes in the minority partition will fail to write because they can only write to one redis instance, i.e. a redis instance running on the same node, but all other workers on nodes in the majority partition will work and write successfully because they can write to two redis instances, i.e. one redis instance running on its own node and another redis instance running on the other node in the majority partition (partition). As a result, not all redis resources are wasted. In contrast to consistent redis, when the entire resource was squandered, 2/3 of the resources are still producing output while just 1/3 is wasted.