Lab2 :Analysis

1. For a fixed input size ,here , the input size taken is 100 files

Concurrency	Time of completion
4	3:36.785
5	3:10.357
6	3:01.892
7	2:28.225
8	2:09.562

So it can be observed that with the increase in worker threads the program executes at a faster rate

Reason:

It is because as the concurrency increases more workers are available to take up the tasks in parallel, for execution. It can be observed that a near linear iso-efficiency is present

2. The designed solution can handle load imbalance because in the solution each stream(w_0 and w_1) are checked for emptiness if either of the stream is empty workers are not allotted to it. Moreover once the task reaches worker pool it can be picked up by any worker so easily parallelizable

3. The code is fault tolerant as the code checks for messages that have not yet been ack-ed and only terminate when the entire stream is consumed. To counter any delays in streams a wait of almost 40s is applied. Moreover the wordset is written in a transaction and the transaction is being watched to ensure isolation (The ack being sent with the transaction) while the

tweet stream data is added to w_0 and w_1 in a transaction to ensure atomicity

4. The code is tolerant for network partition with RabbitMQ . As in the code acks_late is not used because in the tasks file a state is being generated and if acks_late is made true it would re-queue a task in its original state while it might have been altered while the original worker went down. So instead xack is used which ensures that a successful write has been made and as it is done via a transaction so it is isolated. So redis has the all the required entries for stream so until redis goes down , a network partition shouldn't effect.