COL733- LAB2
Vatsal Agrawal - 2021 MCS 2157

I have provided fault-tolerant, load balancing, and straggler solutions to my Programme as described below. I have not changed the client.py stream loading function.

Analysis

Q1 For a fixed input size, measure how the rate at which messages are processed from the '*tweet*' stream varies with an increase in worker threads allocated to the application. Explain your observations.

Number of Tweets in tweet stream = 3003125 (for given data)

Time Taken after filling Tweet Stream to complete processing with 8 threads = **31 seconds** (average) without straggler mitigation solution and **35 seconds** (average) with straggler mitigation solution and fault tolerance handling in all cases and load balancing. The best is 30 seconds. Note – It takes 10 to 15 seconds more if workers (Celery) are started after reading the whole stream for the initialization and starting of celery by the system and receiving tasks.

Rate is decreasing (becoming slower) with the decrease in the number of threads because fewer workers are present for concurrent execution. But the rate is decreasing at a low rate because of load balancing; as soon as one stream is empty, its worker finds other streams tasks to proceed. In 8 threads, we have 3 workers for Tweet and _1 (3T and 3W_1) and 2 workers for W_0 (2W_0); when we go to 7 threads, we have 3T and 2W_1 and 2W_0; this makes w stream computation slow as less number of worker are present to process them and when tweet processing stops its worker (3) are distributed unequally to these streams making the process more slower. While in thread 6, not much effect can be seen compared to 7 because tweet stream 3rd worker is not getting enough load, but it is also not sitting idle (so that it cannot switch to other tasks until streams end), also work done be unequal in this case every time. Similarly, we can justify other readings.

Note:- aggregating operation on w0 and w1 is a relatively costly process as compared to operation on a tweet can also be inferred from this

Time taken to consume only tweet stream and acknowledge it with 8 threads = 25 seconds

Note:- for the case when tweet stream is loaded and celery workers are called to fetch tasks, it takes 12 minutes for concurrency 8 and 7, and11 minutes for concurrency 6 to 4 (because of concurrent execution of tasks and loading, many of the times our worker are idle and more call to Redis uses much more latency). Also, it takes almost 6 minutes only to load the tweet stream

Graph and table are given below for threads.

Concurrency	Time in seconds	Rate in Tweet/sec
8	35	85803.571
7	42	71502.976
6	43	69840.116
5	48	62565.104
4	57	52686.403
3	92	32642.660

Table 1 - Concurrency Vs Time

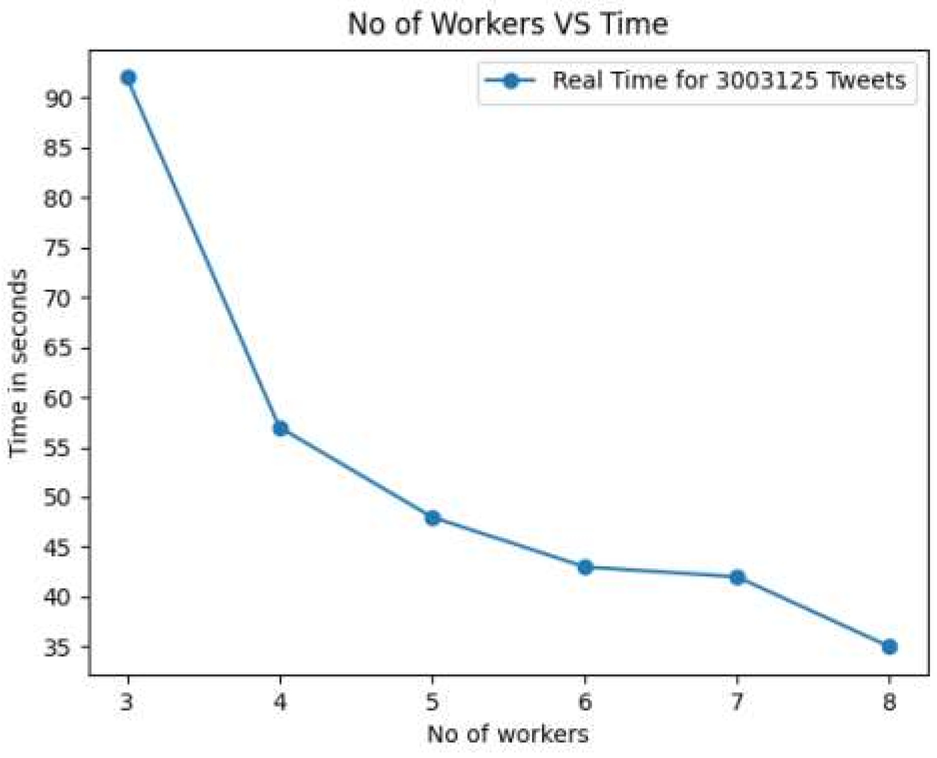


Fig 2 Number of Workers thread Vs Time

Q2 Given there are two streams, namely w_0 and w_1, which store words, how does the designed solution handle load imbalance, such as w_0 starts receiving a lot of messages whereas w_1 is not receiving any new messages?

Load Balancing is achieved by monitoring streams by each worker and setting a timeout for each of them. When workers see the stream is empty, they wait for timeout and die after that, letting new workers see for other streams if they have loads. Note, my Programme is based on the fact that workers are assigned sequentially in the loop to tweet stream then w1 and then w0, so if a worker w1 dies after timeout and w0 is the last stream worker was given, then new worker will see stream tweet for load than others. Note my timeout works in the type of backoff algorithm like first 4 worker timeout for tweet in 1 second, and following 4 workers will timeout after 2 seconds for tweet and each worker for w0 and w1 will timeout at interval starting from 1 second, incrementing each time by 1, this will ensure that workers do not die very fast and new workers need not be initiated again and again and switch to other tasks, which consumes times more than the good of load balancing; also it is done by seeing as w stream depends on tweet. Also, each worker is done in such a way that it may take a pending message (after a specific interval) for straggler and do processing on them, and ensure that same message do not update twice by transaction and ack review before final update. This will also balance loads of stragglers and make the process faster. Prefetching is also off so that workers get distributed by the pattern mentioned

Q3 Describe how your code is tolerant to celery worker failures. In other words, describe why your code is guaranteed to provide the same answer even if a worker crashes.

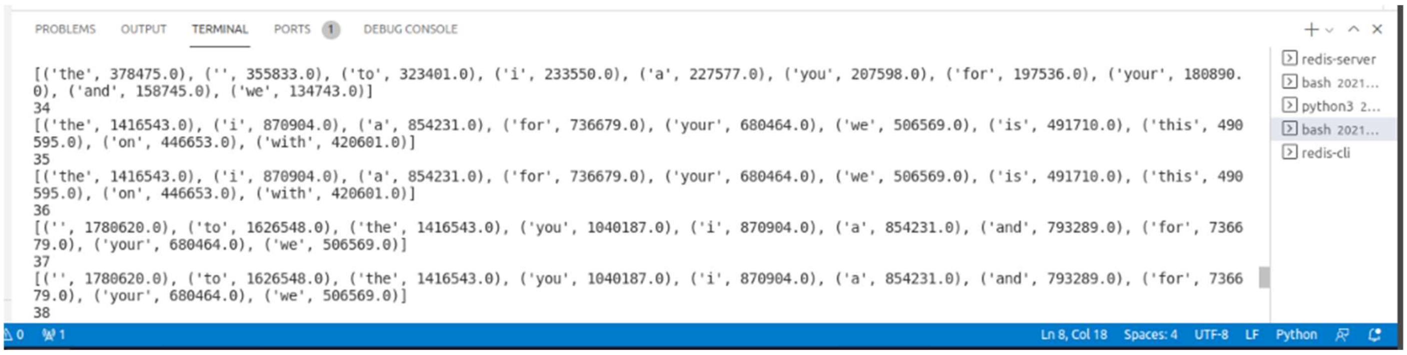
In case of some fault in celery workers, as celery workers die, a new worker will be initiated in the same pattern as described above. Each worker will first see the pending message for that stream with a specific timeout (60 seconds in my case) and will process them and then do standard processing (using auto claim). Also, if some workers die and a new worker is not given to that stream, then also, other workers working on that stream will check after a certain number of iterations (100) of their work and find a pending message and process them. Note:- At least one worker is assigned to each stream in last when all streams are empty so that that new message can be processed. My Programme also works when all streams are empty so that when new data comes at some point in time, it will be processed by maintaining an infinite while loop of workers for all 3 streams, which get executed after a certain number of workers die all 3 streams are empty. Also, backup workers are maintained in the case when infinite loop workers die. Also, if pending messages are due to a straggler, then my Programme will do its work and ensure that no duplicate data entry is fed by seeing the final acknowledged message before sending the final update message to Redis with the help of transactions. Also, transaction helps in atomic behaviour of combined effect of acknowledge sending and updating message in Redis. Note:- Certain parameters need to be changed like max loading for streams and removing backoff and providing timeout to workers, and providing new workers at a constant rate, in real case scenario where tweet stream may be load at a slow rate (like 1 tweet per second).

Q4 Describe how your code can be made tolerant to network partitions between a celery worker and RabbitMQ. A network partition occurs when a celery worker is unable to talk to RabbitMQ. It may be able to talk to Redis.

It can be done by listening to celery for some error from celery regarding rabbitmq not reachable or watching a number of workers active at a time (using monitor commands like worker-online); if the number of workers is less than the concurrency set, then it may be the case that rabbitmq and celery may have some network problem. In this case, direct celery workers can be initiated from the client by a command like run in infinite loop mode as described above for our Programme and watching an active number of workers; if they are less, then reinitiate them. In case rabbitmq comes to live, it can follow its usual process of providing workers tasks, unacknowledged tasks for celery can be recalled, but it will not create the problem of duplicate entry in Redis because the worker will read the latest entry by using the read command feature of Redis (with >).

Message broker main tasks are to provide periodic tasks to the worker (maybe on a different machine) and maintain acknowledgement of completed tasks. Instead of providing periodic tasks, we can run the worker in an infinite loop without a timeout and watch for some worker to die to assign a new worker by the client, as explained above.

Output-



```
[('the', 378475.0), ('', 355833.0), ('to', 323401.0), ('i', 233550.0), ('a', 227577.0), ('you', 207598.0), ('for', 197536.0), ('your', 180890.
0), ('and', 158745.0), ('we', 134743.0)]
34
[('the', 1416543.0), ('i', 870904.0), ('a', 854231.0), ('for', 736679.0), ('your', 680464.0), ('we', 506569.0), ('is', 491710.0), ('this', 490
595.0), ('on', 446653.0), ('with', 420601.0)]
35
[('the', 1416543.0), ('i', 870904.0), ('a', 854231.0), ('for', 736679.0), ('your', 680464.0), ('we', 506569.0), ('is', 491710.0), ('this', 490
595.0), ('on', 446653.0), ('with', 420601.0)]
36
[('', 1780620.0), ('to', 1626548.0), ('the', 1416543.0), ('you', 1040187.0), ('i', 870904.0), ('a', 854231.0), ('and', 793289.0), ('for', 7366
79.0), ('your', 680464.0), ('we', 506569.0)]
37
[('', 1780620.0), ('to', 1626548.0), ('the', 1416543.0), ('you', 1040187.0), ('i', 870904.0), ('a', 854231.0), ('and', 793289.0), ('for', 7366
79.0), ('your', 680464.0), ('we', 506569.0)]
38
```

Fig 2 Default Config 8 thread output from 33 seconds to 37 second

Declaration:-

I, Vatsal Agrawal, 2021MCS2157 has made this Programme independently without taking help, on baadalvm provided.
I have not used any external library except provided with VM.
trend.py is the same as that provided; I have used some part of it in my tasks.py. I have not changed client.py
I have verified my result to be True using test and whole data set using trend.py
The Programme is working for single and multi worker instances with different concurrency with an average time of 35 seconds with straggler mitigation on concurrency 8 and 30 seconds without straggler mitigation. Lowest time observed on a single worker is 30 sec
My Programme is also fault-tolerant, do load balancing and provide the solution to straggler without giving false result as tested and described below.

All the time and data are when celery workers are initialized already; if not already initialized, it takes 10 to 15 seconds more to get tasks to celery and start celery

Result:-

1) I have provided FT, Straggler and load balancing solutions.
2) Autoclaim helps in claiming to unacknowledge tasks. Autoclaim is run every time when new tasks start, or the running worker has done 100 iterations of its work.
3) Autoclaim will see and pick up all tasks pending more than 1 minute either due to straggler or some worker died which claimed that message. It is ensured that regular workers do not go beyond 1 minute for its working.
4) Autoclaim also ensures that if the straggler has acknowledged the tasks after it picked up the tasks, it does not update false results to Redis by using transaction and sending acknowledgement first then update when acknowledgement result is found to be true.
5) I have created 3 general tasks each for each stream which workers can act upon. Also, I have created consumers for each stream. These consumers maintain the worker of that stream.
6) I have provided worker to each stream in the manner of loop such that first tweet worker 1 is initialized then w1 stream worker 1 is initialized and then w0 stram worker 1 is initialized and then tweet worker 2 is made and so on.
7) Total of 60 workers for each stream is initialized with workers greater than 40 serial numbers working in an infinite loop so that when the stream becomes empty, then also worker is fault-tolerant and running. If init.py is to be modified, then the infinite worker can be supplied using that.
8) First, we call init.py, which gives tasks to rabbitmq, then client.py loads the stream, then we initialize celery and count time.
9) I have designed my Programme in such a way that when a worker is empty for some time, then it dies out and call other stream workers according to the loop explained above. This gives load balancing.
10) Die of worker follow a trend where tweet worker can die fast, and w worker dies slowly because w worker depends on tweet workers.
11) Average Serial Time observed is 35 seconds.
12) speed up achieved is 1.142 with tweet stream loaded.
13) concurrent execution of loading and processing takes 12 minutes because of more Redis calls to fewer tweets lines.
14) To avoid stragglers, we have set prefetching to false so that slow worker is not taking the extra job and fast worker can take more job.

File Structure:-

1) Graph contain the image of the graph obtained from data in the data folder
2) tasks.py contains all tasks that celery can do.
3) client.py contains the main Programme that will initialize the worksheet and fill the Redis tweet stream
4) config.py contains default config parameters.
5) init.py contains stream initialization and sending tasks list to celery. It is not a never-ending file.
6) trend.py is the same as trend.py, which was provided for serial computation and ground truth verification.

How to run Programme:-

1) Start init.py
2) Start client.py
3) start celery
4) then see Redis for word count and pending streams