COL733

Lab 2: Stream Processing

Dipen Kumar 2018CS50098

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.

Ans: the rate at which messages are processed from the 'tweet' stream ***increases*** with an increase in worker threads allocated to the application. However the application filling the tweet stream becomes the bottom neck and hence any added extra threads to process message from tweet stream sits idle waiting for tweet stream to get filled.

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?
Ans: My solution handle load imbalance. My master thread initially adds few tasks in RabbitMQ to process finite message (in my case 10,000 tweets) from tweet stream and dump it in w_0 or w_1 respectively. Once these tasks are returned new tasks are added in RabbitMQ according to following scheme.

if size of w_0 / number of tasks to process w_0 > threshold then add task to process w_0

if size of w_1 / number of tasks to process w_1 > threshold then add task to process w_1

and when these above two tasks return (they only return when their respective w_0, w_1 stream is empty), task to process tweet stream is added. This maintain the flow.

In mathematics terms we can say,

let x, y, z is the number of workers at tweet, w_0, and w_1 stream, we have

$x + y + z$ = total workers

$x = y + z$

$y / z = |w\_0| / |w\_1|$

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.
Ans: My design is fault tolerant. my worker read a redis data structure and write in another redis data structure. My claim is that i guarantee that if it reads the data structure and fails to write than I will re-read the same with different worker.

How I know it failed or succeeded to write?

I used pending acks to see that are pending and after given some time I decide to assign to another worker. ack and writing to another data structure is made atomic using transaction to ensure that if ack receive => written or written => ack receive such that it is not written multiple times.

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.

Ans: RabbitMQ is message broker that convey the work that is needed to be done by celery worker. If network partition occur than workers will sit idle waiting to hear from RabbitMQ about the next task. Cleary my design is not tolerant to network partitions between a celery worker and RabbitMQ. It can be made tolerant by designing celery task such that it don't return to master for scheduling next task in RabbitMQ. My design creates one new task for one completed task. Here celery worker can be told a priory about the logic to perform next procedure for which it may need to communicate with other existing tasks running for with they can have a share memory to communicate in redis by passing master thread need to coordinate all rather they all will coordinate themselves and act accordingly for load balancing and all.