Data sync performance and cost optimization report

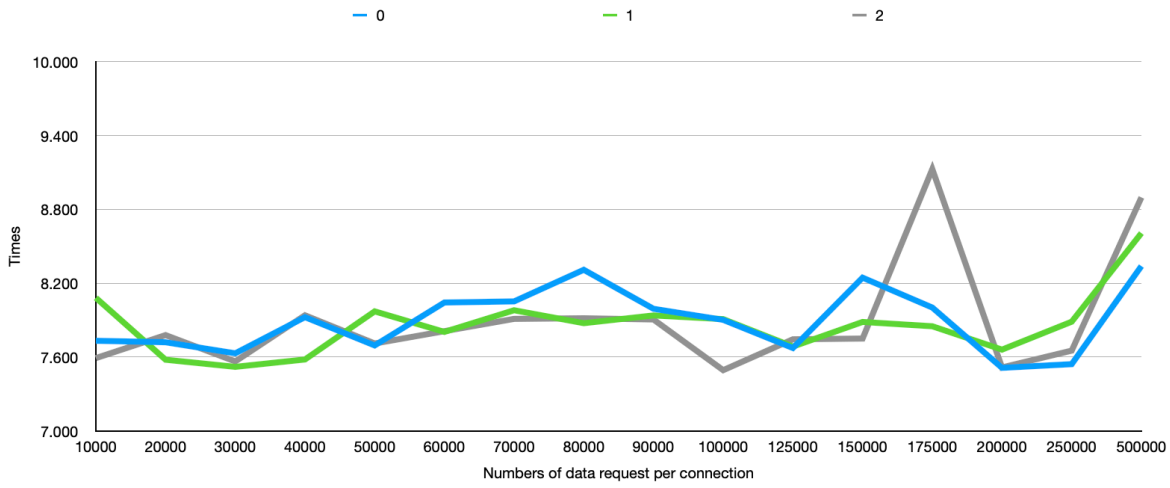
This report is apart into 2 main parts: network and cpu optimize
Github: https://github.com/Chol27/datasync

Network

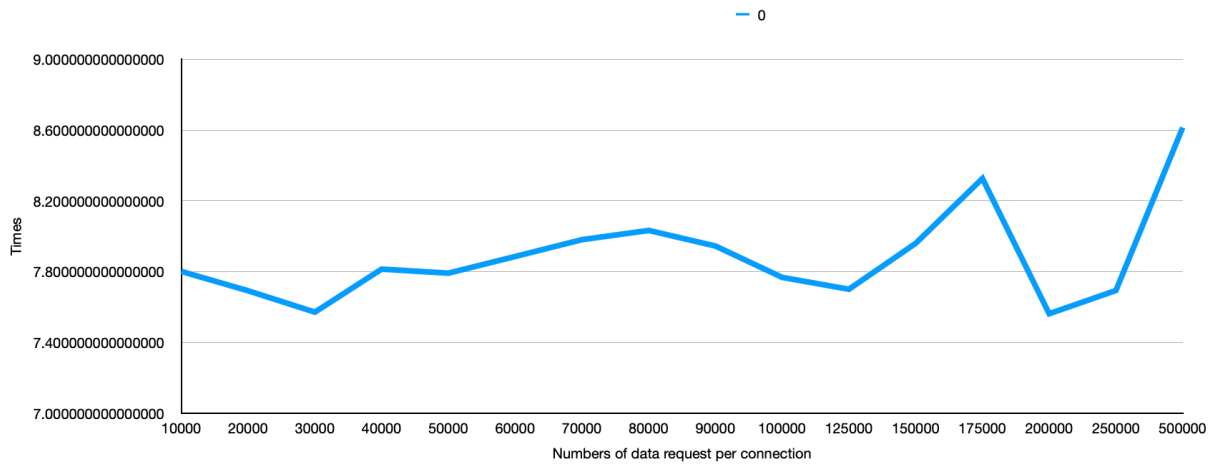
In this part, I tried to baseline times of sending data and can be divided more into 2 parts: database-server and server-client

database-server

As the base 500,000 rows data is inserted into the database and this maybe the biggest bottleneck of the network between database and server so what I tried is I divided request to database in to N connections with request for R rows of datas where N is 500,000/R and do it parallel.
As my understand, this test should be given the result times of each test should be differ since there are packets limits and parallel connections may take advantage of it, however the result showing like this



Real times values from time curl cmd of each tests



Average real times values from time curl cmd

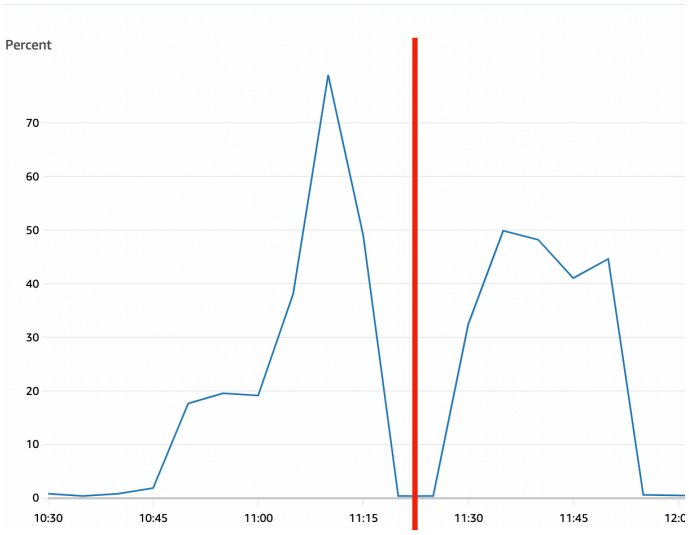
By this end point are spec like this

```
@Get('test/:size')
testFindAll(@Param('size', ParseIntPipe) size: number): Promise<any> {
  console.log('nm', typeof size);
  return this.messageService
    .testFindAll(size)
    .then((dt) => {
      console.log('size', dt.message.length);
      return { message: 'success' };
    })
    .catch((err) => {
      console.log('err', err);
      return { message: 'error' };
    });
}
```

```
async testFindAll(size: number): Promise<FetchAllDTO> {
  console.log('finding');
  const count = await this.actionService.getLatestActionId();
  let iterator = 0;
  const promises = [];
  while (iterator < count) {
    promises.push(
      this.messageRepo.find({
        order: { createActionId: 'ASC' },
        skip: iterator,
        take: Math.min(size, count - iterator),
      }),
    );
    iterator += size;
  }      You, 1 second ago • Uncommitted changes
  return {
    message: (await Promise.all(promises)).flat(),
    newLatestActionId: count,
  };
}
```

By size is specify the number of rows that one connection get from database.

And the CPU utilization showing like this (The testing times range is after at red line.)



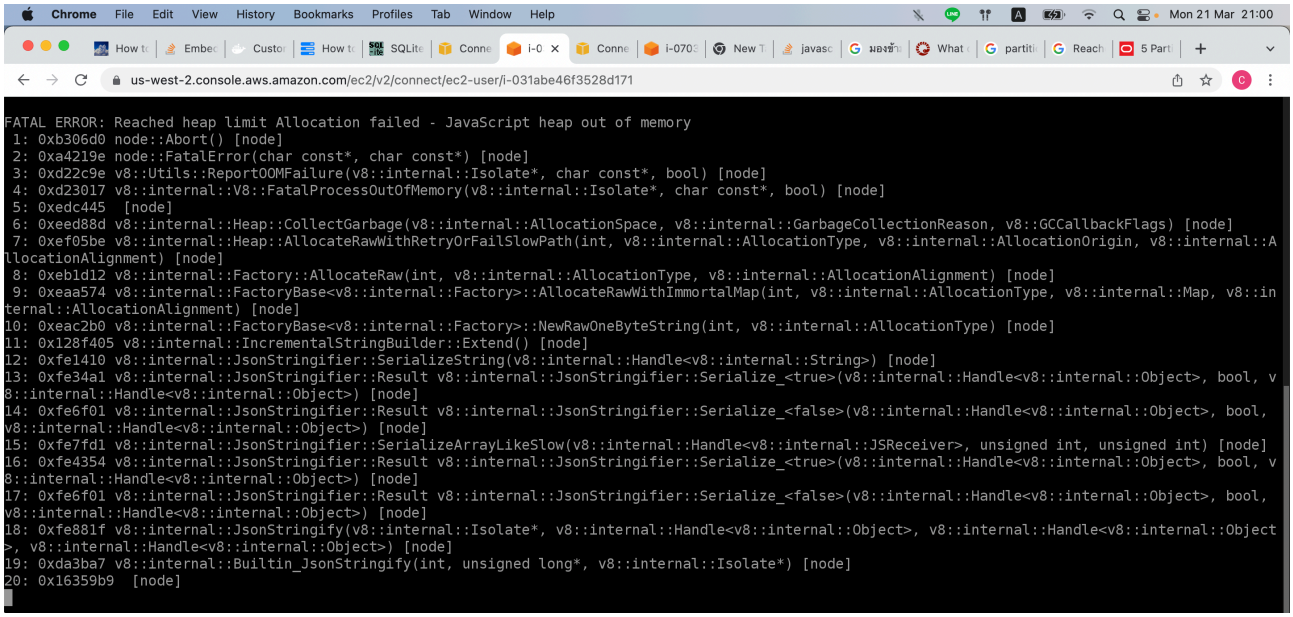
CPU utilization of server

CPU utilization of database

As my observation from this experiment I implied that the response times from request starting to slower when the data is getting up from 250,000 to 500,000 (didn't test between of these value since the limit of size we still could use 250,000 * 2 connections which are not that much differ if there are some of number in interval better) and this told that times to perform array concatenate could be overlook, but CPU utilization of database are significantly lower when it's come to lower connection so the maximum 250,000 rows per query and the number of connections are not improve speed of data transfer so choosing 250,000 maximum row per query is my choice. And my assumption is number of connections are not affect to times to transfer or may the load time that database have to do the query are nearing the differ time of transfer this size of data.

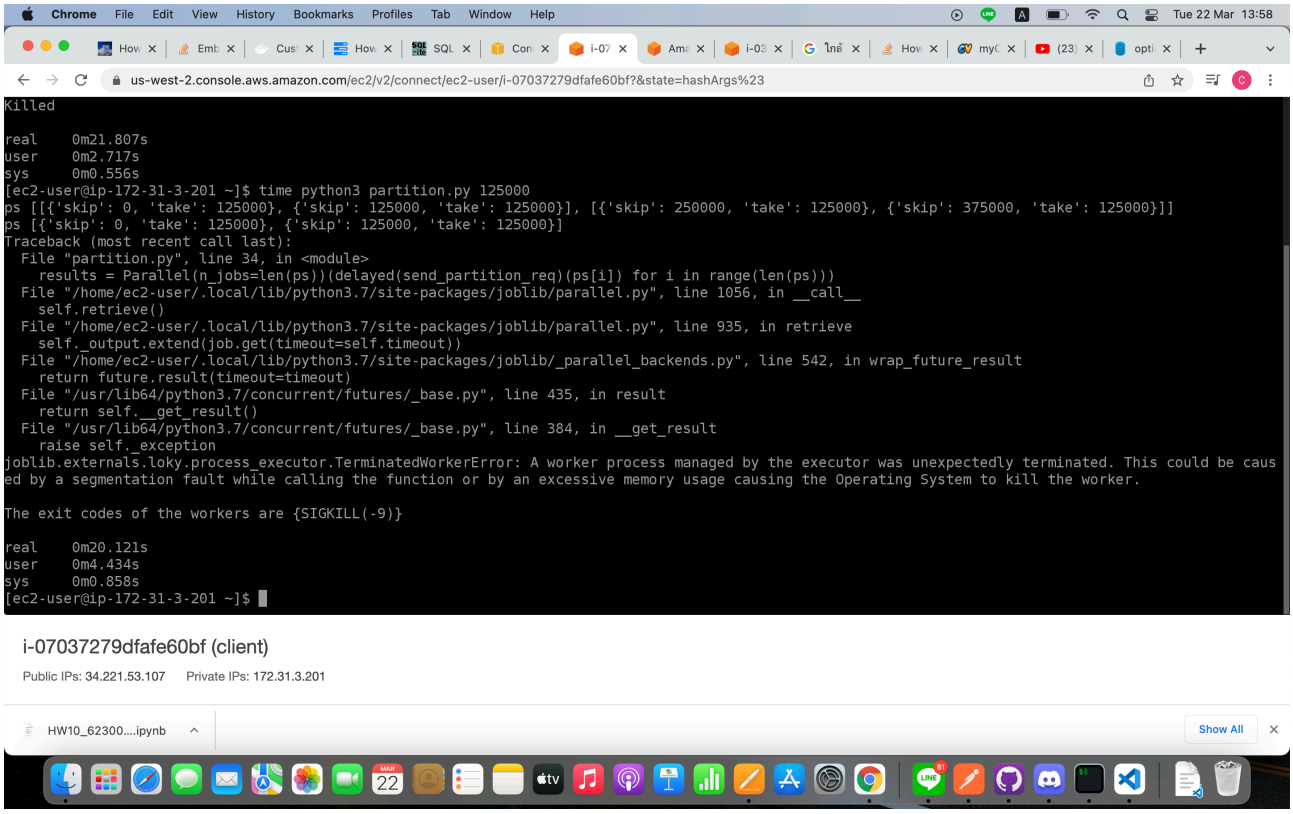
server-client

As limit of resource javascript could not write large 500,000 sizes array member JSON file



This problem is relied by using **gzip compression** and could transfer 250,000 rows of datas (more than this won't affect to our test since since it not enough for 500,000 rows so we still have to apart it into at least 2 transfers if we want to go for less connection)

For more connection test, I tried using python thread, thank to: https://stackoverflow.com/questions/9786102/how-do-i-parallelize-a-simple-python-loop
This is result of try using 2 threads with 125,000 rows transfer per thread. While just doing single 250,000 transfer the memory would enough and even faster than doing 2 threads. This mean this python thread lib is using some allocate and may need to optimize max thread and transfer per thread too.



The thing I do is defined a function like this.
I'll receive partitionSize and maxSize which partitionSize is number of rows which one thread would get, and maxSize would be the number of rows which all thread get in same times. This way I could define number of thread with floor of maxSize/partitionSize.

```
async testFindAll2(
  partitionSize: number,
  maxSize: number,
): Promise<FetchAllDTO> {
  console.log('finding');
  const count = await this.actionService.getLatestActionId();
  let iterator = 0;
  const partitionSets = [];
  let partitionSet = [];
  const setSize = Math.floor(maxSize / partitionSize);
  if (setSize === 0) throw new BadRequestException();
  while (iterator < count) {
    partitionSet.push({      You, 7 minutes ago • Uncommitted ch
      skip: iterator,
      take: Math.min(partitionSize, count - iterator),
    });
    iterator += partitionSize;
    if (partitionSet.length === setSize) {
      partitionSets.push(partitionSet);
      partitionSet = [];
    }
  }
  return {
    newLatestActionId: count,
    partitionSets,
  };
}
```

After some tests, I decided to doing ordinary 250,000 rows transfer two times and keep assumption from database-server part. (It always crash on size around 100,000 and 2 threads and I think lower this would not worth to do a parallel get.

CPU Optimize

Due to my lack of knowledge in network (prove by my network midterm test part II score). This part may more tangible to me with some of concept and algorithm.

In order to optimize CPU of database, server and client, the concept I used is CPU times is faster than transfer times. So structure data well and reduce it to smallest need since the **database** is better than transfer and structure in other layer.

Ideation

My idea is to take advantage of cluster index, so I don't have to unnecessaries fetch among whole records, the next thing to think is how I take advantage from it.

Action

As concept prof. giving in last class, I tried to track number of all **action** occur to the message table (CRUD), and giving the action **incremental id** using as time of all thing happen table (reason not using datetime is number is easier debugging furthermore it is unique (this test datetime would unique too but not in real situation))

```
datasync=# \d action
                                      Table "public.action"
      Column       |          Type           | Collation | Nullable |              Default
-------------------+-------------------------+-----------+----------+-----------------------------------
 id                | integer                 |           | not null | nextval('action_id_seq'::regclass)
 message_create_id | integer                 |           |          |
 action_type       | action_action_type_enum |           | not null |
Indexes:
    "PK_2d9db9cf5edfbbae74eb56e3a39" PRIMARY KEY, btree (id)
    "action_idx" btree (action_type, message_create_id) CLUSTER
Referenced by:
    TABLE "message" CONSTRAINT "FK_4f56665bc3f006ed12c3b79a264" FOREIGN KEY (latest_action_id) REFERENCES action(id)
```

Now I have actionId to tracking times I want to optimize size of change send from database. There are two ways in my idea which is

1. Differ from the data client currently have
2. Whole update that client doesn't have

I have no idea how to fast fetch current user record and compare to new record before sending out so I choose the second way.

And since I will take advantage of actionId the user would get latestActionId (LIDX) on each online to keep it on client.

What to transfer

[Create] New Message

First, if there are new records client doesn't have, we have to send every columns of that records so if we cluster message on action id it was created, the query new records would be cheap since we searching only new record that actionId more than LIDX

```
datasync=# \d message
                        Table "public.message"
      Column      |          Type           | Collation | Nullable | Default
------------------+-------------------------+-----------+----------+---------
 uuid             | character(36)           |           | not null |
 author           | character varying(64)   |           | not null |
 message          | character varying(1024) |           | not null |
 likes            | integer                 |           | not null |
 create_action_id | integer                 |           | not null |
 latest_action_id | integer                 |           |          |
```

[Update] MessageBatch

Define **batch** is the new update on each colums

Now I want to transfer only all new batch to user and take advantage from cluster index, so I define messageBatch table for each row update like this

```
datasync=# \d author_message_batch ;
                       Table "public.author_message_batch"
          Column           |         Type          | Collation | Nullable | Default
---------------------------+-----------------------+-----------+----------+---------
 action_id                 | integer               |           | not null |
 message_create_action_id | integer               |           | not null |
 is_latest                 | boolean               |           | not null | true
 updated_value             | character varying(64) |           |          |
Indexes:
    "PK_f62fe8a0943b9dc4bd2ecf05711" PRIMARY KEY, btree (action_id)
    "author_batch_idx" btree (is_latest, action_id) CLUSTER
```

message_create_action_id is refer to actionId the message of this batch is created

is_latest is flag to track if this batch is latest or not

updated_value is new value in the batch

I choose to cluster on is_latest which would cut all non-latest batch exactly (actually we can just delete old one and don't using this flag but I this not affect performance that much and I think it better practice to keep versioning of message)

The next thing to think is which one should be use as secondary index, action_id (AIDX) or message_create_action_id (MIDX).

The batch we want to query is row that LIDX < AIDX and LIDX >= MIDX

Comparing which one of these two should be qualify first

If we go on AIDX worst case would be like

Last fetch: LIDX = 900,000 and 400,000 records is updated before so there are 400,000 batch which got AIDX < LIDX and we have to go through all these 400,000 rows

Fi we go on MIDX worst case we would fetch only 100,000 new update so I go for MIDX

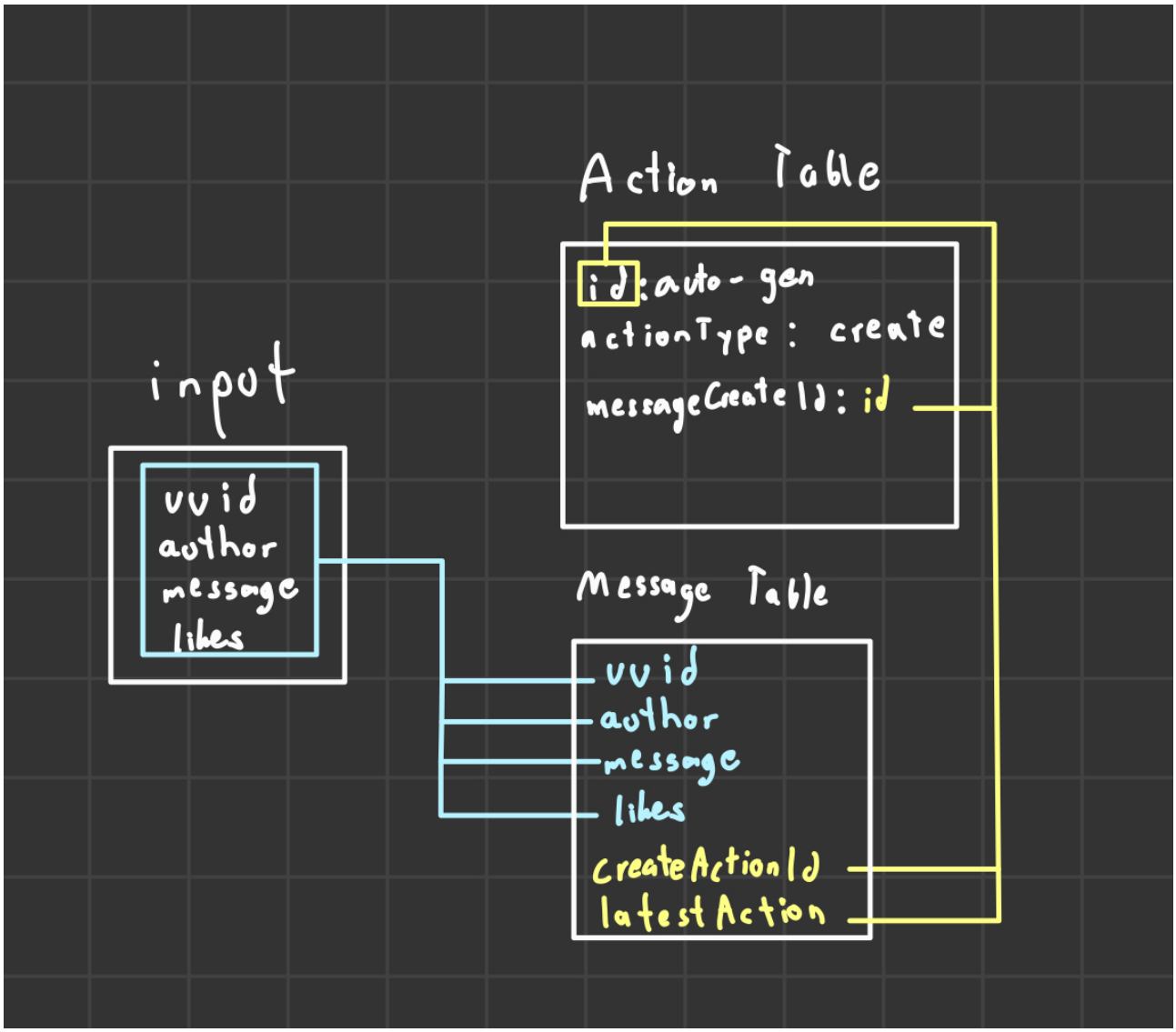
[Delete] Delete Action

Since we track createActionId of message in action too, we could get through action which type is delete and get all that AIDX > LIDX so I go cluster on action_type, however there are factor that made a unnecessaries like update and could have same worst case, so I choose the LIDX as secondary index like messageBatch.

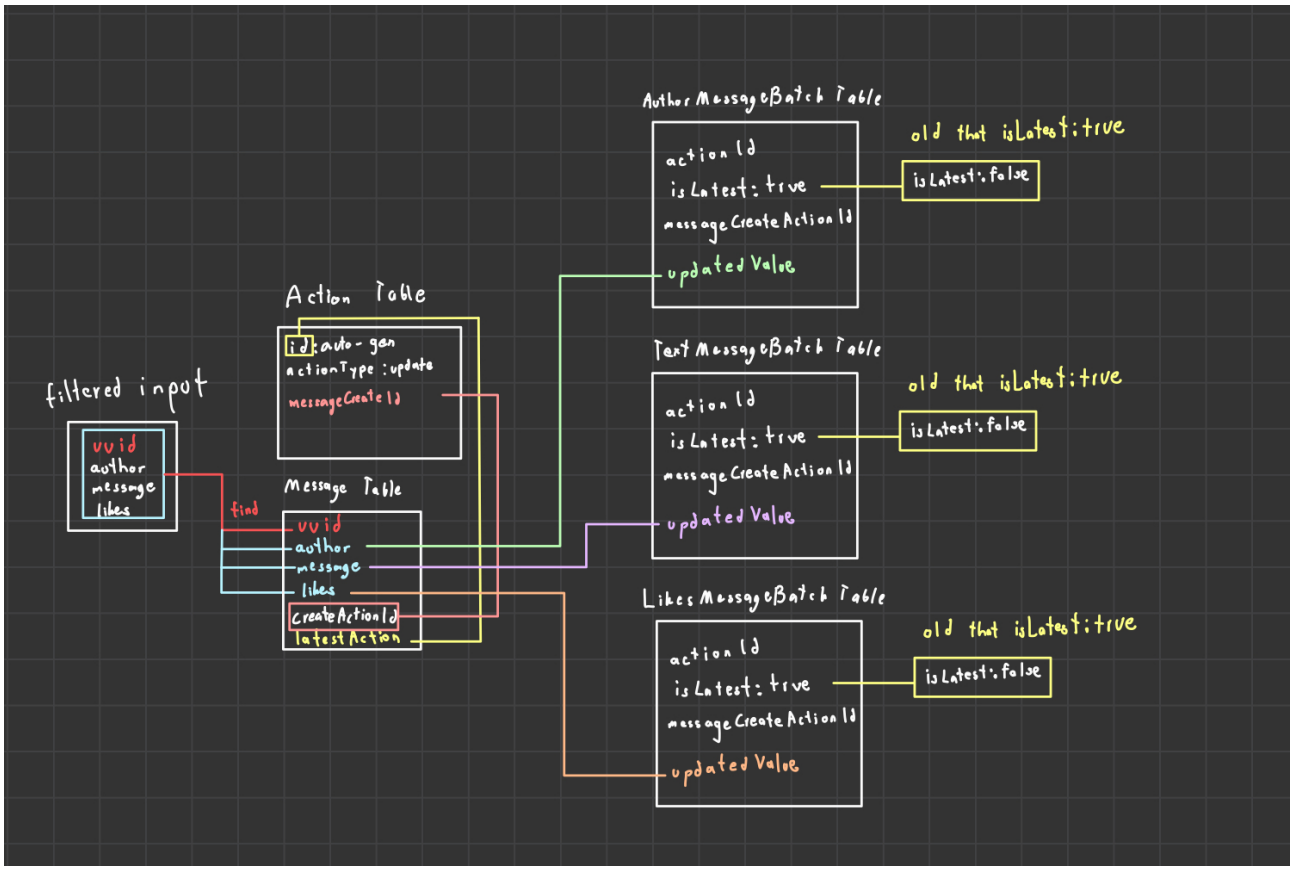
Some Decision
Actually I could partition newly create message into messageBatch too and I will save time to decide LIDX >= MIDX but it need to transfer whole CIDX of three messageBatch (there may have better way but I can't think of)

Server And Client
Using NestJs as server and python pandas as client (just on my own proficient).

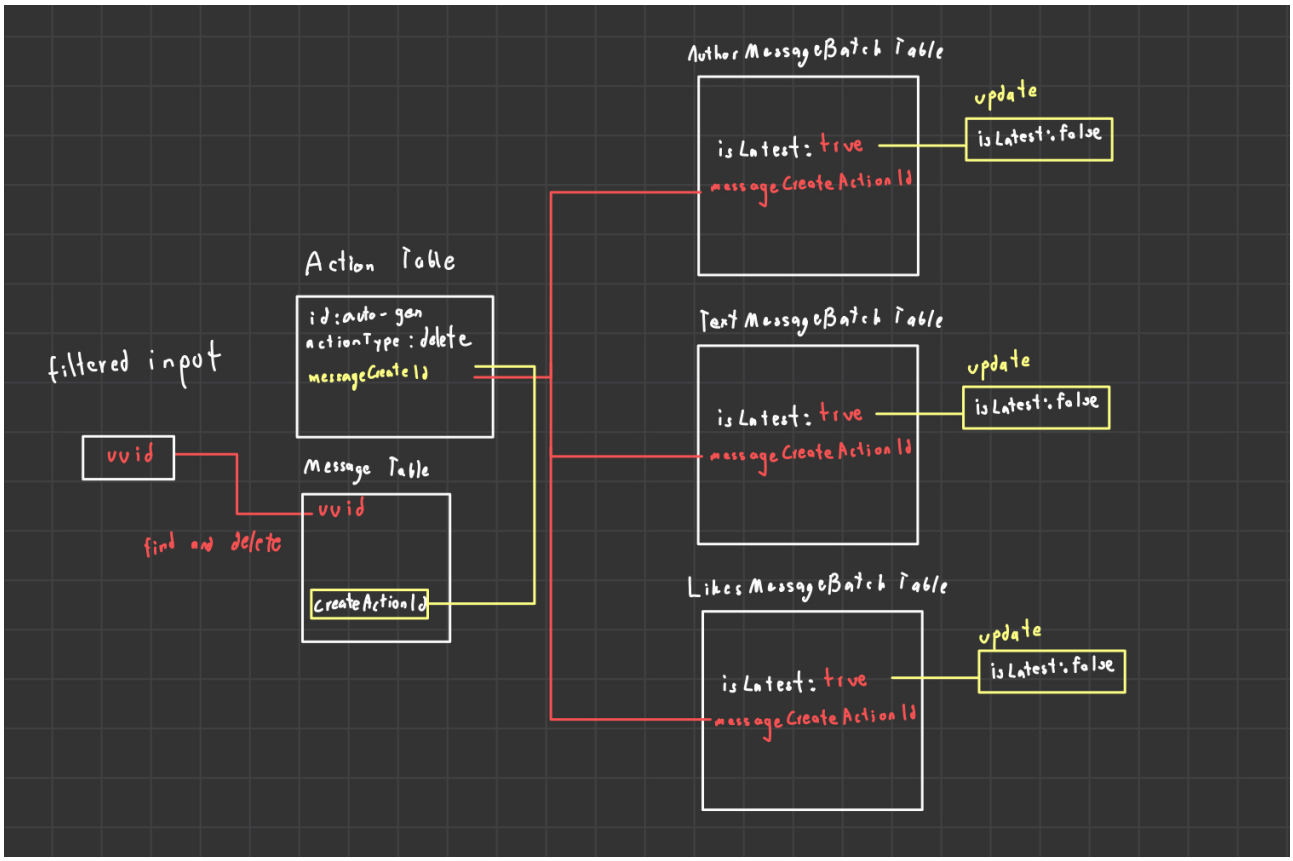
My report may confuse so for Visualize this I have a workflow diagram as picture below



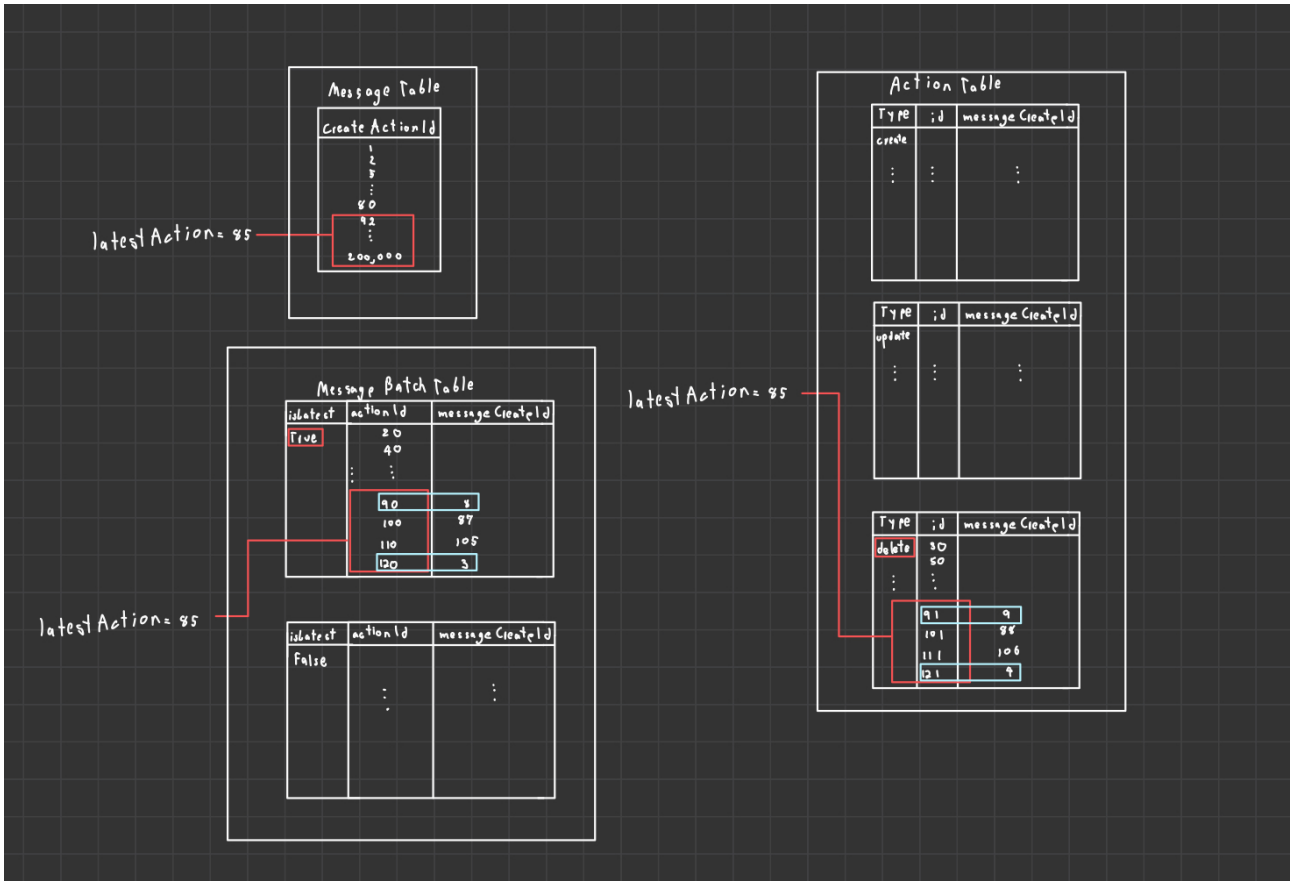
Create



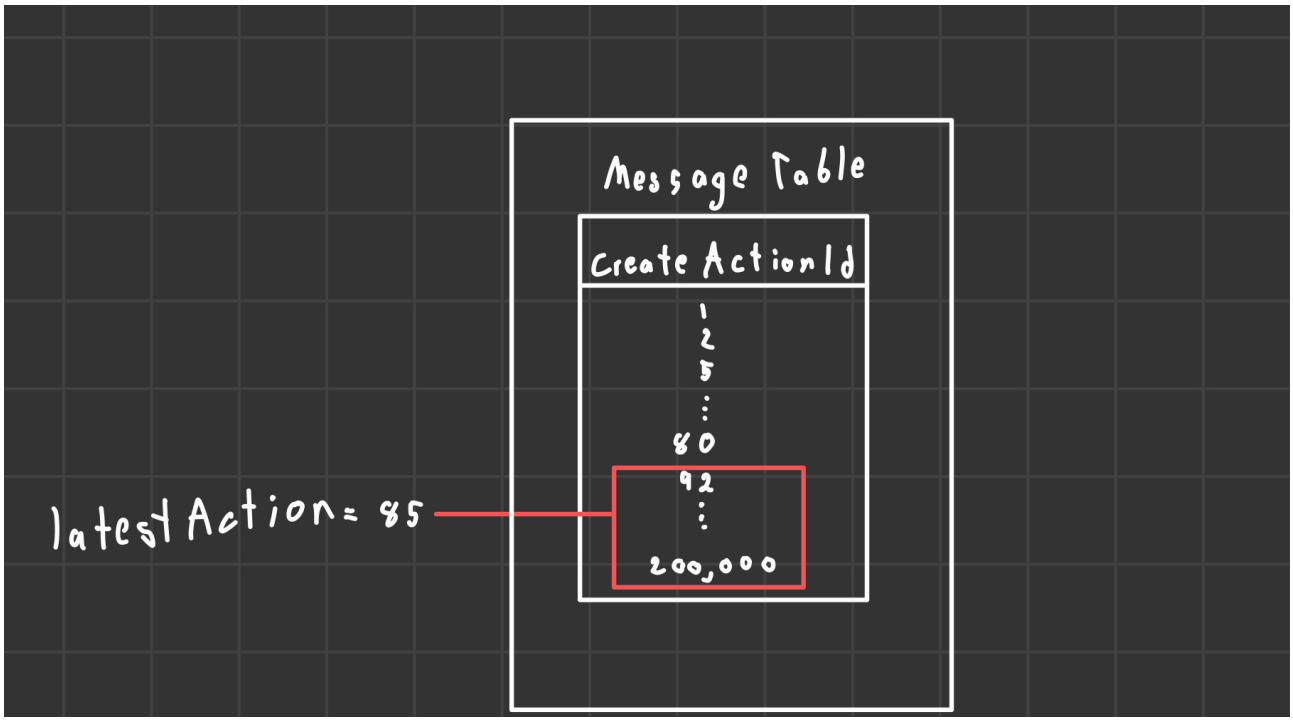
Update



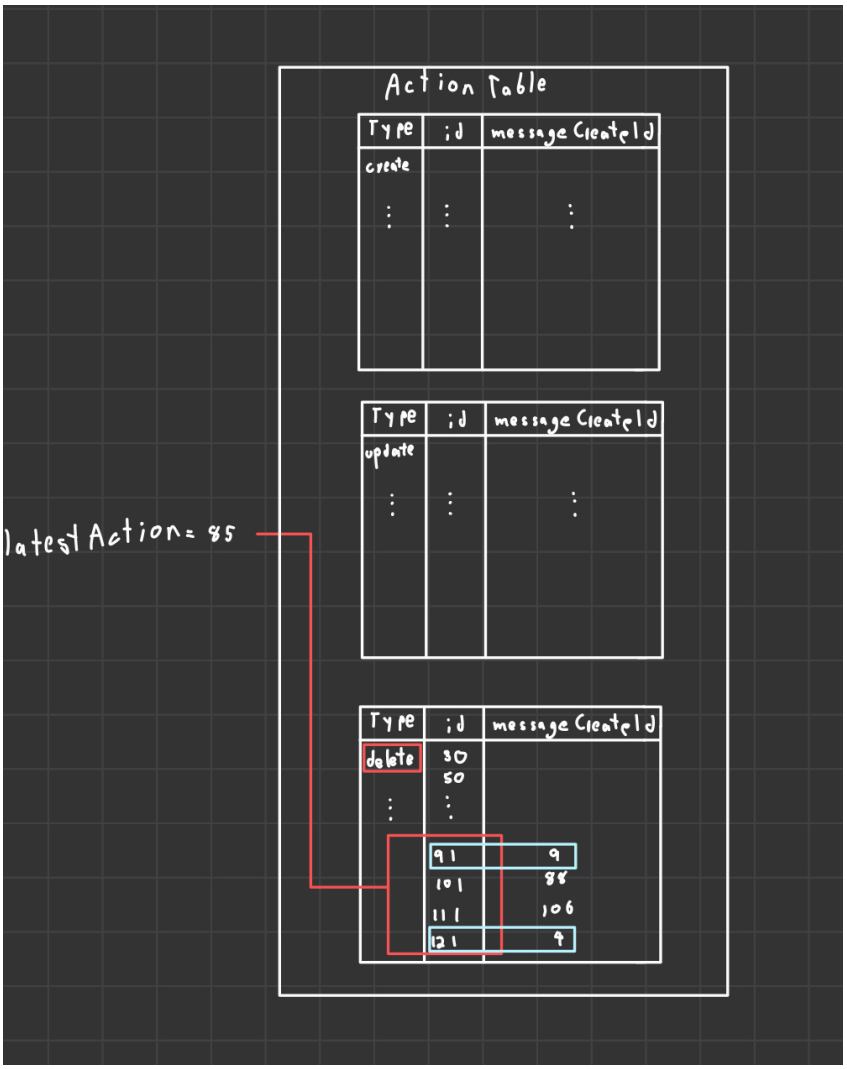
Delete



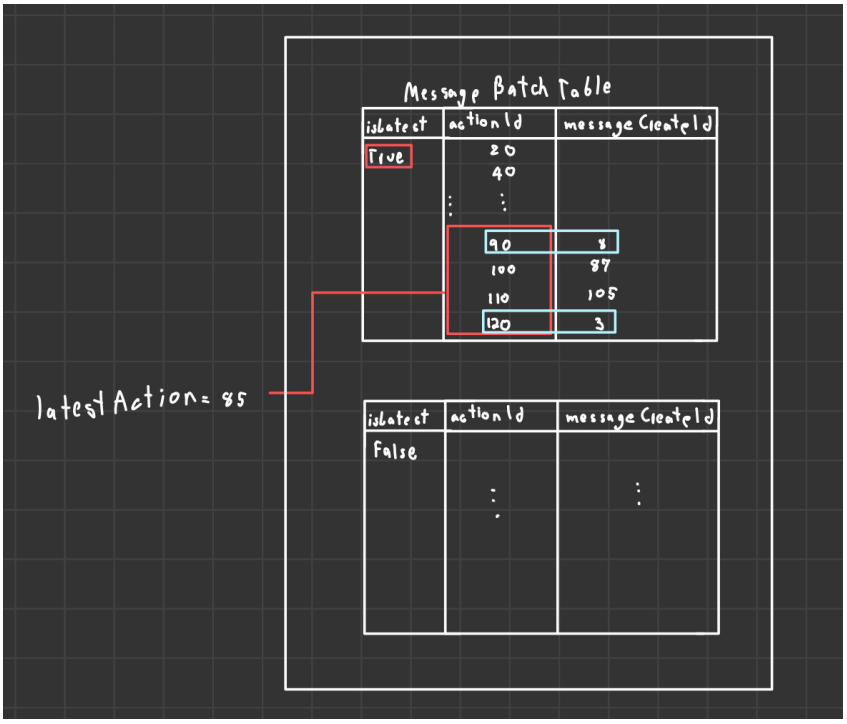
Read



Read (Message)



Read (Action)



Read (Each MessageBatch)