Report

Introduction:

This report will introduce how MongoDB executes queries and aggregations, particularly how indexes may help to improve execution performance by analyzing 6 questions in the assignment. All the queries and aggregations will import two data sets “tweets_hurricane.json” and “users_hurricane.json” as collection name tweets and users respectively. Performing on Robo 3T and eventually put all queries and aggregations together in a JavaScript file.

Performance analysis of query implementations

Q1:

Find the number of general tweets with at least one reply and one retweet in the data set. Note that a general tweet is a tweet with neither a replyto_id field, nor a retweet_id field; a reply is a tweet with the replyto_id field; a retweet is a tweet with the retweet_id field.

```
db.tweets.aggregate(
    [
```

Firstly I used MongoDB aggregation, because Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. This is very suitable for this problem.

```
db.tweets.createIndex({replyto_id:1})
db.tweets.createIndex({retweet_id:1})
```

I **created 2 indices on replyto_id and retweet_id** objects and will analyze the difference between using index and not using below.

Stage0:

```
{$match:
    {$and:[{replyto_id: {$exists: false}}, {retweet_id: {$exists: false}}]}
},
```

[0]	{ 3 fields }
$cursor	{ 2 fields }
nReturned	1641
executionTimeMillisEstimate	5

```
[0]                                  { 3 fields }
  $cursor                            { 2 fields }
  nReturned                          1641
  executionTimeMillisEstimate        2
```

About the $match stage, even though I added index on replyto_id and retweet_id, the execution time only 3 millis away. This is because whether index is added or not, this stage here will need to traverse all data. Also the explanation did not show the indexsUsed.

Stage1:

```
{$lookup: {
  from: "tweets",
  localField: "id" ,
  foreignField: "replyto_id",
  as: "replyto_tweet_Array"}
},
```

```
[1]                                  { 7 fields }
  $lookup                            { 4 fields }
  totalDocsExamined                  16410000
  totalKeysExamined                  0
  collectionScans                    3282
  indexesUsed                        [ 0 elements ]
  nReturned                          1641
  executionTimeMillisEstimate        5522
```

```
[1]                                  { 7 fields }
  $lookup                            { 4 fields }
  totalDocsExamined                  21
  totalKeysExamined                  21
  collectionScans                    0
  indexesUsed                        [ 1 element ]
  nReturned                          1641
  executionTimeMillisEstimate        85
```

About the $lookup stage, the excuting time before adding index and after has almost 5400 milliseconds difference. This is because Firstly, before adding index, the time complexity of append each 'replyto_id' to id can be assessed as O(n^2), but after adding index, the data is stored in a B-tree, and the total time complexity of this step can be assessed as O(nlog(n)). So as the data become larger the excuting time gap between the two methods will rapidly increase. Secondly, we can see the totalDocsExamied before creating index is 16410000, this is because for each "id", it will need to go through almost whole data set, to find the matching "replyto_id" in average. After adding index, the data in tweets collection are put into entries, so in the secend image, the **totalKeyExamied incresed from 0 to 1**, it just need to go through the sorted key in a B-tree data structure, this resulted in a significant reduction in time. By the way i**n this $lookup stage it just need to create the index of the foreignField which is "replyto_id"**, we do not need to creat index for localField which is "id". It is nothing help and after my test, if I create index for localField, the executing time will even be higher like below:

[1]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	21
totalKeysExamined	21
collectionScans	0
indexesUsed	[1 element]
nReturned	1641
executionTimeMillisEstimate	88

Stage2:

```
{$match:
    {$expr:
        {$gt:[{$size: "$replyto_tweet_Array"}, 0]}
    }
},
```

[2]	{ 3 fields }
$match	{ 1 field }
nReturned	19
executionTimeMillisEstimate	5522
[2]	{ 3 fields }
$match	{ 1 field }
nReturned	19
executionTimeMillisEstimate	85

I have no idea why the execute time of both before and after adding index data will equal to last $lookup stage, I guessed that the execution time is superimposed in the explanation. But after searching online and post question on ed, the tutor on ed told me that the stages do not run in parallel and the given execution times are estimates, not actual execution times. So according to the time complexity, the execution time of both should be close.

Stage3:

```
{$lookup: {
  from: "tweets",
  localField: "id" ,
  foreignField: "retweet_id",
  as: "retweet_Array"}
},
```

[3]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	190000
totalKeysExamined	0
collectionScans	38
indexesUsed	[0 elements]
nReturned	19
executionTimeMillisEstimate	5591

[3]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	106
totalKeysExamined	106
collectionScans	0
indexesUsed	[1 element]
nReturned	19
executionTimeMillisEstimate	85

This $lookup stage after **creating retweet_id index**, also saved 5500 milliseconds, same reason as $lookup in stage 1. After creating index, the totalDocsExamied decreased to 106 from 190000, and totalKeysExamied increased same reason as stage 1.

Stage4:

```
{$match:
    {$expr:
        {$gt:[{$size: "$retweet_Array"}, 0]}
    }
},
```

[4]	{ 3 fields }
$match	{ 1 field }
nReturned	10
executionTimeMillisEstimate	5591

[4]	{ 3 fields }
$match	{ 1 field }
nReturned	10
executionTimeMillisEstimate	85

This is same as stage 2, except for the object name changed to “retweet_Array” which is a table created by last $lookup stage.

Stage5:

```
{$count: "Number of tweets"}
```

[5]	{ 6 fields }	Object
$group	{ 2 fields }	Object
maxAccumulatorMemoryUsageBytes	{ 1 field }	Object
totalOutputDataSizeBytes	229	Int64
usedDisk	false	Boolean
nReturned	1	Int64
executionTimeMillisEstimate	5591	Int64
[6]	{ 3 fields }	Object
$project	{ 2 fields }	Object
nReturned	1	Int64
executionTimeMillisEstimate	5591	Int64

```
[5]                                  { 6 fields }
  $group                             { 2 fields }
  maxAccumulatorMemoryUsageBytes     { 1 field }
  totalOutputDataSizeBytes           229
  usedDisk                           false
  nReturned                          1
  executionTimeMillisEstimate        85
[6]                                  { 3 fields }
  $project                           { 2 fields }
  nReturned                          1
  executionTimeMillisEstimate        85
```

In this $count stage, this command include two steps, we can obviously find by explain(). The first step is $group and the second step is $project. In the $group stage, **there is a totalOutputDataSizeBytes parameter which represents the memory usage the stage take.** The nReturned parameter represents the data finally showed on the interface, here the one data is our target result.

Q2:

Find the reply tweet that has the most retweets in the data set.

```
db.tweets.explain("executionStats").aggregate(
  [
```

As Q1 said, aggregate is more suitable in this case. And below is the explain() of my using aggregation stages, and I will only put a comparison image before and after adding the index when I need it.

Indexing:

```
db.tweets.createIndex({retweet_id:1})
```

Stage 0:

```
{$match:
    {replyto_id: {$exists:true}}
},
```

```
[0]                                  { 3 fields }
  $cursor                            { 2 fields }
  nReturned                          621
  executionTimeMillisEstimate        3
```

Find the 621 data which is contain replyto_id, with execution time 3ms.

Stage 1:

```
{$lookup: {
  from: "tweets",
  localField: "id" ,
  foreignField: "retweet_id",
  as: "retweets"}
},
```

[1]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	6210000
totalKeysExamined	0
collectionScans	1242
indexesUsed	[0 elements]
nReturned	621
executionTimeMillisEstimate	2199

[1]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	27
totalKeysExamined	27
collectionScans	0
indexesUsed	[1 element]
nReturned	621
executionTimeMillisEstimate	40

After **creating index of retweet_id,** execution time decreases 2150ms. And the totalDocsExamined almost decreased 6210000, and the totalDocsExamined actually is the totalKeysExamined. After lookup, the return document is still 621.

Stage 2:

```
{$project:
   {
      _id: 0,
      id: 1,
      retweet_count: {$size: "$retweets"}
   }
},
```

[2]	{ 3 fields }
$project	{ 3 fields }
nReturned	621
executionTimeMillisEstimate	40

$project does not change the document returned and take 40ms for modifying the data.

Stage 3:

```
   {$sort: {retweet_count: -1}},
```

[3]	{ 5 fields }
$sort	{ 2 fields }
totalDataSizeSortedBytesEstimate	0
usedDisk	false
nReturned	1
executionTimeMillisEstimate	40

$sort stage **used 0 memory** showed by explain, but actually it does take some memory otherwise the **totalOutputDataSizeBytes parameter will not appear.** And this stage takes 40 ms and return 1 document.

Stage 4:

```
{$limit:1}
```

This stage can not be explained, it just makes the interface show the first document.

Q3:

Find the top 5 hashtags appearing as the FIRST hashtag in a general or reply tweet, ignoring the case of the hashtag. Note that the order does not matter if a few hashtags have the same occurrence number.

```
db.tweets.explain("executionStats").aggregate(
    [
```

Refer to the first question about why use aggregate here.

Index: in this question I did create any index. Simply say that it is because$ lookup is not used in this question. For detailed explanation refer to the first question.

Stage 0:

```
{$match:
    {$or: [
            {$and:[{replyto_id: {$exists: false}}, {retweet_id: {$exists: false}}, {hash_tags: {$exists: true}}]},
            {$and:[{replyto_id: {$exists: true}}, {hash_tags: {$exists: true}}]}
          ]
    }
},
```

[0]	{ 3 fields }
$cursor	{ 2 fields }
nReturned	211
executionTimeMillisEstimate	6

In this $match stage, it contained one $or operation and two $and operations. This stage takes 6 ms and return 211 documents.

Stage 1:

```
{$project:
    {
      first_hashtag:{ $arrayElemAt: ["$hash_tags.text", 0]}
    }

},
```

```
{$group:
    {
        _id:{$toLower: "$first_hashtag"},
        count: {$sum:1}
    }
},
```

[1]	{ 6 fields }
$group	{ 2 fields }
maxAccumulatorMemoryUsageBytes	{ 1 field }
totalOutputDataSizeBytes	25272
usedDisk	false
nReturned	106
executionTimeMillisEstimate	6

For some unknow reasons the $project stage did not show on the explain() method.
In this $group stage, it causes **totalOutputDataSizeBytes: 25272**, which is a huge cost of memory, this stage takes 6 ms and return 106 documents.

Stage 2:

```
{$sort:
    {count:-1}
},
```

[2]	{ 5 fields }
$sort	{ 2 fields }
totalDataSizeSortedBytesEstimate	3756
usedDisk	false
nReturned	5
executionTimeMillisEstimate	6

In this $sort stage. It takes 6 ms and return 5 documents. But it cost **3756 Bytes memory.**

Stage 3:

```
{$project:
    {
      _id: 0,
      tag: "$_id",
      count: "$count"
    }

}
```

```
[3]                                  { 3 fields }
  $project                           { 3 fields }
    tag                              $_id
    count                            $count
    _id                              false
  nReturned                          5
  executionTimeMillisEstimate        6
```

In this $project stage. It takes 6 ms and return 5 documents.

Q4:

For a given hash_tag, there are many tweets including that hash_tag. Some of those tweets mention one or many users. Among all users mentioned in those tweets, find the top 5 users with the most followers_count. For each user, you should print out the id, name and location. Not all users have a profile in the users data set; you can ignore those that do not have a profile. If there are less than 5 users with profile, print just those users with a profile.

```
db.tweets.explain("executionStats").aggregate(
  [
```

Refer to the first question about why use aggregate here.

Index: in this question I did create any index. I was supposed to created but the index I need to create is a object inside an array.

Stage 0:

```
{$match:
    {$and:[
            {user_mentions: {$exists: true}},
            {hash_tags: {$exists: true}}
        ]
    }
},
```

```
[0]                                  { 3 fields }
  $cursor                            { 2 fields }
  nReturned                          881
  executionTimeMillisEstimate        13
```

In this $match stage. It takes 13 ms and return 881 documents.

Stage 1, 2, 4:

```
{$unwind: "$hash_tags" },
{$unwind: "$hash_tags.text"},
{$unwind: "$user_mentions"},
```

[1]	{ 3 fields }
$unwind	{ 1 field }
nReturned	1383
executionTimeMillisEstimate	5
[2]	{ 3 fields }
$unwind	{ 1 field }
nReturned	1383
executionTimeMillisEstimate	5
[4]	{ 3 fields }
$unwind	{ 1 field }
nReturned	328
executionTimeMillisEstimate	5

In these three unwind stage. All three stages take 5 ms and return 881, 1383, 328 documents respectively. And I have no idea why the explain do not follow the order the query write.

Stage 3:

```
{$match:
   {
       "hash_tags.text": "Ida"
   }
},
```

[3]	{ 3 fields }
$match	{ 1 field }
nReturned	244
executionTimeMillisEstimate	5

In this $match stage. It takes 5 ms and return 244 documents.

Stage 5:

```
{$lookup: {
     from: "users",
     localField: "user_mentions.id" ,
     foreignField: "id",
     as: "mentioned_user_info"}
},
```

[5]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	2731584
totalKeysExamined	0
collectionScans	656
indexesUsed	[0 elements]
nReturned	328
executionTimeMillisEstimate	1536

In this $lookup stage, it examined 2731584 documents, without using index, returned 328 documents and take 1536ms.

Stage 6:

```
{$match:
   {$expr:
       {$gt:[{$size: "$mentioned_user_info"}, 0]}
   }
},
```

[6]	{ 3 fields }
$match	{ 1 field }
nReturned	37
executionTimeMillisEstimate	1536

In this $match stage. It takes 1536 ms and returned 37 documents.

Stage 7:

```
{$unwind: "$mentioned_user_info"},
```

[7]	{ 3 fields }
$unwind	{ 1 field }
nReturned	37
executionTimeMillisEstimate	1536

In this $unwind stage. It takes 1536 ms and returned 37 documents.

Stage 8:

```
{$match:
   {$and:[
           {"mentioned_user_info.id": {$exists: true}},
           {"mentioned_user_info.name": {$exists: true}},
           {"mentioned_user_info.location": {$exists: true}},
         ]
   }
},
```

[8]	{ 3 fields }
$match	{ 1 field }
nReturned	37
executionTimeMillisEstimate	1536

In this $match stage. It takes 1536 ms and returned 37 documents.

Stage 9:

```
{$group:
   {
       _id: "$mentioned_user_info",

   }
},
```

[9]	{ 6 fields }
$group	{ 1 field }
maxAccumulatorMemoryUsageBytes	{ 0 fields }
totalOutputDataSizeBytes	17741
usedDisk	false
nReturned	13
executionTimeMillisEstimate	1536

In this $group stage, it causes **totalOutputDataSizeBytes: 17741**, which is a huge cost of memory, this stage takes 1536 ms and return 13 documents.

Stage 10:

```
{$sort:
     {"_id.followers_count":-1}
},
```

[10]	{ 5 fields }
$sort	{ 2 fields }
totalDataSizeSortedBytesEstimate	13815
usedDisk	false
nReturned	5
executionTimeMillisEstimate	1536

In this $sort stage, it causes **totalOutputDataSizeBytes: 13815** Bytes, this stage takes 1536 ms and return 5 documents.

Stage 11:

```
{$project:
   {
     _id: 0,
     id: "$_id.id",
     name: "$_id.name",
     location: "$_id.location",
     followers_count: "$_id.followers_count"
   }
```

[11]	{ 3 fields }
$project	{ 5 fields }
nReturned	5
executionTimeMillisEstimate	1536

In this $project stage. It takes 1536 ms and returned 5 documents.

Q4 Alternative:

```
db.users.aggregate(
```

Refer to the first question about why use aggregate here.

Index:

```
db.users.createIndex({id:1})
```

This method is based on users collection. And I will just analyze some key different stages

Stage 1:

```
{$lookup: {
     from: "tweets",
     localField: "id" ,
     foreignField: "user_mentions.id",
     as: "tweets_mentioned_me"}
},
```

[1]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	1932
totalKeysExamined	1932
collectionScans	0
indexesUsed	[1 element]
nReturned	8328
executionTimeMillisEstimate	454

In this $lookup stage, it examined 1932 key documents, **with using indexing on "id"**, returned 8328 documents and take 454ms.

Stage 9:

```
{$group:
   {
       _id: "$id",

   }
},
```

[9]	{ 6 fields }
$group	{ 1 field }
_id	$id
maxAccumulatorMemoryUsageBytes	{ 0 fields }
totalOutputDataSizeBytes	2561
usedDisk	false
nReturned	13
executionTimeMillisEstimate	454

In this $group stage, it causes **totalOutputDataSizeBytes: 2561** Bytes, this stage takes 454 ms and return 13 documents.

Stage 10:

```
{$lookup: {
     from: "users",
     localField: "_id" ,
     foreignField: "id",
     as: "user_info"}
},
```

[10]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	13
totalKeysExamined	13
collectionScans	0
indexesUsed	[1 element]
nReturned	13
executionTimeMillisEstimate	455

In this $lookup stage, it examined 13 key documents, **with using indexing on "id"**, returned 13 documents and take 455ms.

Stage 11:

```
{$sort:
     {"user_info.followers_count":-1}
},
```

[11]	{ 5 fields }
$sort	{ 1 field }
totalDataSizeSortedBytesEstimate	9593
usedDisk	false
nReturned	5
executionTimeMillisEstimate	455

In this $sort stage, it causes **totalOutputDataSizeBytes: 9593** Bytes, this stage takes 455 ms and return 5 documents.

And according to the time and memory result, this alternative way is better, which takes less execution time and less memory.

Q5:

Find the number of general tweets published by users with neither location nor description information.

```
db.tweets.explain("executionStats").aggregate(
  [
```

Refer to the first question about why use aggregate here.

Index:

```
db.users.createIndex({id:1})
```

Stage 0:

```
{$match:
      {$and:[{replyto_id: {$exists: false}}, {retweet_id: {$exists: false}}]}
},
```

$cursor	{ 2 fields }
nReturned	1641
executionTimeMillisEstimate	6

In this $match stage. It takes 6 ms and returned 1641 documents.

Stage 1:

```
{$lookup: {
    from: "users",
    localField: "user_id" ,
    foreignField: "id",
    as: "user"}
},
```

[1]	{ 7 fields }
$lookup	{ 7 fields }
totalDocsExamined	13666248
totalKeysExamined	0
collectionScans	3282
indexesUsed	[0 elements]
nReturned	114
executionTimeMillisEstimate	5572

[1]	{ 7 fields }
$lookup	{ 7 fields }
totalDocsExamined	1641
totalKeysExamined	1641
collectionScans	0
indexesUsed	[1 element]
nReturned	114
executionTimeMillisEstimate	112

After **creating index of id in users collection,** execution time decreases 5400ms. And the totalDocsExamined almost decreased 13666000 documents, and the totalDocsExamined actually is the totalKeysExamined. With returning 114 docuements.

```
{$unwind: "$user" },

{$match:
    {$and:[{"user.location": ""}, {"user.description": ""}]}
},
```

These two stages disappear in the explain.

Stage 2,3:

```
{$count: "tweet_count"}
```

[2]	{ 6 fields }
$group	{ 2 fields }
maxAccumulatorMemoryUsageBytes	{ 1 field }
totalOutputDataSizeBytes	229
usedDisk	false
nReturned	1
executionTimeMillisEstimate	112
[3]	{ 3 fields }
$project	{ 2 fields }
nReturned	1
executionTimeMillisEstimate	112

In this $count stage, this command include two steps, we can obviously find by explain(). The first step is $group and the second step is $project. In the $group stage, **there is a totalOutputDataSizeBytes parameter which represents the memory usage the stage take which now take 229 Bytes.** The nReturned parameter represents the data finally showed on the interface, here the one data is our target result. In this $project stage. It takes 112 ms and returned 1 documents.

Q6

Find the general tweet that receives most retweets in the first hour after it is published. Print out the tweet Id and the number of retweets it received within the first hour.

```
db.tweets.explain("executionStats").aggregate(
  [
```

Refer to the first question about why use aggregate here.

Index:

```
db.tweets.createIndex({retweet_id:1})
```

Stage 0:

```
{$match:
    {$and:[{replyto_id: {$exists: false}}, {retweet_id: {$exists: false}}]}
},
```

[0]	{ 3 fields }
$cursor	{ 2 fields }
nReturned	1641
executionTimeMillisEstimate	9

In this $match stage. It takes 9 ms and returned 1641 documents.

Stage 1:

```
{$lookup: {
  from: "tweets",
  localField: "id" ,
  foreignField: "retweet_id",
  as: "retweets"}
},
```

[1]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	16410000
totalKeysExamined	0
collectionScans	3282
indexesUsed	[0 elements]
nReturned	1641
executionTimeMillisEstimate	5944

[1]	{ 7 fields }
$lookup	{ 4 fields }
totalDocsExamined	892
totalKeysExamined	892
collectionScans	0
indexesUsed	[1 element]
nReturned	1641
executionTimeMillisEstimate	105

After **creating index of id in users collection,** execution time decreases 5800ms. And the totalDocsExamined almost decreased 16410000 documents, and the totalDocsExamined actually is the totalKeysExamined. With returning 1641 docuements.

Stage 2:

```
{$match:
    {$expr:
        {$gt:[{$size: "$retweets"}, 0]}
    }
},
```

[2]	{ 3 fields }
$match	{ 1 field }
nReturned	243
executionTimeMillisEstimate	105

In this $match stage. It takes 105 ms and returned 243 documents.

Stage 3:

```
{$unwind: "$retweets" },
```

[3]	{ 3 fields }
$unwind	{ 1 field }
nReturned	892
executionTimeMillisEstimate	105

In this $match stage. It takes 105 ms and returned 892 documents.

Stage 4:

```
{$project:
    {
      _id: 1,
      id:1,
      created_at: {$toDate: "$created_at"},
      retweet_at: {$toDate: "$retweets.created_at"}
    }

},
```

[4]	{ 3 fields }
$project	{ 4 fields }
nReturned	892
executionTimeMillisEstimate	105

In this $project stage. It takes 105 ms and returned 892 documents.

Stage 5:

```
{$project:
    {
      _id: 1,
      id:1,
      time: {$divide:[{$subtract:["$retweet_at", "$created_at"]}, 60000]},
    }
},
```

[5]	{ 3 fields }
$project	{ 3 fields }
nReturned	892
executionTimeMillisEstimate	105

In this $project stage. It takes 105 ms and returned 892 documents.

Stage 6:

```
{$match:
    {time:{$lt: 60}}
},
```

[6]	{ 3 fields }
$match	{ 1 field }
nReturned	861
executionTimeMillisEstimate	105

In this $match stage. It takes 105 ms and returned 892 documents.

Stage 7:

```
{$group:
    {
        _id: "$id",
        retweet_count: {$sum:1}
    }
},
```

[7]	{ 6 fields }
$group	{ 2 fields }
maxAccumulatorMemoryUsageBytes	{ 1 field }
totalOutputDataSizeBytes	55647
usedDisk	false
nReturned	243
executionTimeMillisEstimate	105

In the $group stage, **there is a totalOutputDataSizeBytes parameter which represents the memory usage the stage take which now take 55647 Bytes.** The nReturned parameter represents the data finally showed on the interface here is 243 documents, and take 105 ms this stage

Stage 8:

```
{$sort:
  {retweet_count:-1}
},
```

[8]	{ 5 fields }
$sort	{ 2 fields }
totalDataSizeSortedBytesEstimate	0
usedDisk	false
nReturned	1
executionTimeMillisEstimate	105

In this $sort stage. It takes 105 ms and return 1 documents. But it cost **0 Bytes memory.**

Stage 9:

```
{$project:
    {
      _id: 0,
      id: "$_id",
      retweet_count:"$retweet_count"
    }

}
```

[9]	{ 3 fields }
$project	{ 3 fields }
nReturned	1
executionTimeMillisEstimate	105

In this $project stage. It takes 105 ms and returned 1 document.

My alternative way of question is better. Taking less time and less memory.

Conclusion

1. Memory taking happened in $sort and $group
2. the stages do not run in parallel and the given execution times are estimates, not actual execution times.
3. We only need to create the index of the foreignField in the $lookup command, localField indexing is nothing help