Final Report

Github repo:

https://github.com/CS682-S22/dsd-final-project-Jennytang1224

Background:

This report is to analyze the performances in the Kafka project. The kafka project I previously built was a sub-pub system with partitioning mechanism: producer produces data to a load balancer, based on partitioning mechanism of choice, the load balancer will calculate the brokerID and PartitionID accordingly, then send data to brokers to store. On the consumer side, consumers will subscribe to a specific topic of the data and also be providing the starting position of the data. Depending on if the broker is push-based or pull-based, it will have different algorithms to send data to the consumers. This Project is based on my curiosity on how data and partitioning affect program performance.

Setup Details:

1. I used System.currentTimeMillis() to time the program.

2. To time how much time load balancer is processing data from producer and calculating partition # and broker #. The runtime includes the producer reading data from the file line by line and sending each line of data to load balancer and until load balancer finishes calculating corresponding brokerID and partition ID, and then sending the last piece of data.

3. To determine how much time a consumer is subscribing data and receiving data, the runtime includes from subscribing the first piece of data, to finishing receiving the last piece of data.

4. I'm running each experiment on Virtual Machine for 3 rounds for each experiment and taking the average score to present in the tables and graphs.

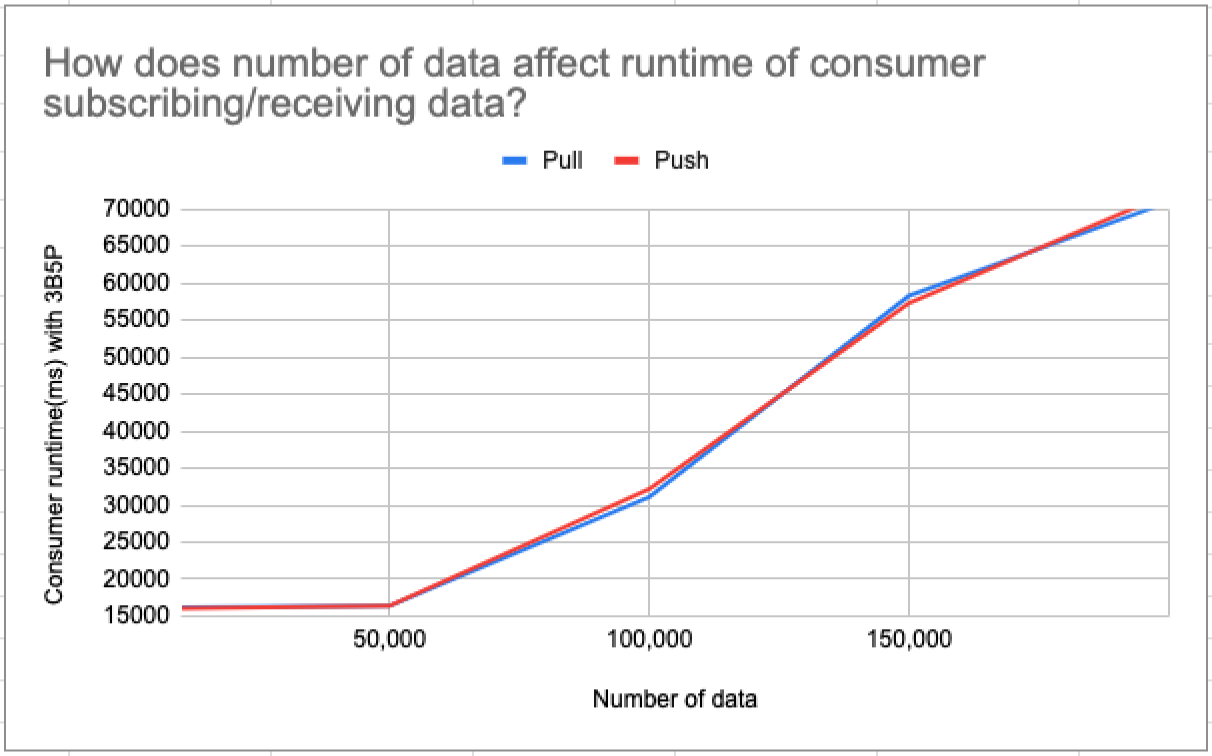
Data Analysis and Visualization:

I implemented this project on my local machine and then tested the performance on assigned virtual machines. I've also read research papers and articles online to have more insights about how kafka partitions and brokers work and compared my results with their assumptions.

Below are my data analysis and visualization:

1. How does the *number of data* affect *runtime of consumer* subscribing/receiving data comparing pull-based and push-based?

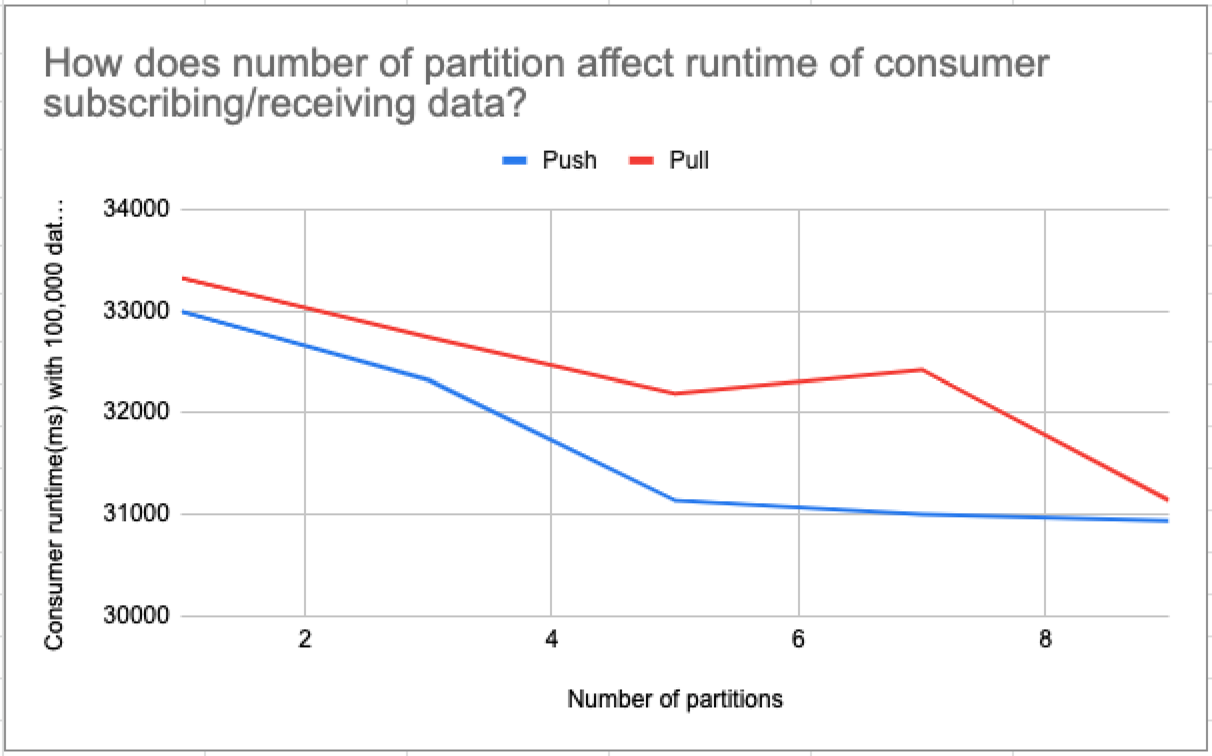
	Consumer runtime(ms) with 3B5P	
Number of data	**Pull**	**Push**
10,000	16186	16102
50,000	16378	16401
100,000	31139	32192
150,000	58343	57342
200,000	70992	72122



Observation:
From the graph above, pull and push doesn't have much differences on consumer runtime given the number of brokers, partitions and data are the same. However, I'm expecting pull is slower than push due to the traffic pull generated to make requests. I think one reason the result is not as I expected was due to the number of data. My assumption is the more data I'm using the bigger the difference we may be able to see. Technically speaking, from what we learned about Kafka, the reason Kafka is using pull-based broker instead of traditional push-based is because different consumers are able to control its own polling speed; however the disadvantage of pull-based was wasting resources due to polling regularly.

2. How does the *number of partitions* affect *runtime of consumer* subscribing/receiving data comparing pull-based and push-based?

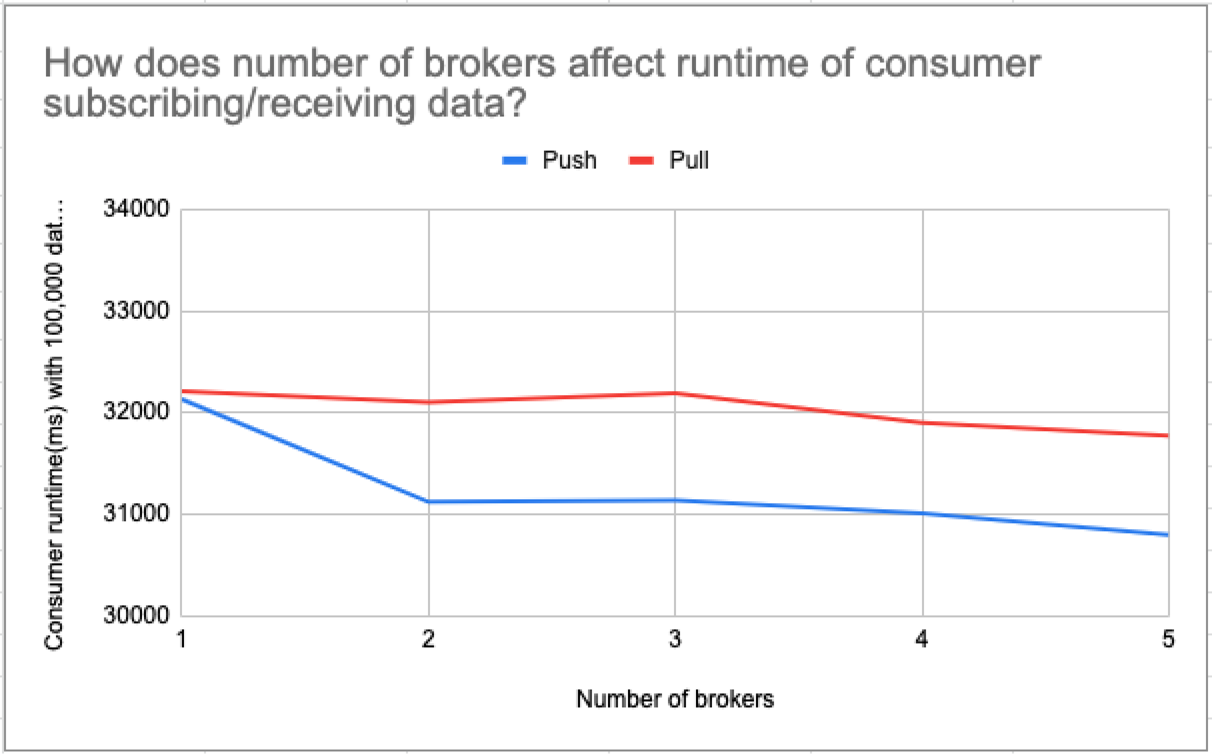
	Consumer runtime(ms) with 100,000 data and 3B	
Number of partitions	**Push**	**Pull**
1	32991	33322
3	32321	32739
5	31139	32192
7	31002	32421
9	30937	31141



Observation:
In general, with the number of partitions increasing, both push and pull decreases in consumer runtime, also pull needs a little more time than push here since it generates more traffic. I read some research paper about kafka performance, it mentioned a concept called “data-rate-per-partition”, which is the rate at which data travels through the partition. By having more partitions, we increase the parallelism of data consumption, which scales up the number of data fetched per request. More partitions will alleviate the number of requests(traffic), and the messages are delivered in a bigger batch, this assumption is reflected in the graph. The more partitions in kafka, the higher throughput one can achieve.

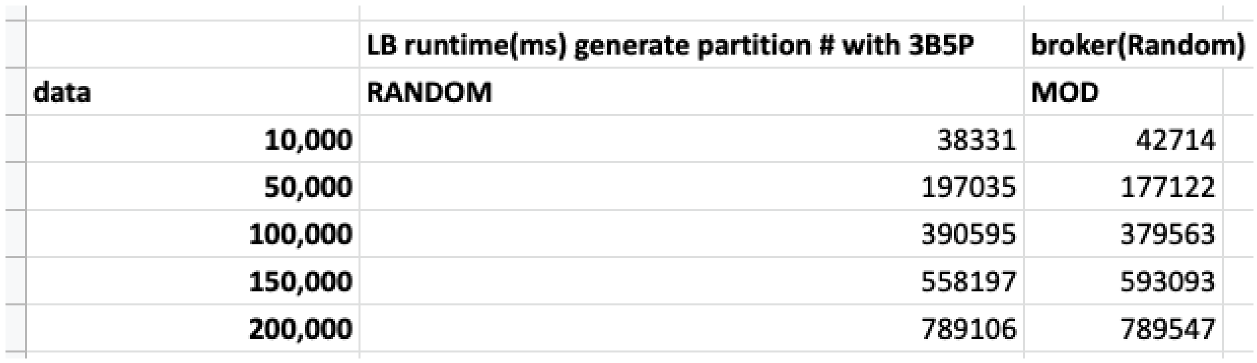
3. How does the *number of brokers* affect *runtime of consumer* subscribing/receiving data comparing pull-based and push-based?

	Consumer runtime(ms) with 100,000 data with 5P	
Number of brokers	Push	Pull
1	32135	32214
2	31124	32105
3	31139	32192
4	31012	31902
5	30799	31775

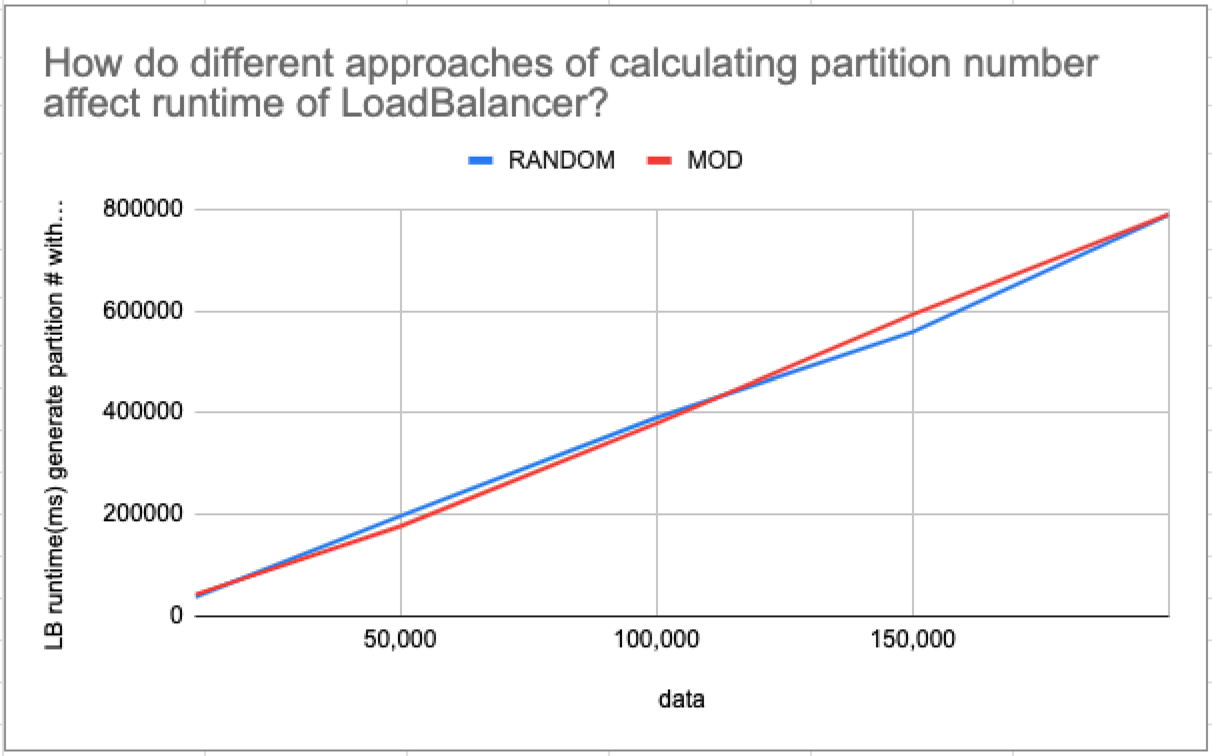


Observation:
Same reason as we discussed in partitions above, increasing the number of brokers can also help improve the performance since it will also increase the parallelism of data consumption, which scales up the number of data fetched per request. In my implementation, to contact and request brokers for data are parallely happening at the same time with multiple threads. Also push here is slightly better than push as we expected since pull is generating request traffic.

4. How do different approaches of *calculating partition number* affect *runtime of LoadBalancer*?

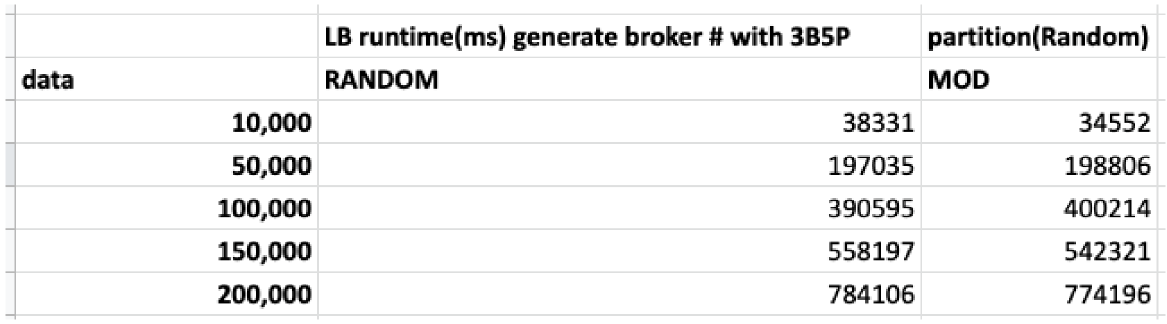


	LB runtime(ms) generate partition # with 3B5P	broker(Random)
data	RANDOM	MOD
10,000	38331	42714
50,000	197035	177122
100,000	390595	379563
150,000	558197	593093
200,000	789106	789547

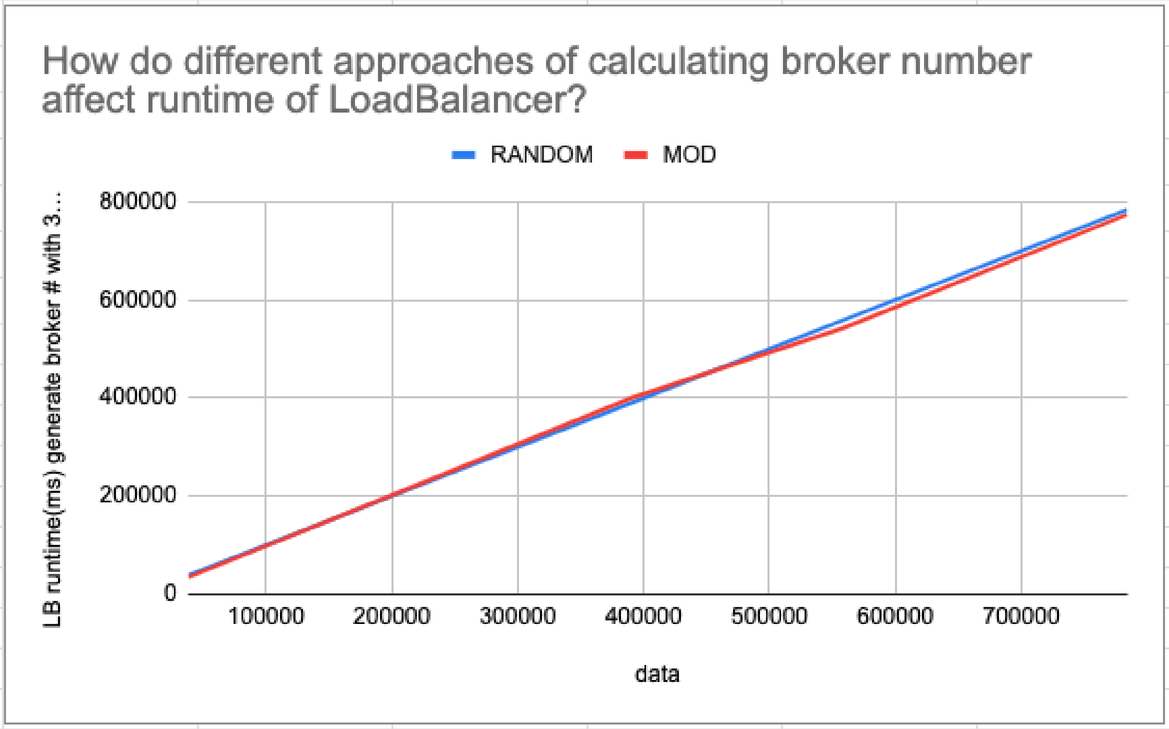


Observation:
Random generation and mod have similar performances. In my mod implementation, I uses the java hashkey() to hash the key first and then use % to get the mod value and use Math.abs() to take the absolute value, therefore the overall calculation is indeed expensive. Meanwhile, the random generation was using java.util.random, which is a relatively expensive operation as well. So they end up having very close performances. Online also mentioned java.util.SplitableRandom gives a faster and better statistical distribution(~30 times faster), I think it could be a good idea to try.

5. How do different approaches of *calculating broker number* affect *runtime of LoadBalancer*?



	LB runtime(ms) generate broker # with 3B5P	**partition(Random)**
data	**RANDOM**	**MOD**
10,000	38331	34552
50,000	197035	198806
100,000	390595	400214
150,000	558197	542321
200,000	784106	774196

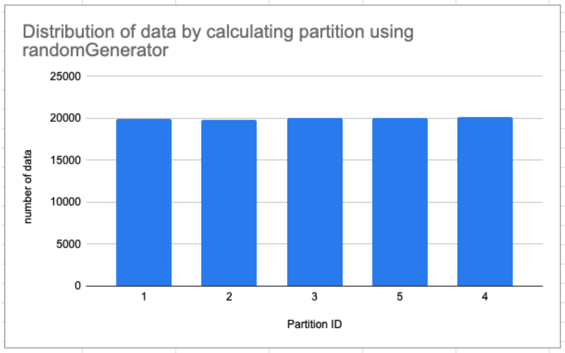


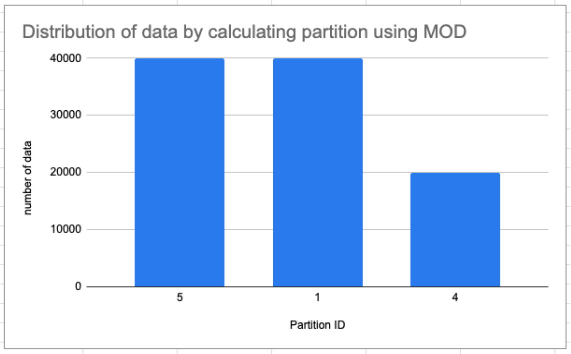
Observation:
Similar to the above experiment, both random and mod have the same runtime for load balancer. The runtime for load balancer includes:

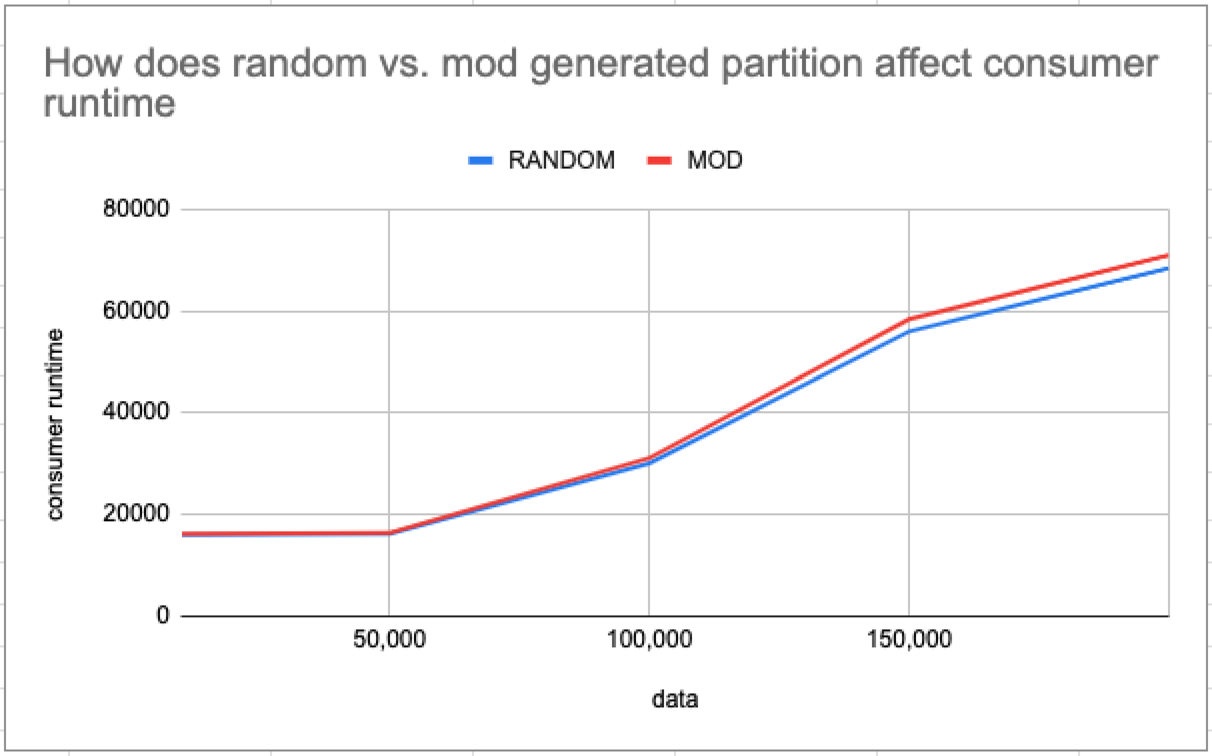
1. the time producer read data line by line from the file,
2. then send data to the load balancer
3. and load balancer calculates partition and broker numbers and sends data to the corresponding broker with its partition.

Given the time to run number 1 and number 2 are the same for different approaches, the graph means they also have similar speed to do the calculations given there are same number of data to send, this also complements the analysis from the previous experiment on partition(mod uses hashkey and % and math.abs while random is an expensive operation in general).

6. Distribution of data by different approaches of *calculating partition number*: (100k data with 3 brokers and 5 partitions, pull)

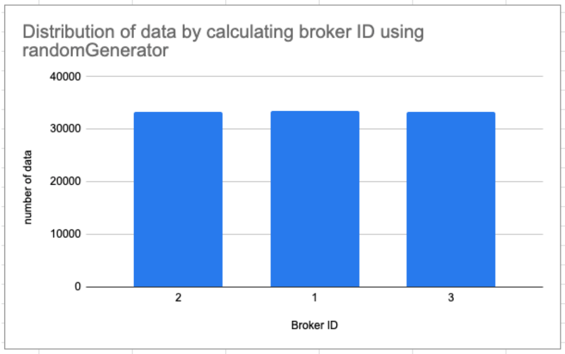


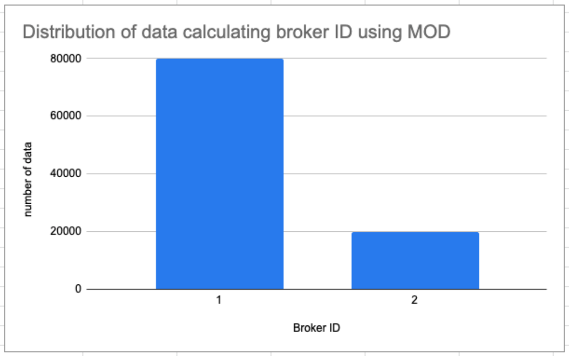


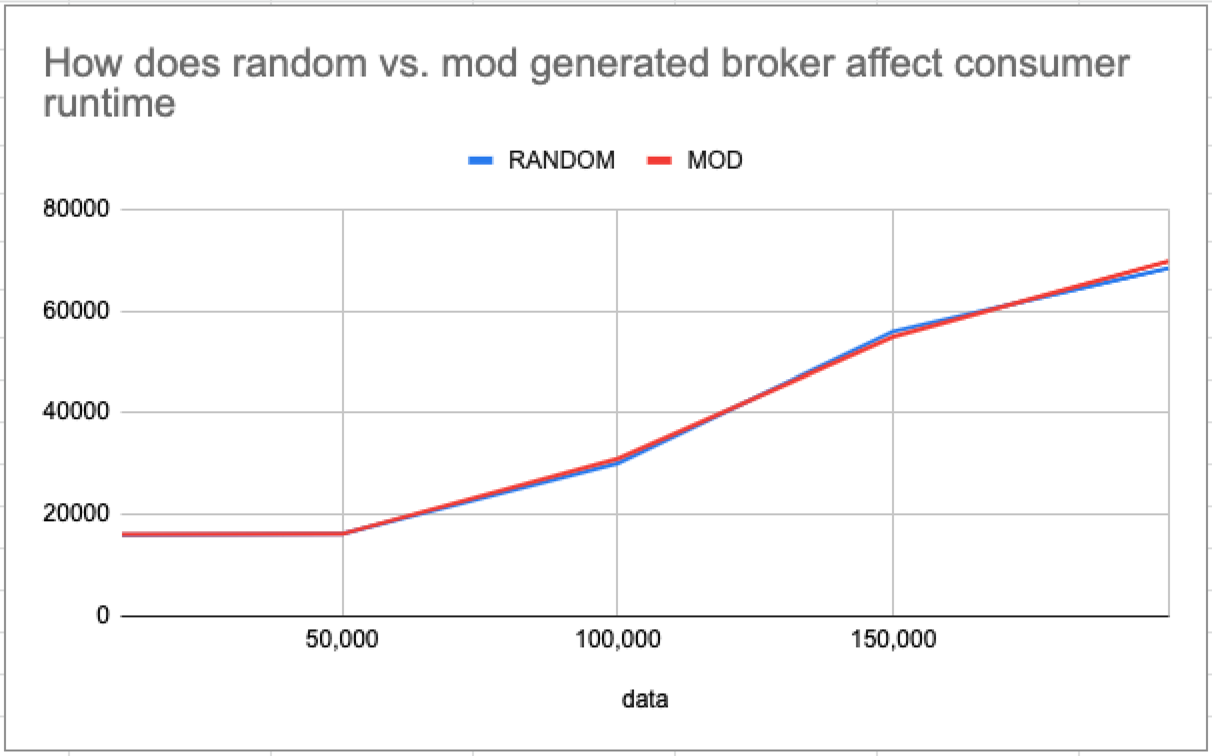


Observation:
Randomly generated partitioning evenly distributes the load of data to each partition compared to using mod. This will provides more balanced performance and make consumers more scalable in theory. In the runtime graph, random has a slightly better performance than mod (less consumer runtime). My assumption is: in each partition inside of a broker for the requested topic, if the number of data in each partition is small it will cause that message to be delivered in a smaller batch. Then consumers may need to make more requests with a new starting position, this increases the consumer runtime. That’s why evenly distributed is better at performances.

7. Distribution of data by different approaches of *calculating broker number*: (100k data with 3 brokers and 5 partitions, pull)







Observation:
Same as partitions, randomly generating broker ID gives a more evenly distributed load of data to consumers. However, In my runtime graph, two of them share the very close performance because the consumer still needs to go to each broker to get data and because my partition here was generated randomly so in each partition array I have quite evenly distributed data. In my implementation, the consumer request brokers for data happen simultaneously via threads instead of sequential requests. Therefore, the runtime for consumer really depends on number of data given we have one consumer

Future work:

In my performance analysis, there are many places I would like to dig into more if I have more time:

1. I also noticed somehow my local machine runs almost 5 times faster than the VM I was using, I think it’s interesting to look into how different machines affect the runtime in general due to different computing power(CPU), and the number of users operating on the machine(traffic), or even disk space can also affect the performances.

2. I would like to experience more data to see when does the exponential growth in some graphs start happening, as well as if there will be more interesting patterns that we don’t expect to see may happen due to the potential background overhead.

3. Number of topics in the data may also affect the runtime since under each broker, there’ll be more maps that map topics to partitions, this could be adding more runtime for going through the map.

Summary:

In conclusion, the biggest takeaways are:

1. By tuning up the number of partitions and brokers, we could potentially improve the performance of kafka because we are increasing the parallelism. The more partitions in kafka, the higher throughput one can achieve.

2. Meanwhile, different ways of generating partition id and broker id will also change the distributions: randomly-generated partitions will give more evenly distributed data among all partitions/brokers compared to mod-generated, this will provide more balanced performance and make consumers more scalable.

3. Push and pull based brokers don’t have huge differences in terms of performance when having a small amount of data, but theoretically pull will generate more request traffic, which could affect the performance negatively.