Cloud Computing - Group 3

Isaiah Pajaro

Vraj Panchal

**K-Means: Locally and in the Cloud**

**Purpose**

Today, we live in a world full of data, with everything being online; data such as website analytics, surveys, customer trends, etc, is everywhere. Modern businesses have become more reliant on this data, and processing it efficiently, making conclusions, and executing business ventures on those conclusions have become ever more critical. However, not all businesses can house vast amounts of data or perform useful transformations on it. As such, cloud-computing-as-a-service has increased, allowing businesses to offload the low-level, behind-the-scenes operations that make data analysis happen. However, we wanted to know if this is the optimal solution and what it would have been like if a business had decided to run the computations themselves, being limited in pure machine processing power. As such, this project is about Traditional Parallel Computing vs Parallel Computing with Cloud. In this scenario, we used Google's BigQuery to emulate a business outsourcing their data work to a cloud provider vs running such tasks on a local computer. We will also look into single vs multi-threaded performance for a more holistic view.

**Background**

The algorithm we settled on using for our test is the K-means algorithm. This algorithm was created in 1957 by Stuart Lloyd as a pulse-code modulation technique, a method used to represent sampled analog signals digitally. He found that he was required to organize the signal space in clusters, so he proposed this algorithm.

The algorithm works like this:

1. Assign your initial centroid points to a random point in a data set.
2. For every other point in the data set, find the distance between each cluster and that point and assign that point to the closest cluster.
3. Reassign the initial cluster to the centroid of the points that belong in the cluster.
4. Repeat steps 2 and 3 until the centroid stops changing.
5. Repeat the algorithm until the variance of the clusters has reached a minimal

After a while, we will reach equilibrium, and the algorithm will be stopped. One thing to notice is that the data transformations can be made independently. As such, it can be parallelized without much issue.

Furthermore, K-means and its use cases are plenty. For eCommerce, businesses use the data from their customers to group them into spending and interest demographics. Often recommending an item that other people (in the same group) looked at or bought. With today's blurred lines in music genres, music platforms use it to group similar songs to put in playlists and see growing music trends. For machine learning, K-means can teach a model object classification.

As you can see, the algorithm is parallelizable and has endless use cases. As such, K-means is the perfect candidate to deploy on the cloud, and this algorithm is something a business would actually deploy. For the following sections, we will test what it is like to run K-means on a local machine vs a cloud instance.

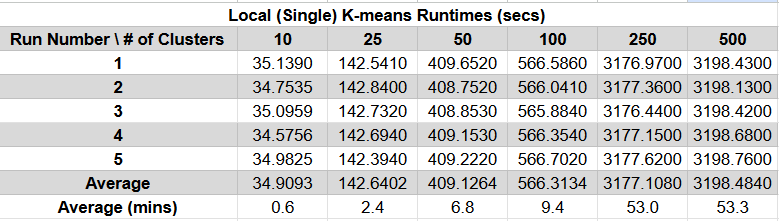
**Local Tests**

Test setup:

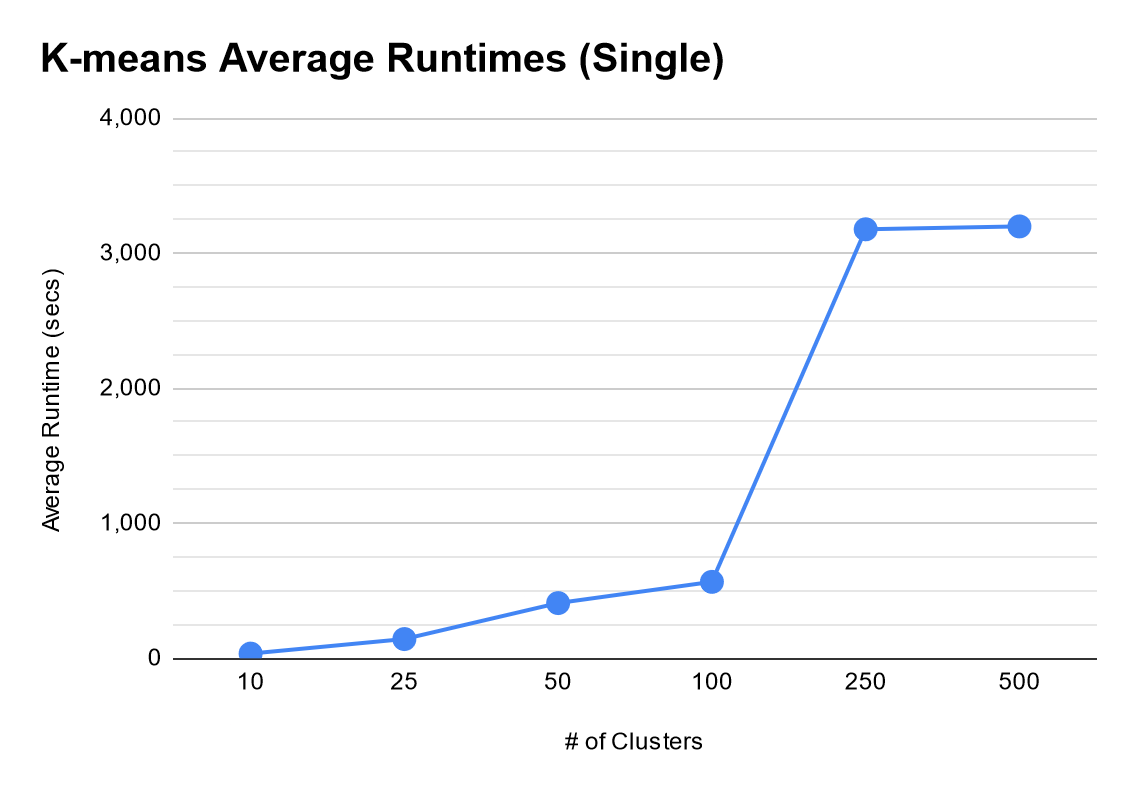
* A desktop with the following specs will represent the local system:
  + AMD Ryzen 7 5600X @ 4.26 GHz
  + 16 GB of DDR4 RAM @ 3200 Mhz
* The dataset we will run K-means on is graph1, a 7 MB file used in the labs from this semester.
* Given that K-means allows the user to determine the number of clusters, we have chosen the # of clusters to be 10, 25, 50, 100, 250, 500
  + These numbers will give a good enough spread for various situations.
* Five runs will be made for each # of clusters, and we will take the average for evaluation.
* The code is found in the submission.
  + The single-threaded run is C++ code of K-means
  + The multi-threaded run is C++ code of K-means with OpenMP.

Evaluation:

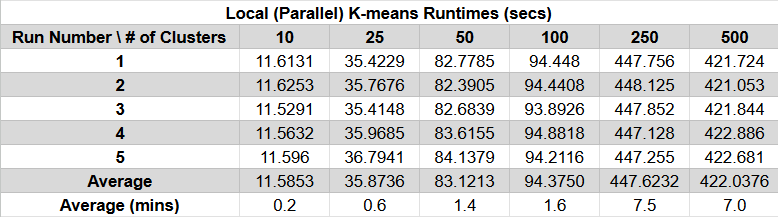
Single-threaded tests up first



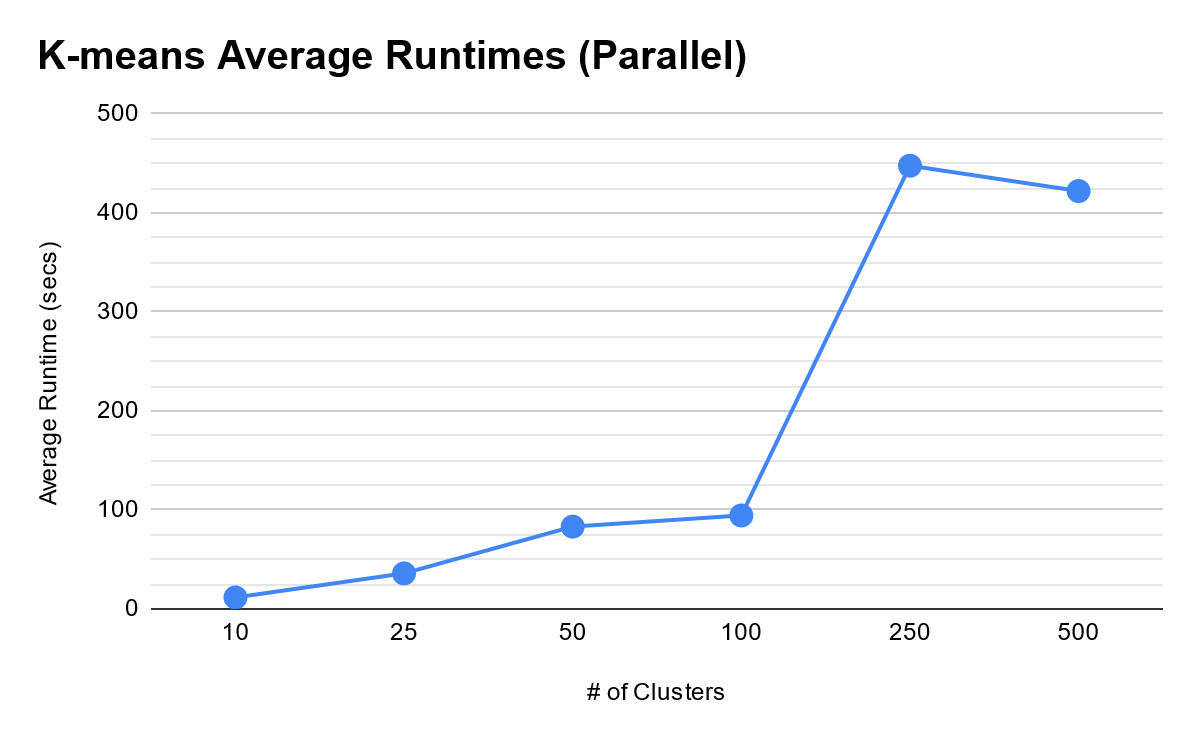
We notice that as the number of clusters increases, so does the runtime. It does so quite dramatically after 100. Here is a graph to visualize the steep jump:



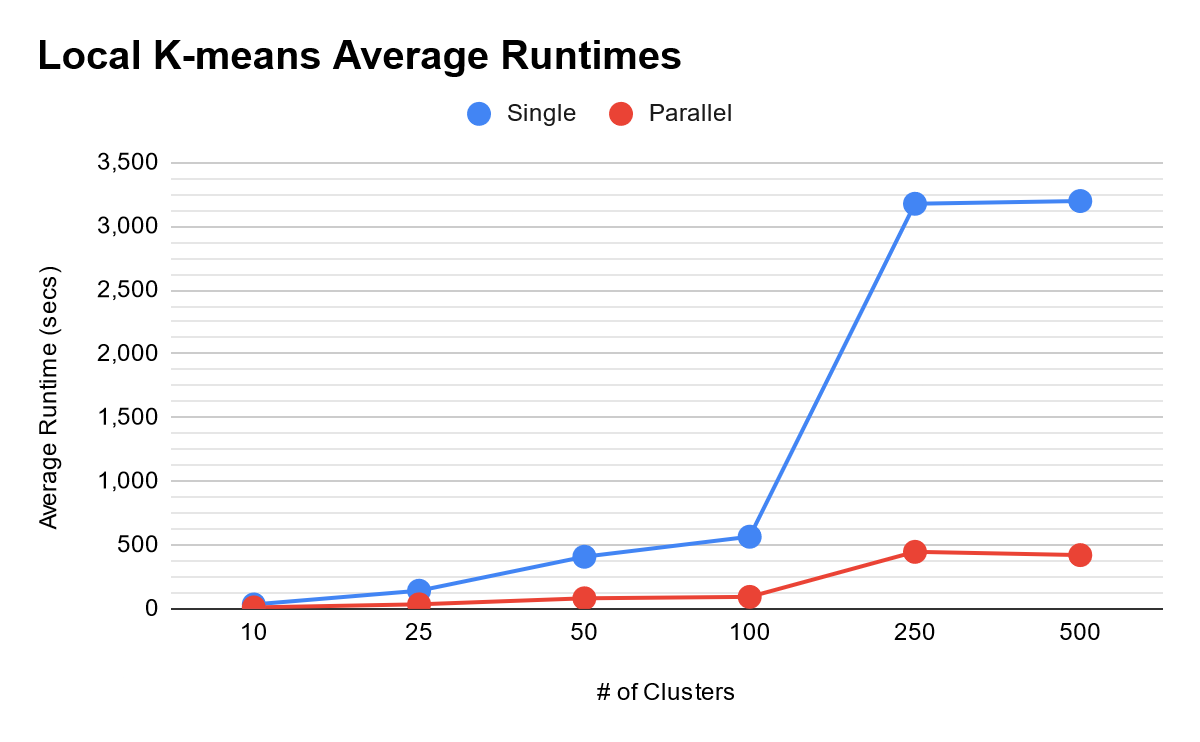
If we attempt the same tests in parallel/multi-threaded code, we get the following results:



We see here that there is a dramatic improvement in runtimes across the board. However, it still keeps the general trend of runtime having a steep jump after 100 clusters:



Regardless, we see a massive improvement across the board with a parallelized workflow:



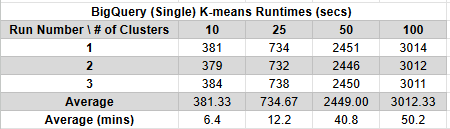
**Cloud Tests**

Test setup:

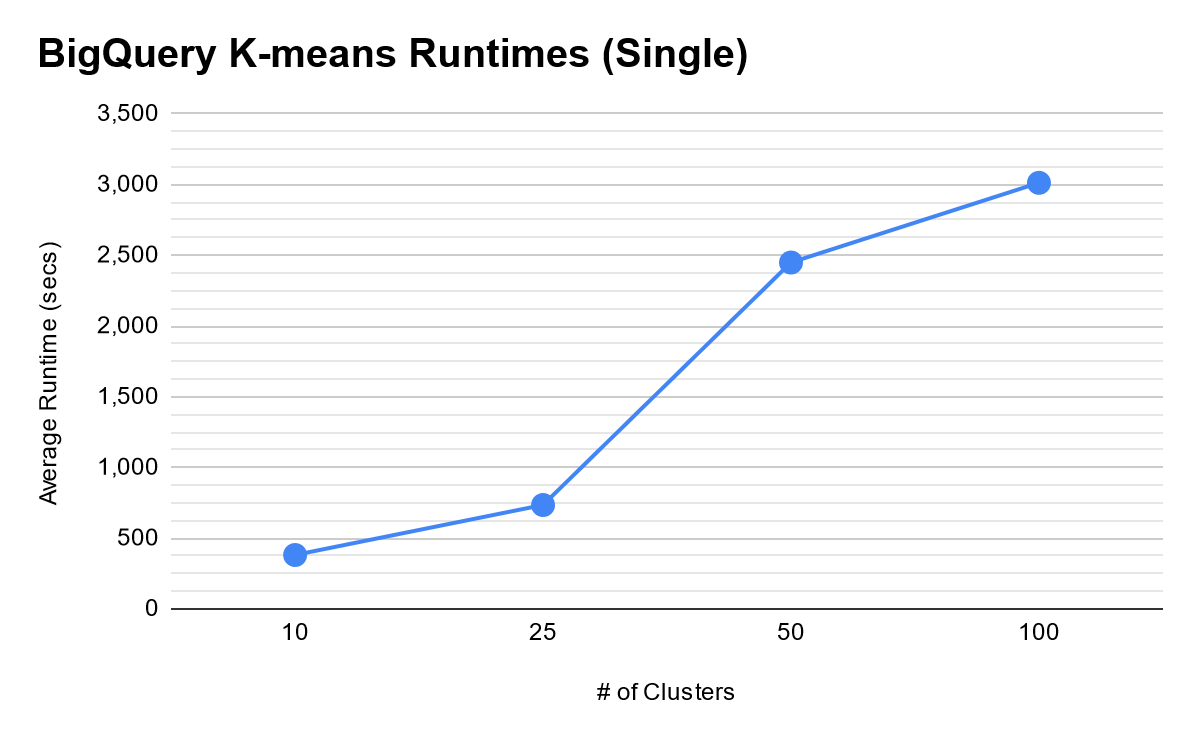
* Google's Cloud Product, BigQuery, will represent the cloud system.
* The dataset we will run K-means on is graph1, a 7 MB file used in the labs from this semester.
* Given that K-means allows the user to determine the number of clusters, we have chosen the # of clusters to be 10, 25, 50, 100
  + Google limits the number of clusters to 100, so 250 and 500 will not be tested.
  + These numbers will still give us usable results.
* Three runs will be made for each # of clusters, and we will take the average for evaluation.
  + Three runs are taken here instead of 5 like previously due to costs. Google Cloud charges for runtime down to the nearest second.
* Code is written in BigQuery notation, based on SQL statements.
  + The single-threaded data is provided by Google by adding up single-threaded compute time across all devices involved in the computation. Google shows this metric to show "how much time you would have used if done sequentially."
  + The multi-threaded data is provided by Google and is the time from sending a job to the global queue until execution is complete. This is the time Google uses to determine how much to charge.

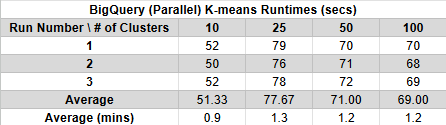
Evaluation:

Single-threaded tests up first:

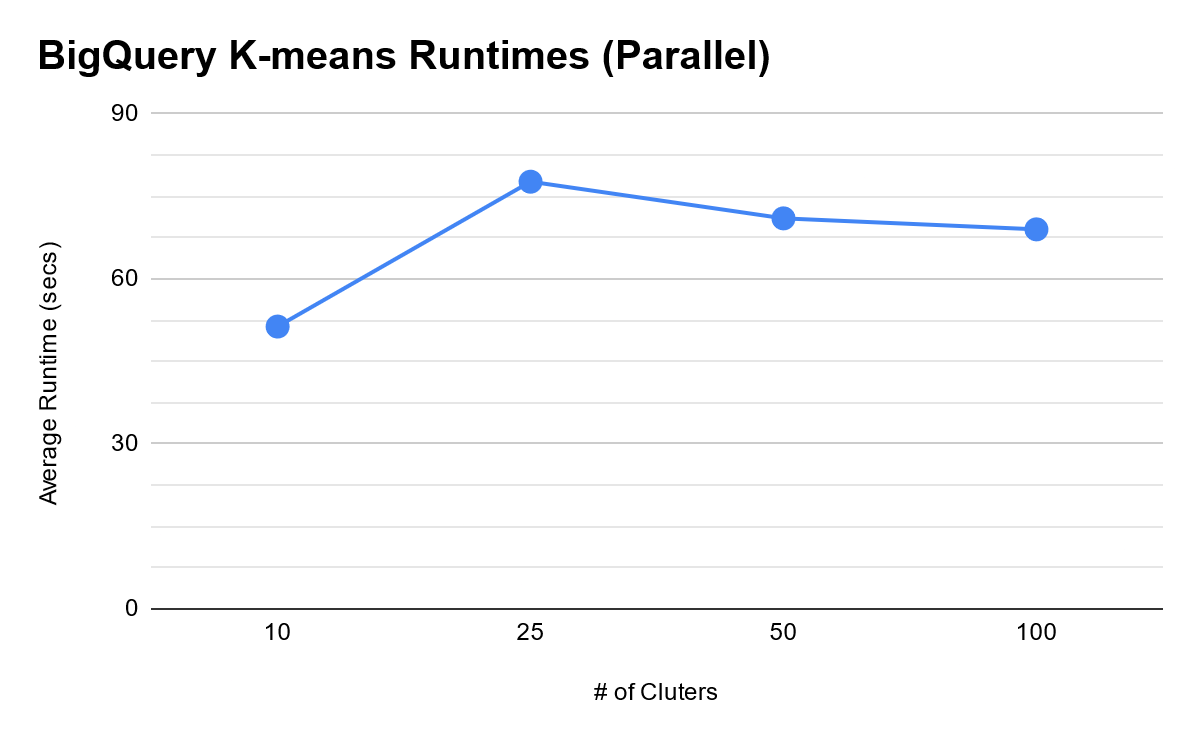


We notice that as the number of clusters increases, so does the runtime. It does so relatively quickly after 25. Here is a graph to visualize the jump:

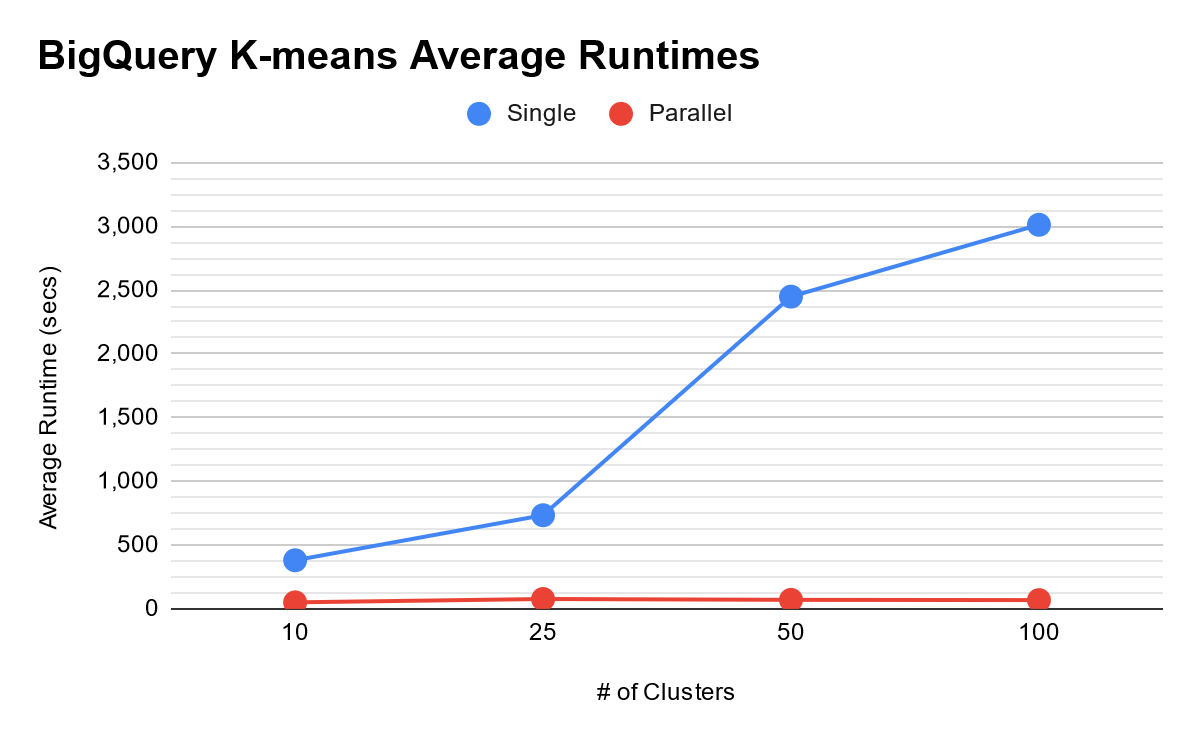
If we attempt the same tests in parallel/multi-threaded code, we get the following results:



We see here that there is a dramatic improvement in runtimes across the board. However, what is interesting is that that runtime generally stays flat and reaches a pseudo-equilibrium. As such, the data points are relatively close in time, which is noticeable after 25 clusters:



We can deduce that Google's implementation of K-means is well-optimized and does not seem affected by cluster size. Instead, there is a lengthy overhead time before the algorithm computes. Regardless, we see a massive improvement across the board with a parallelized workflow:

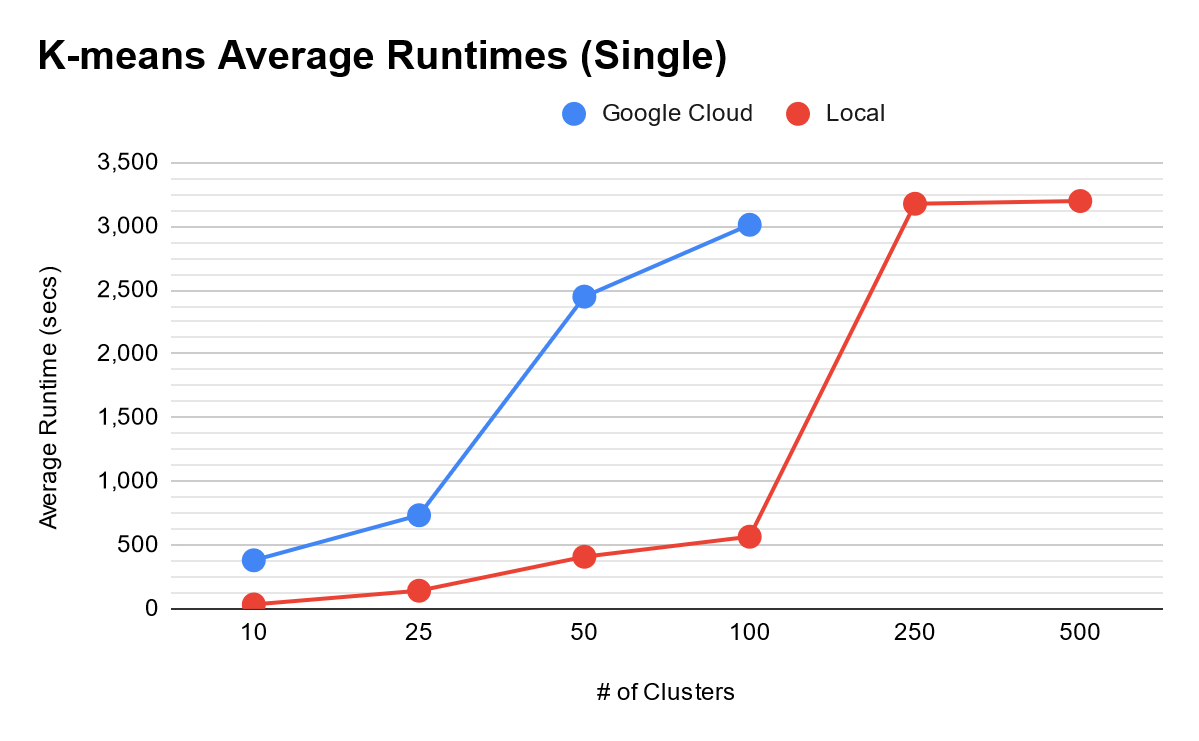


The discrepancy is so stark that the parallel data looks flat compared to its single counterpart in BigQuery.

One thing we must discuss when using cloud-computing-as-a-service is the pricing model—Google charges for storing data and running computations on it. As of writing, storage costs $0.02/GB per month. Computations are $0.10/hour. These tests were done on a 7MB file for 14 minutes, equating to about $0.03 to get these data points. This cost seems low, but it is a consideration for a business looking to scale such computations to large datasets. However, the cost may be worth it for cloud-computing-as-a-service advantages, such as easier management or quick deployment.

**Local vs Cloud; Single vs Parallel**

Aggregating the data from both tests, we get the following results:



We notice here that the cloud implementation is slower than the local implementation. This is different from what we initially expected, but a deeper analysis clarified the situation, and it comes down to how Google sets up cloud jobs.

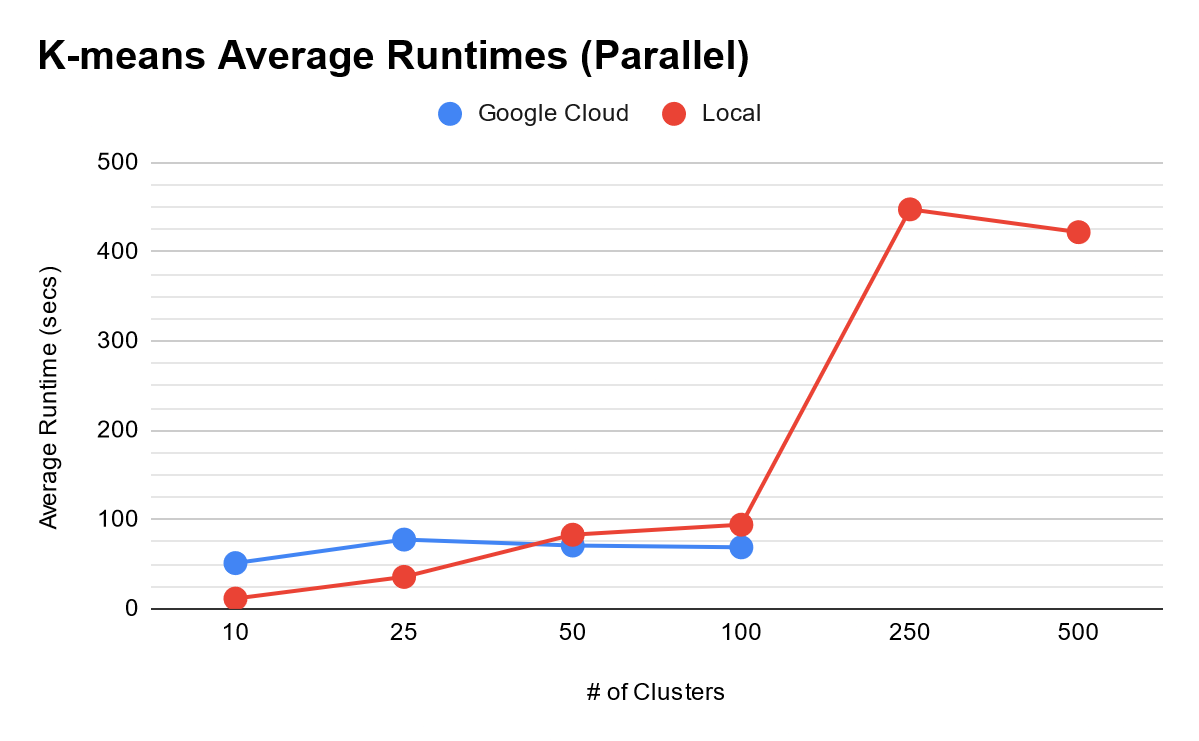
When starting a job, Google:

* Loads the data into its servers
* Adds it to the global queue
* Waits for the next available worker to take the job

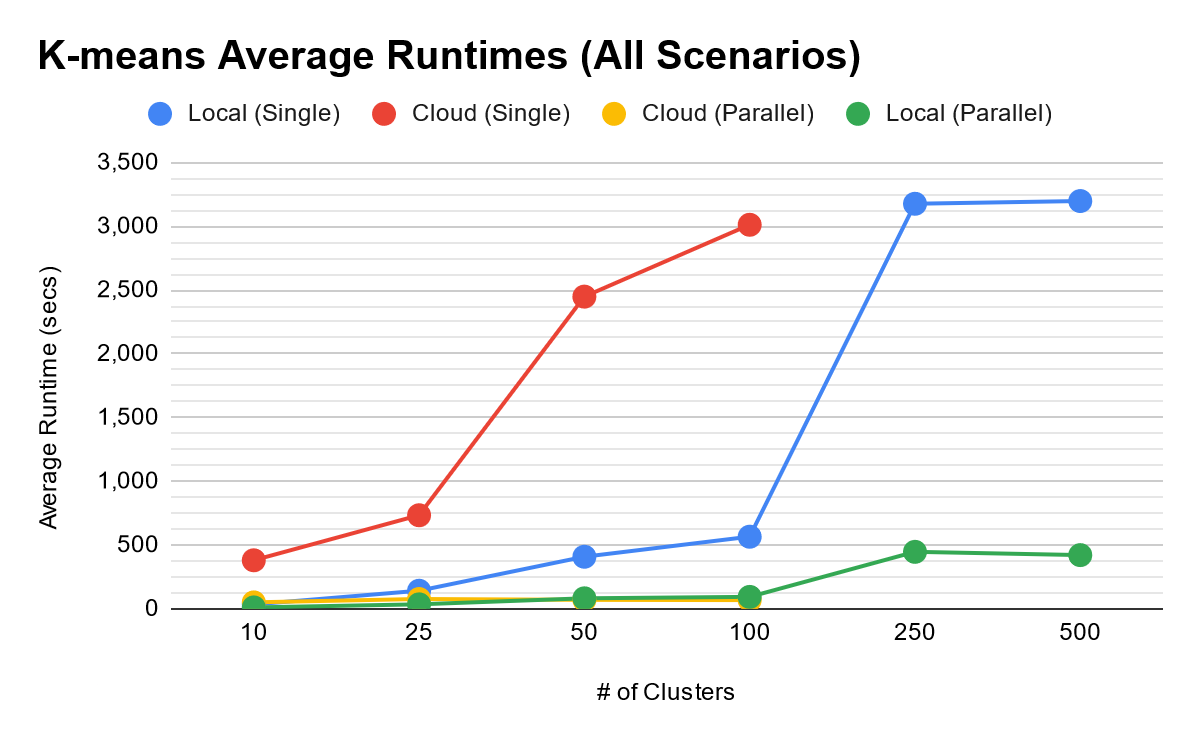
So we spend much time waiting, and when we do get a chance, we are computing with a single thread (which is slow). Meanwhile, the local instance can get right to work.

Thankfully, the default mode in BigQuery is multi-threaded / parallel computing. Note: We assume that the single-threaded cloud implementation would be faster than its local counterpart if the dataset was large enough, and compute performance would play a more significant role than loading data.

Looking at the parallel results, we see a similar trend. The cloud implementation is slower than the local implementation, and we made the same observations in the single-threaded comparison for lower cluster counts/computational load. However, once the computational workload ramps up and we hit 50 clusters, the cloud implementation overtakes the local implementation. With a more extensive data set that requires more computations, the cloud implementation would be able to flex its muscles and widen the gap even more. It would be ideal to see if this holds for 250 and 500 clusters, but 100 is the limit set in place by Google.



Here is everything overall:



What we can notice across the board is that parallel implementations are better than single-threaded implementations as long as the increased throughput is greater than the context switching.

**Conclusion**

Cloud-computing-as-a-service is growing and has the means to provide businesses value at a cost. As we saw here, it can be used to run helpful business analytical workflows, such as K-means on customer data. It also excels if the workload is heavy and parallelizable, a characteristic of the big data that businesses deal with daily. Overall, cloud-computing-as-a-service is likely to stay as it can do the above, all while being quick to deploy, having customer support, and not having to manage the low-level workings of servers.