Processing of Streaming Data Using AWS Infrastructure Services

Team 7: Red Octopus

Anshuman Goel Rachit Thirani Vivek Varma Nadimpalli

CSC 591 Data Intensive Computing

December 5, 2017

Abstract

This paper discusses about building a pipeline by utilizing the Amazon Web Services (AWS) infrastructure services. The application streams Twitter data and write high volumes of data into Kinesis stream with varying velocities. The other part of application reads data from Kinesis shards and saves it in DynamoDB continuously while doing processing on the data. To analyze the data stored in DynamoDB, AWS Elastic Map Reduce (EMR) is used in batches whenever required. Data is pulled from DynamoDB, saved to HDFS storage and then analyzed. The primarily analysis performed on the data was to get the most common name in Twitter who tweets. Prominently Hive, Pig and Spark are installed in EMR instances to analyze the data and some execution parameters. The main challenge was to make a data intensive application. Streaming Twitter doesn't give high volume of data therefore data was being replicated multiple times with load balancing over all the Kinesis shards.

1 Introduction

The aim is to perform data intensive computing in a distributed environment. AWS provides lots of cloud-computing services which are highly scalable, reliable and available throughout the globe. An application has been developed which utilize various AWS infrastructure services to analyze massive amounts of data faster than the traditional systems and also lowers number of man hours and cost.

This particular application retrieves data from TwitterAPI. However, due to limitations of this API, it does not allow heavy streaming of data. Therefore, application replicates the data and feeds that data into Kinesis shards. The other consumer on the other hand reads data from these shards and store it persistently in DynamoDB which is a NoSQL Document type database. To perform some kind of analysis on data in regular batches, data is being copied from DynamoDB to HDFS and analyzed using Hive.

1.1 Amazon Kinesis

Amazon Kinesis is an AWS streaming service used to collect and process data in real-time as mentioned in [1]. It helps to process large volumes of streaming data cost effectively. The data can be ingested and pushed to database of data warehouses or utilized to build any real time application. Amazon Kinesis helps in analyzing data on the fly only if it is below the reading capacity of the shard. As one can write into shard at maximum of 1 MB/s and can read from shard at maximum of 2 MB/s.

Kinesis has data streams in which each record has a unique sequence number. Records are distributed across shards. The sequence number provided by the Kinesis steam has been used as the unique identifier for storing the records in the DynamoDB to avoid over-writing of existing records in a key value database. A shard can be thought as a queue where the records are being placed by the producers and consumers can consume those records in our practice. However, shard in [2] has been identified as a group of records. As discussed above there is certain limit for writing and reading from the shard which can lead to bottleneck in some cases. Therefore a careful consideration needs to be in order to use for on the fly consumption or some appropriate design should be chosen. Partition Key determines shard id where the Kinesis will put the record. It is done by calculating MD5 hash. The value of hash determines in which shard the data will be placed. [2]

Pricing in Kinesis is based on the number of shards reserved in the stream.

1.2 Amazon DynamoDB

DynamoDB is a NoSQL cloud database service provided by AWS, which supports key-value and document store models, with latency in milliseconds even at large scale. Its flexible data model, reliable performance, and automatic scaling of throughput capacity, makes it a great fit for low latency application like gaming. It is also highly scalable, a requirement in the industry. [4]

Writing data to DynamoDB can also be complex task like Amazon Kinesis. Just like the limits on reading and writing in the shards, the DynamoDB has capacity units for reading and writing. One has to specify the proper number of capacity units such that DynamoDB doesn't throttle down due to heavy number of requests. The reading capacity units can be read at the speed of 20 to 40 KB/s depending upon the consistency model and can write at a speed of 5 KB/s. [5]

Just like Kinesis the pricing is also based on the number of capacity units. Moreover, AWS allows DynamoDB to auto-scale these capacity units. However, the writing or reading capacity of one capacity unit is very low. A typical application will require thousands of such units to work well and efficiently. Also, storing data permanently can be quite expensive. Storing 1 TB of data can easily cost approximately USD 170 per month [6].

1.3 Amazon Elastic Compute Cloud (EC2)

Amazon EC2 provides VM machines as per the client requirements. Just with few clicks on Amazon web console one can create an EC2 instance and can connect to that that instance through ssh protocol easily. It helps to reduce time in procuring new web servers and also

the cost. Amazon spot instances for EC2 provides really cheap access for student developers to use and learn them. [7]

However, due to security mechanism one must be acquainted with SSH and certificates. Since, AWS allows only secure and authenticated connection to EC2 instances. However, the key can be easily created and saved into the local machine for the accessing the EC2 instances anytime. It is always recommended to add a pass-phrase for your key to avoid any misuse of the key.

1.4 Amazon Elastic Map Reduce (EMR)

Amazon EMR processes big data across a Hadoop cluster with the ability to dynamically resize the resource use based on the demand. It can also interact with the data in stored DynamoDB and Amazon S3 also to run the map-reduce functions on the data. EMR also provides variety of frame works to perform the Map Reduce Task such as Spark, Hive, Pig Latin. [9]

EMR basically creates number of EC2 instances and number of softwares related to Big Data technologies installed as mentioned in the web console by the user. It greatly reduces the stress of setting up the cluster environment and installation of the various packages and softwares. For the current package, Hadoop, Hive, Pig and Spark are chosen for various testing and implementation purposes. Due to the size of the data that we are processing is quite large we prefer to choose the machine which has large amount of storage available and is cheap too. Thus, we land up in choosing *m1.xlarge* machines for master and other slave nodes. According to [8], m1.xlarge provides 4 vCPU's with 15 GB RAM and 420 GB of SSD storage. *m1.xlarge* are mostly general purpose CPU with RHEL operating system installed in it mostly. It is also known to provide high network performance which is also a critical part of our system while replicating the data in HDFS and also while performing the processing on data for Map and Reduce tasks.

1.5 Apache Tez

Amazon AWS uses *Apache Tez*. *Tez* is a frame built on top of *Yarn* which generalizes the MapReduce paradigm to a more powerful framework based on expressing computations as a data flow graph. *Tez* aims to address the broad spectrum of use cases in the data-processing domain in Hadoop, ranging from latency to complexity of the execution. It gives a better execution performance.

Tez API has a Directed Acyclic Graph (DAG) which correspond to one object. It defines the overall job. Vertex defines the user logic along with the resources and the environment needed to execute the user logic. One Vertex corresponds to one step in the job. Edge defines the connection between producer and consumer vertices.

Tez is flexible and can run MR jobs without significant modification. It converts the multiple MR jobs to MRRjobs, i.e. a single map with multiple reduce stages. The benefits of this approach is that the data can be moved from one process to another process without writing it to HDFS, saving the write and read time, resulting in better performance.

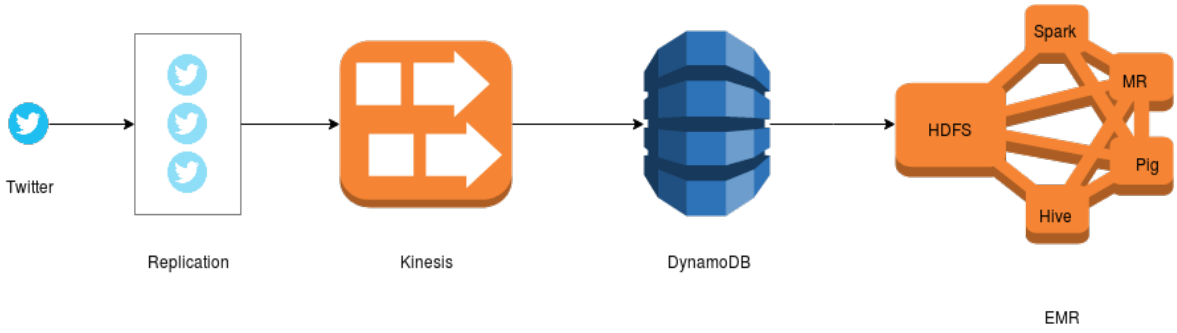

Figure 1: Architecture

1.6 Hive

Hive developed by Facebook for querying the data for data-warehouse project over Hadoop as most of the queries in the database were SQL type. It is like SQL which can do query and summarize results and easily be used for analysis. In Hive, the written query uses Java API to build Map-Reduce Java code internally for the SQL. Even Amazon EMR provides support for Hive and is quite popular among users. [11]

Amazon EMR Hive version has been modified to add support for DynamoDB. The Hive query engine determines firstly the reading capacity of the DynamoDB and perform all the reads within it. This ensure that DynamoDB doesn't throttle. However, it can be a bottle neck if data has to read quickly. The only way is to increase the number of reading capacity before initiating Hive query. It is also observed that auto-scaling also doesn't help in this as Hive query engine only reads the number of capacity units in the starting, it doesn't deal into account that reading capacity units can be automatically scaled too or can be scaled up or down manually in between too.

2 Architecture

2.1 Streaming Data from Twitter using Kinesis

AWS Kinesis is used to stream the data from the Twitter servers using the TwitterAPI. TwitterAPI helps in fetching data from Twitter REST API and helps in streaming data easily. However, the data volume generated and received by the TwitterAPI is not large enough. Therefore, a multi-threaded program helps to generate large volumes of data by replicating the data received from Twitter. A thread retrieves the streaming data from Twitter and places that data in a shared buffer, while the other remaining threads read from the shared buffer and continuously push that data into the AWS Kinesis shards. For interacting with Kinesis *boto3* library of *Python* is used just because of easy semantics it provides.

To feed the data into a shard, a hash is automatically calculated from the PartitionKey [2] which results in a low success rate because of collisions. To overcome it, each process on different machines determined their unique shard using ExplicitHashKey [3] to ingest data into that particular shard. This hash is being calculated using the command line parameter

as the process number (not to be confused with process ID). The process number is the specific shard ID where the process the want to sends the data.

The Twitter allows only a maximum of two simultaneous connections for a unique access key which forced us to create multiple applications to generate various unique Twitter API keys. Thus, each set of two process can use a given Twitter API key.

To increase the volume of data with varying velocity, the alternative approach without self-duplicating the data will require a lot of processes and each one requiring a Twitter API. Thus, creating a large amount of Twitter API keys. Managing that large number of keys will itself be a task and is also completely beyond the scope of the system. Thus, a smart approach is adopted to replicate the data and writing the data in chunks in Kinesis shards. The main producer thread of the process will continuously fetch tweets with their meta-data from Twitter in every call and as soon as the buffer is filled it will start deleting the objects from the buffer to make space for new tweets.

The other threads will just copy the data in chunks of 10 tweets and push those 100 tweets into the Kinesis stream. Also writing into Kinesis in chunks helps to optimally use the shards at the maximum capacity as and when required. This mechanism helps in duplicating the data which allows large volume of data at different velocities.

The data was generated at about 800 MB/min with a total of 11 process at 11 different machines each with 10 replication threads, a shared buffer of size/ length 15000. A total of 11 shards were created in AWS Kinesis. Eleven of these streaming programs ran simultaneously on different machines including NCSU VCL machines and AWS EC2 instances. Each process would generate and push data into one of the eleven AWS Kinesis shards.

To save the cost but at the expensive of maintaining the infrastructure, Kafka can also be also be used to perform streaming [20]. However, due to limited number of VCL machines AWS Kinesis is preferred over Kafka. Moreover, it also helps in reducing the deployment time of a streaming service across various nodes.

2.2 Storing the Streamed data in DynamoDB

By default, the data is stored in Kinesis shards for a maximum of 24 hours. The default value can be changed to say 7 days but it cannot be infinitely large. Thus, the data need to be transferred into some persistent storage. Therefore, AWS DynamoDB is chosen as a persistent store. Firstly, a table is created with key as Sequence Number. The Sequence Number is the unique record number in the Kinesis stream. The Sequence Number is being chosen to make every record unique to avoid over-writing of records with same key.

Since the write capacity of writing units in DynamoDB is quite less. A multi-threaded program is required to read data continuously from all the shards. Processing is being done on the data to make the it quite homogeneous by discarding the records that does not fit to some required structure, i.e. filtering out the tweets which have missing values for the required attributes.

Moreover, since the data can be read from shards at different speeds because of the availability of the data in the shards and also depends upon the OS scheduling of threads, a predefined number of writing capacity will not be right. Thus, the auto-scaling feature of capacity units in DynamoDB has been enabled. Therefore, the number of writing capacity

units have been auto-scaled from minimum of 5 to 4000 with a target utilization of 80% to process all the needs.

Since, storing data in DynamoDB above the free tier can be quite expensive, the application is running continuously. The data later on for analyzing have been replicated multiple times in HDFS which is lot cheaper. In next section, the data is being transferred from DynamoDB to HDFS. In this particular case data can be directly written to HDFS. However, DynamoDB is chosen as a transaction database. Since in many practical cases transaction database is being used and HDFS is used mainly for historical data to do some kind of analysis on the data.

2.3 Moving data form DynamoDB to HDFS to apply Map Reduce Task

As per [6], performing operations on DynamoDB directly is costly if one has to perform it multiple times. Also storing large amount of data in DynamoDB can also be costly [6]. Therefore, the data is moved to HDFS from DynamoDB in batches whenever required.

Before moving the data, a cluster is needs to be setup. Amazon EMR service as defined in [9] has been used which makes the task a lot easier. A total of four nodes of type *m1.xlarge* were chosen to setup the cluster with large SSD storage of 420 GB at each node.

In [10], Amazon recommends using Hive to transfer the data from DynamoDB to HDFS. As one can select the required attributes only and Hive query engine automatically manages the transfer of data according to the number of specified reading units of the DynamoDB so that requests does not throttle.

Therefore an external table is created using hive which is mapped to the data in the DynamoDB table. Another external table was created in HDFS to store the data that will be copied from DynamoDB to HDFS. Thus, Hive knows the structure of the data and it will be very easy to copy only the required data. Finally, using INSERT, data was inserted into the table in HDFS from DynamoDB using Hive.

2.4 Analysis of data

After transferring the data from DynamoDB to HDFS, data is replicated to increase the size as lot of data is being filtered out while storing the data in DynamoDB and transferring it from DynamoDB to HDFS. This is being to see the affect of large amount of data to the processing in a cluster environment.

The application developed was restricted to perform only simple analysis on data. The application searches for the most common name that is found among all the users who tweets. This analysis ultimately lands on the word count problem on the names. Firstly, all other attributes are filtered and only name is chosen. Name can be comprised of more than one word. Therefore, they are split to get all worse within a name and then to perform the main analysis. After splitting, words are grouped uniquely and the number of occurrence of each word is saved along the name in the output file or is printed on the screen.

The above step is performed every time as soon as data is replicated every time. Some statistics like the time to complete, number of map tasks, number of reduces task and time

to replicate the whole data are noted and is also analyzed to get the deep understanding of Map-Reduce framework mainly using Hive.

The Hive query that has being written is a nested query. It firstly selects the *name* column from the table and then split all the words. The outer query on the other hand counts the occurrence of each word in the result generated by the inner query and printing the whole result on the screen in the ascending order. Thus, the last line of the result shows the most frequent name used while posting tweets.

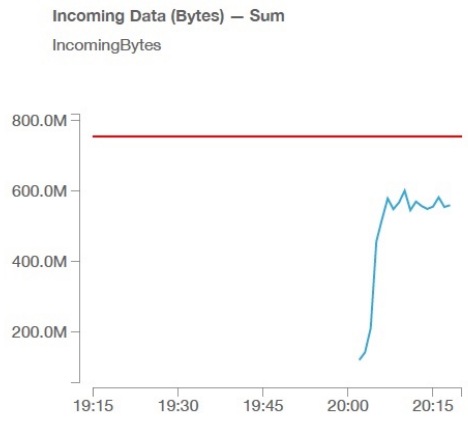

Figure 2: Incoming Data

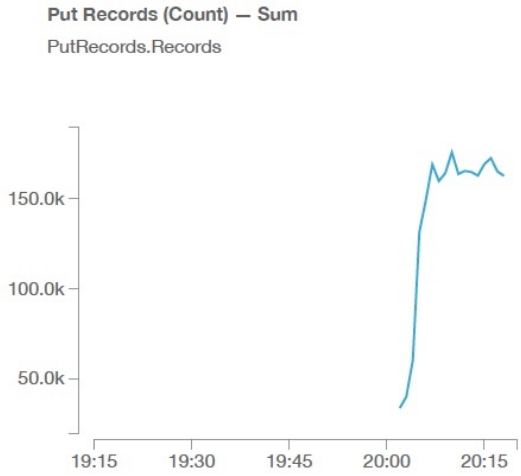

Figure 3: Put Records

3 Performance and Results

For this particular application, 11 VCL machines are being used to generate data from Twitter. A Kinesis stream with a total of 11 shards was created and proper parameters were

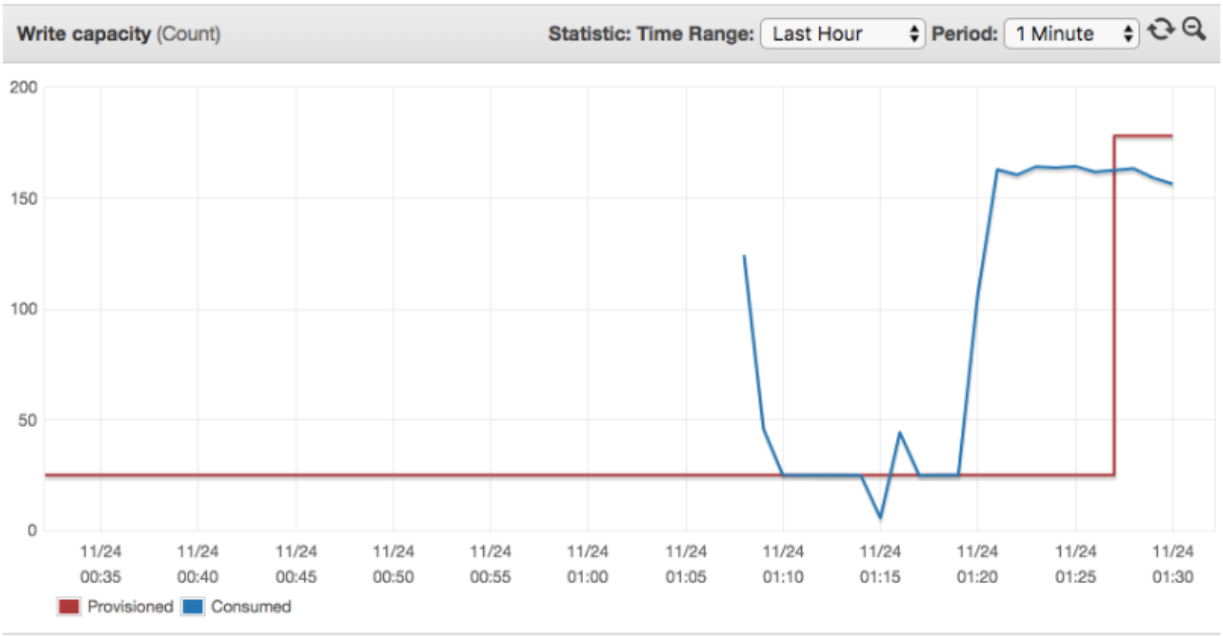

Figure 4: Write Capacity

Table 1: Hive Statistics

Size of Data (GB)	Number of Map Tasks	Number of Reduce1 Tasks	Number of Reduce2 Tasks	Execution Time (s)
1.8	25	5	1	52.3
3.6	50	9	1	94.1
7.2	99	17	1	112.07
14.4	86	33	1	130.04
28.8	87	66	1	161.98
57.6	85	131	1	213.85
115.2	86	262	1	339.33
230.4	149	524	1	538.73
460.8	299	1009	1	1242.95
921.6	599	1009	1	2160.88
1843.2	1200	1009	1	4540.66

supplied to the clients to generate one to one mapping between clients and shards. Figure 2 and 3 show the results of the volume and the velocity with which the data is being generated or fed into the Kinesis stream. The figure 2 depicts that on average we are able to generate the data between 600 to 800 MB/min through replication of tweets and load balancing which accounts for more than 150,000 records per minute.

DynamoDB is customized to handle large number of write requests. It auto-scales the number of capacity units to handle varying number of write requests which helps in saving cost too. Figure 4 depict this property of DynamoDB. As soon as write begin to throttle it increases the write capacity units from 25 to 170.

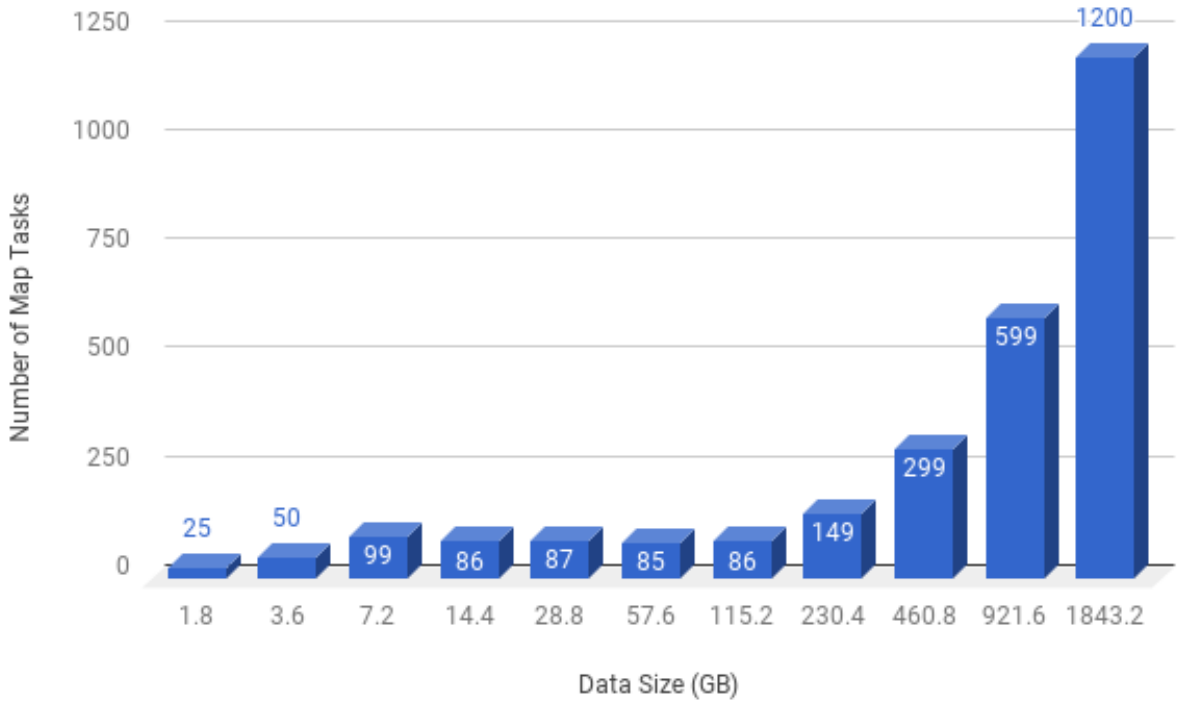

Figure 5: Number of Map Tasks

m1.xlarge machines has been used while creating the cluster which have 4 vCPU with 15 GB RAM and 1680 GB SSD storage space. The analysis on the data is performed using this cluster by doubling the size of data upto 1.8 TB.

As seen in the figure 5, for map tasks, the number is constant until a point, where as it doubles as the data is increased. This is because map is a distributed task, and as the data size increases, we need more map tasks to divide the data and work on the data. This increases the speed and simultaneously the computations are done on the different splits of data as defined by *Tez* [17].

For the reduce tasks, the number increases, but after the increase in data after a point, the reduce tasks remain constant which is 1099 by default. This is because there is a limit on the number of reduce tasks and we did reach that limit. It may be interesting to figure out that at a certain point of time the reduce tasks are more than map tasks. This is because the of *Apache Tez*. AWS uses *Tez* [16] by default in EMR, which is built on top of Yarn and gives a better performance for most of the workloads. It uses Map-Reduce-Reduce paradigm which doesn't write intermediate results to disks but passes the intermediate result directly to reducers[17]. This makes the processing faster as it saves time by avoiding disk IO's.

The total execution time doubles 7. This is because as the data size increases, the map tasks and reduce tasks also increase and we need more time to perform the analysis on the data. However, to reduce the execution time cluster size should be increased. All the execution observed parameters have been noted in the Table 1.

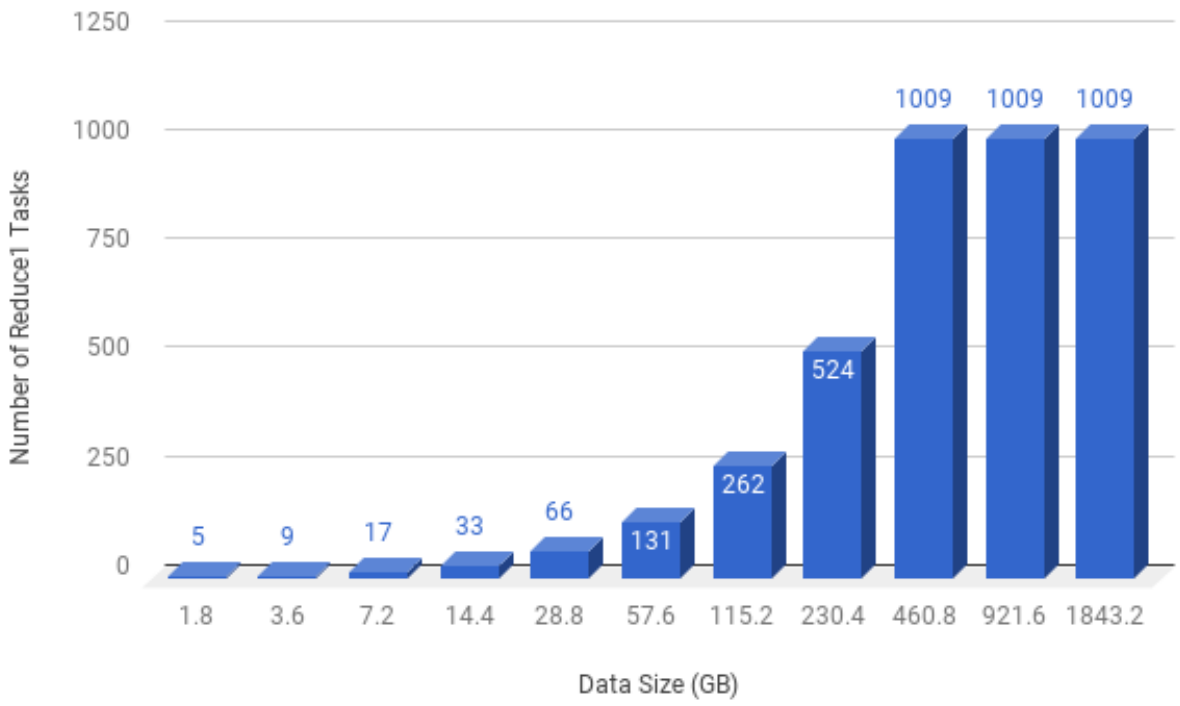

Figure 6: Number of Reduce Tasks

4 Related Work

The number of frameworks or solutions for Big Data are being increased continuously. New solutions or frameworks are coming to the market every year. Thus, determining the solution and integrating those solutions is a big challenge. Like [18] discusses about the various Big Data solutions and their comparisons. It also helps in understanding of how Hadoop frameworks works and it is being used by Hive, Pig and others. AWS in Hive uses *Tez* which is kind of implementation discussed by [19] helps in understanding the difference in the number of map tasks against the expected one.

Streaming data in Big Data is a challenging problem always. [20] uses Kafka to stream the tweets in real-time. AWS Kinesis is quite very similar to handle large amounts of data to do processing in a very similar fashion.

5 Conclusion

The system developed is able to capture and process large volumes of streaming data with varying velocity using tweets from Twitter. The data rate was further increased through replication of data received through Twitter. For ingesting data upto 800 MB/min, Amazon Kinesis is being used to ingest the streaming data efficiently by directing the data from one particular to a particular Kinesis shards. The consumers part of the application/ system gathers the replicated twitter data from the shards and storing it in DynamoDB, a NoSQL

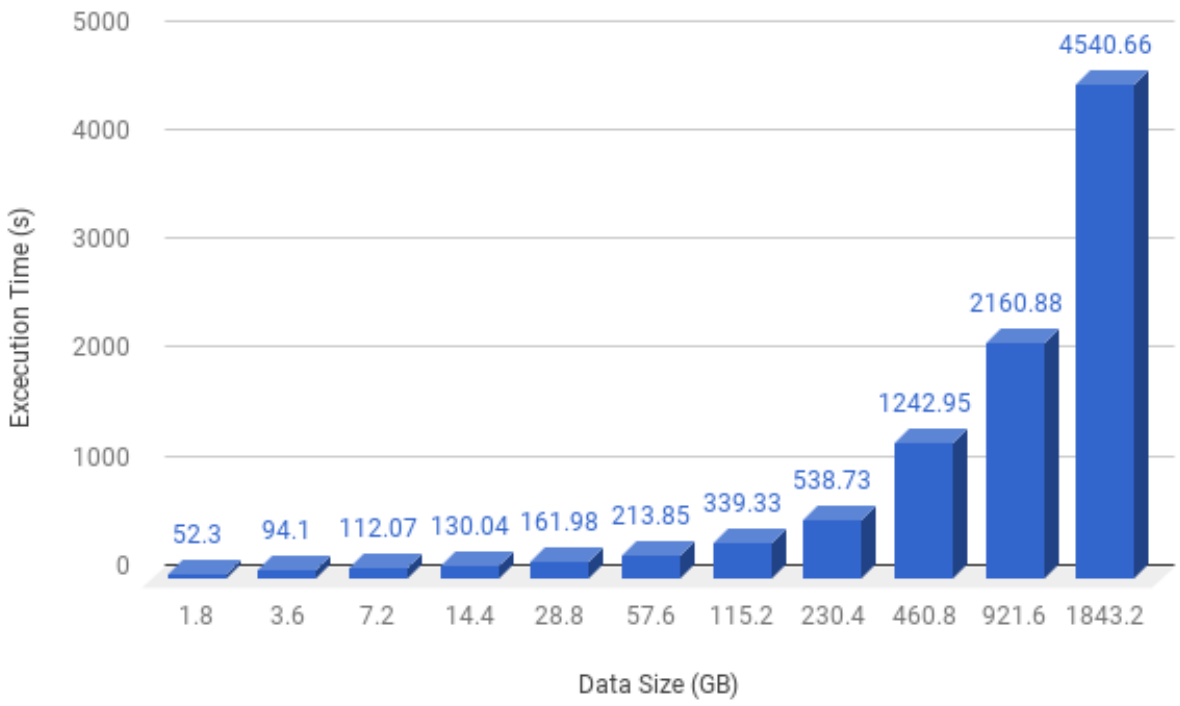

Figure 7: Time Taken

database. At any point the data can be easily transferred from DynamoDB to HDFS using HDFS to do any kind of batch processing from this data using various services offered by AWS EMR.

Acknowledgment

We thank our professor Dr. Vincent Freeh from the North Carolina State University who provided insight and expertise that greatly assisted the research. We also thank Liang Dong for continued assistance. Their feedback helped us to overcome many hurdles in this academic project.

References

[1] *Amazon Kinesis*, 2017. [Online]. Available: `https://aws.amazon.com/kinesis/`. [Accessed: 12- Nov- 2017].

[2] *Amazon Kinesis Streams Key Concepts*, 2017. [Online]. Available: `http://docs.aws.amazon.com/streams/latest/dev/key-concepts.html`. [Accessed: 12- Nov- 2017].

[3] *Amazon Kinesis PutRecord*, 2017. [Online]. Available: `http://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecord.html`. [Accessed: 12- Nov- 2017].

[4] *Amazon DynamoDB*, 2017. [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed: 12- Nov- 2017].

[5] *Throughput Capacity for Reads and Writes*, 2017. [Online]. Available: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ProvisionedThroughput.html. [Accessed: 12- Nov- 2017].

[6] *Amazon DynamoDB Pricing*, 2017. [Online]. Available: https://aws.amazon.com/dynamodb/pricing/. [Accessed: 14- Nov- 2017].

[7] *Amazon EC2*, 2017. [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 22- Nov- 2017].

[8] *Amazon Instance Details*, 2017. [Online]. Available: https://aws.amazon.com/ec2/previous-generation/. [Accessed: 22- Nov- 2017].

[9] *Amazon EMR*, 2017. [Online]. Available: https://aws.amazon.com/emr/. [Accessed: 12- Nov- 2017].

[10] *Copying Data Between DynamoDB and HDFS*, 2017. [Online]. Available: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EMRforDynamoDB.CopyingData.HDFS.html. [Accessed: 14- Nov- 2017].

[11] *Apache Hive*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Hive. [Accessed: Accessed: 12- Nov- 2017].

[12] *Pig (programming tool)*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Pig_(programming_tool). [Accessed: 12- Nov- 2017].

[13] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker and Ion Stoica, *Spark: Cluster Computing with Working Sets*.NSDI12, April 2012, San Jose, CA, USA.

[14] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker and Ion Stoica, *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*.HotCloud 10.10-10 (2010).

[15] Andrew S. Tanenbaum and David J. Wetherall, *Computer Networks*, 5th ed'.Addison-Wesley, 2011.

[16] *Hive on Amazon EMR*, 2017. [Online]. Available: http://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive-differences.html). [Accessed: 04- Dec- 2017].

[17] *Apache Tez*, 2017. [Online]. Available: https://www.infoq.com/articles/apache-tez-saha-murthy). [Accessed: 04- Dec- 2017].

[18] *A. Bhardwaj, Vanraj, A. Kumar, Y. Narayan and P. Kumar*, 2015. *2nd International Conference on Recent Advances in Engineering Computational Sciences (RAECS),Big data emerging technologies: A CaseStudy with analyzing twitter data using apache hive.*

[19] *Xiaohong Zhang, Guowei Wang, Zijing Yang and Yang Ding*, 2012.*2012 International Conference on Systems and Informatics (ICSAI 2012), A Two-phase Execution Engine of Reduce Tasks In Hadoop MapReduce.*

[20] *Babak Yadranjiaghdam, Seyedfaraz Yasrobi and Nasseh Tabrizi* , 2017.*2017 IEEE 6th International Congress on Big Data, Developing a Real-time Data Analytics Framework For Twitter Streaming Data.*