Data Streaming and Processing with AWS Kinesis

Xinzuo Wang
xw3xp@virginia.edu

Abstract— **Streaming data generated continuously by varies data sources contains information may be valuable to users and applications. Having the ability to process and analyze this data in real-time is essential for use cases which need continuous data flow, such as website analytics and machine learning. In this brief paper, we design a data streaming and processing system based on AWS Kinesis. The system consists of two integrated modules: the data streaming module and the data processing and analyzing module. For data input and streaming, we build the data streaming module using Kinesis Streams. Kinesis Streams collects the data and streams the data to the costumed destinations. For data processing and analyzing module, we employ several AWS services to achieve the goal. One implementation is based on Kinesis Analytics and AWS Lambda. In this implementation, we incorporated these two functions to provide real-time data analysis. The other implementation is based on AWS EMR. In this implementation, we build our real-time application using Spark framework, and the code is run on AWS EMR. Compared with non-real-time non-cloud methods for data processing, our system has features such as 1) high data availability, 2) high service uptime, 3) high scalability, and 4) cost effective.**

Index Terms— **data streaming, streaming analytics, real-time, AWS Kinesis, Apache Spark.**

I. INTRODUCTION

Data processing has a wide range of applications, and processing the streaming data in real-time is becoming more and more important since the number of applications which tackles real-time data streams is growing fast. Real-time data holds potentially high value for users, but it also comes with a expiration time. If the data is not processed in a certain window of time, it will probably become unimportant and never needed. Nowadays, this kind of streaming data can be found everywhere, such as the real-time data from temperature sensor, blip in log file, and price changes in stock market.

Various data streams could have very different features [1]. For example, the data from wireless temperature sensors depends on sampling, and the data stream could be noisy. On the other hand, data from financial market have high complexity, and the analysis needed to be done in real time. To handle the real-time data properly, processing and analyzing the real-time streaming data is indispensable. Streaming analytics solutions enable us to extract information in real time, or process them in batches for later use. The power of streaming analytics is such that it allows for the streaming of millions of events in a second and thus allows us to build mission-critical applications that require the performance to be quick and efficient [2],[3].

Amazon Web Services provides several options to work with streaming data, and one of the major tools is Kinesis. Kinesis is similar to Apache Kafka, it is a fully managed service that makes it easy to collect, process, and analyze real-time streaming data. It enables user to process streaming data at any scale, along with the flexibility to choose the tools that best suit the requirements of certain application. In this paper, we used all three Kinesis tools – Kinesis Streams, Kinesis Analytics, and Kinesis Firehose. Kinesis Streams is capable of capturing large amounts of data from data producers, and streaming it into custom applications for data processing and analysis. Kinesis Firehose is able to capture streaming data, perform in-line processing to clean, format, and deliver this data to service destinations such as S3, Redshift and Lambda. Kinesis Analytics is used to analyze streaming data in real time. Together with the Kinesis services, in this paper, we also employed AWS Elastic Map Reduce (EMR). It is a managed cluster platform running big data frameworks, such as Apache Hadoop and Spark, it can process data for analytics purposes and read/write data from/into WAS data stores and databases. The structure of AWS streaming and processing system is shown in Fig. 1.

In this paper, we propose a data streaming and processing system based on AWS Kinesis. The system consists of two integrated modules: the data streaming module and the data processing and analyzing module. For data streaming, we build the data streaming module using Kinesis Streams. Kinesis Streams collects the data and streams the data to the costumed destinations. For data processing and analyzing module, we employ several AWS services to achieve the goal. One implementation is based on Kinesis Analytics and AWS Lambda. In this implementation, we incorporate these two functions to provide real-time data analysis. The other implementation is based on AWS EMR. In this implementation, we build our real-time application using Spark framework, and the code is run on AWS EMR. To illustrate the performance of our system, we programed an application to analyze the tweets from Twitter in real time. The producer injects the twitter feed into Kinesis Streams, and the consumer perform analytics jobs such as printing, counting, and clustering. Meanwhile, we used Kinesis Analytics and AWS Lambda to do data analysis. Also, we use Kinesis Firehose to deliver the processed data to different data stores, and thus use the data to perform analytics jobs on EMR.

The rest of this paper is organized as follows: Section II describes the features of cloud-based real-time data streaming and processing system and its benefits. Section III shows the design of data streaming and processing system. Section IV is

The author is with the Computer Science Department, University of Virginia, Charlottesville, USA

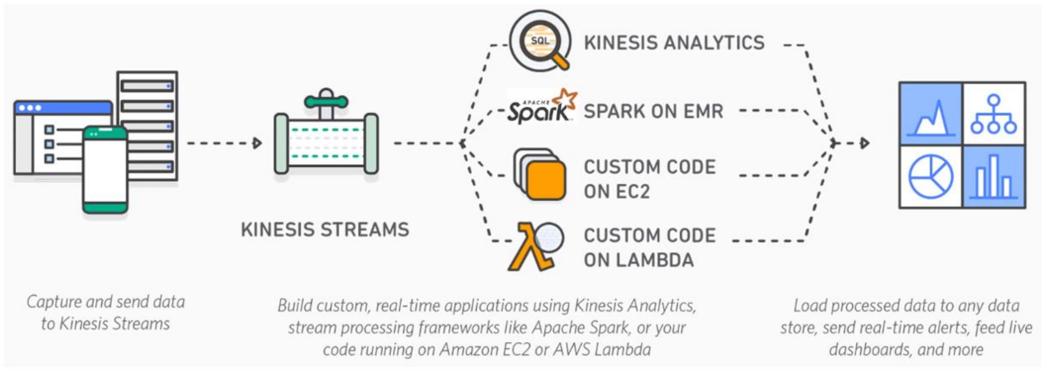

KINESIS ANALYTICS
SPARK ON EMR
CUSTOM CODE ON EC2
CUSTOM CODE ON LAMBDA
KINESIS STREAMS
Capture and send data to Kinesis Streams
Build custom, real-time applications using Kinesis Analytics, stream processing frameworks like Apache Spark, or your code running on Amazon EC2 or AWS Lambda
Load processed data to any data store, send real-time alerts, feed live dashboards, and more

Fig.1 AWS Data Streaming and Processing System (from http://bit.ly/2iYX1ex)

the demonstration and experiment results. Section V is the conclusion and future considerations.

II. Problem And Feature Description

Using cloud solutions for processing data in real-time has obvious benefits over non-cloud non-real-time solutions.

Stream Processing targets the scenarios in which data's value diminishes fast with time. The key strength of stream processing is that it can provide insights faster, often within milliseconds to seconds [4]. And the following are some of the feature of stream processing regarding the problems of non-real-time solutions:

1) Some data comes as a never-ending stream of events. To do batch processing, the data collection has to be stopped at some time and processes the data. In contrast, streaming handles the data streams in real time. The data manipulations and analytics jobs can be done almost simultaneously.

2) Batch processing lets the data build up and try to process them at once while stream processing process data as they come in hence spread the processing over time.

3) There are cases where data is too huge to be stored. For these use cases, stream processing is a better way to handle large data and retain only useful bits.

For cloud-based data streaming and processing services, such as AWS Kinesis [5], it provides the following benefits over non-cloud solutions:

1) When the data streaming and processing is done on devices such as laptop, there are risks of data loss and service unavailability. The cloud solution will solve this problem using synchronous replication and stream protection to guarantee high service uptime.

2) Cloud platform provides high scalability. The throughput of the stream can be adjusted at any time based on the volume of the input data, and it defeats non-cloud solutions which cannot scale dynamically.

3) The cost is only the resources used. And the price is as little as $0.015 per hour. Consider the scalability and performance, such solution is cost effective.

III. System Design

The system consists of two integrated modules: the data streaming module and the data processing and analyzing module. For data streaming, we build the data streaming module using Kinesis Streams. For data processing and analyzing module, we have implementations 1) on Kinesis Analytics and AWS Lambda, and 2) on AWS EMR. An Twitter application which uses Kinesis structure to analyze the tweets in real time is created. The producer injects the twitter feed into Kinesis Streams, and the consumer perform analytics jobs. We also used Kinesis Analytics, AWS Lambda, and EMR to do data analysis.

A. Create the Twitter Application

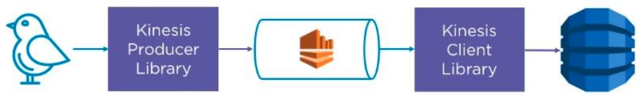


Fig. 2. The structure of twitter application.

The twitter application we developed mainly consists of two parts: the producer function which connects to the tweet feed API and ingest the data stream to Kinesis Streams, and the consumer function which process the records and generate the analytical results. The following sections explain this application in detail.

1. Application Producer

1) Ingest tweets into Kinesis Streams

In producer function, firstly we have to configure the credentials to connect to Twitter API and request tweet feeds. The tweet samples will be provided in real time.

Secondly, we need to create listener, which detects the status of the tweets. When there is a new tweet available, the onMessage function will be called, and it will ingest the tweets into the Kinesis Streams we previously configured.

To write the data to Kinesis Streams, we need to add user record and add call back to check the status.

Code1: ***Write Twitter data to Kinesis Streams***

```
// configure the twitter API
createTwitterStream(){
   configurationBuilder
      .setAuthConsumerKey()
      .setAuthConsumerSecret()
      .setAuthAccessToken()
      .setAuthAccessTokenSecret();
}
// put record to Kinesis
KinesisProducer.addUserRecord(
        "stream-name",
        partitionKey,
        recordBytes);
    Futures.addCallback();
```

The status of the data stream can be monitored on AWS Console, under the tag Put Record. The results are shown in Section IV.

2. Application Consumer

The consumer is used to process the records from the stream with KCL. we can obtain the records when we provide the partition key, data and sequence number. Apart from this getRecords function, we also need to build the worker to obtain the record of the destination. To be noted, the records from KPL is aggregated to reduce the input volume, thus, when we extract the record from the stream, it need to be de-aggregated to display the original content. This mechanism will be further explain in Section III-B.

Code2: ***Get records from Kinesis Streams***

```
    // get records
    processRecords(RecordsInput){
        for(getRecords())
            getPartitionKey();
            getData();
            getSequenceNumber();
    }
    // build and run worker
    KCLCofig(
        "application-name",
        "stream-name",
        "worker-name",
    );
    worker.build();
    worker.run();
```

The status of the data stream can be monitored on AWS Console, under the tag Get Record. The results are shown in Section IV.

3. Stream Shards

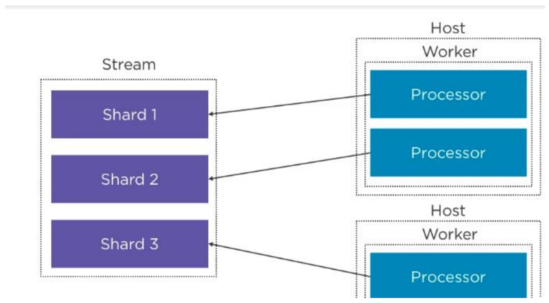


Fig. 3. The connection between shards and processors.

We can configure the shards we need based on the throughput of the system. With more shards, we will be able to handle larger volume of input data. One feature of the Kinesis Stream is that it scales automatically. Say if we have two shards, when we create one worker to process the records, it will create two processors, and each of them will be assigned a shard of Kinesis Stream. We can have multiple workers working on several hosts. With new shard established, the worker will create new processor too.

4. Language Analysis

To achieve real-time stream data analysis, we wrote a function to cluster the twitter by the language. When processing the records, we firstly set the time interval (to one minute), and use the previous minutes' statistics to generate the results. We also set a checkpoint every time the new interval starts, in this way, we can guarantee the data records will still be read continuously.

Code3: ***Get and process records in real time***

```
    processRecords(RecordsInput){
        for(getRecords()){
            if(new_interval){
                save_prev_records();
                checkpoint(status);
                current_time(status);
            }
            processTweets();
            recover_status(status);
    }
```

For generating the result, the program firstly iterates through all the languages and record the counts of each language, then it will save the statistics to destination data store. The results are shown in Section IV.

5. Connect to Storage Destinations

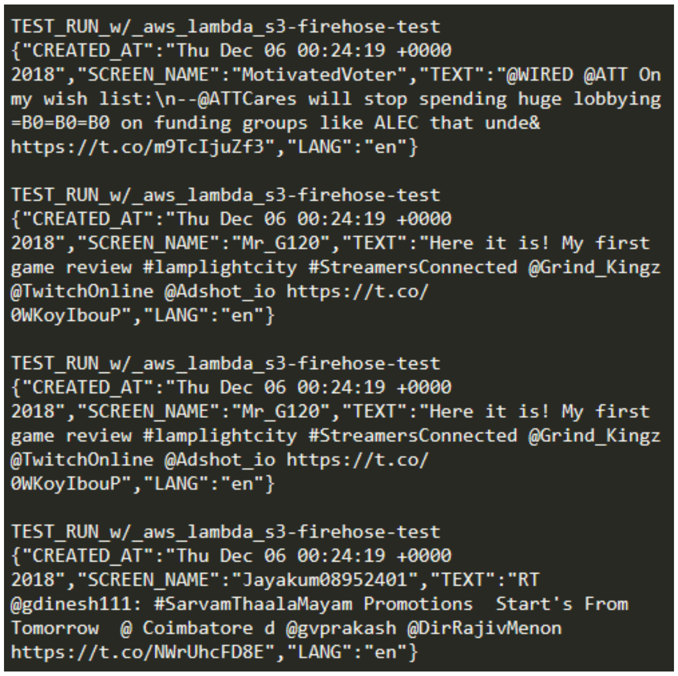
For Kinesis Streams, there are several storage destinations to choose from. In our application, we used AWS S3 as the data store. The API of S3 is simple, to store the data from application consumer to S3, all we need to do is create the properties (name of the application, the bucket name, buffer size limit), and run the worker.

In our later implementations, we also tried funneling data with Kinesis Firehose. Firehose provides more tools to process data. To implement Firehose, we need to specify the source of the data stream (Kinesis Streams), and decide if "transform records" is needed (this step can be integrated with AWS Lambda), in the end, choose destination (s3), and launch. Now the data could be read from accessed from the specified s3. In this way, we can duplicate the kinesis stream, or use the transform records function to process the streaming data at the same time.

For language analysis function, we also employed the AWS Lambda to preprocess the records extracted from the stream. In this implementation, we firstly used Lambda to decode the KPL records (which are in binary), and then filtered the tweets by 'Language = 'en'', and changed the format of the tweets, and then stored them in S3. As we can see from Fig.4, we successfully decode the records, and add a line of words "TEST_RUN…" before each new tweet.

Code4: ***Lambda function for KPL record preprocessing***

```
   const output = event.records.map(function(record) {
        var textTrans = new Buffer(
                    record.data, 'base64').toString('ascii');

        return {
            recordId: record.recordId,
            result: 'Ok',
            data: new Buffer(
                "TEST_RUN_w/_aws_lambda_s3-firehose-test\n"
                + textTrans + "\n\n").toString('base64'),
        }
```



```
TEST_RUN_w/_aws_lambda_s3-firehose-test
{"CREATED_AT":"Thu Dec 06 00:24:19 +0000
2018","SCREEN_NAME":"MotivatedVoter","TEXT":"@WIRED @ATT On
my wish list:\n--@ATTCares will stop spending huge lobbying
=B0=B0=B0 on funding groups like ALEC that unde&
https://t.co/m9TcIjuZf3","LANG":"en"}

TEST_RUN_w/_aws_lambda_s3-firehose-test
{"CREATED_AT":"Thu Dec 06 00:24:19 +0000
2018","SCREEN_NAME":"Mr_G120","TEXT":"Here it is! My first
game review #lamplightcity #StreamersConnected @Grind_Kingz
@TwitchOnline @Adshot_io https://t.co/
0WKoyIbouP","LANG":"en"}

TEST_RUN_w/_aws_lambda_s3-firehose-test
{"CREATED_AT":"Thu Dec 06 00:24:19 +0000
2018","SCREEN_NAME":"Mr_G120","TEXT":"Here it is! My first
game review #lamplightcity #StreamersConnected @Grind_Kingz
@TwitchOnline @Adshot_io https://t.co/
0WKoyIbouP","LANG":"en"}

TEST_RUN_w/_aws_lambda_s3-firehose-test
{"CREATED_AT":"Thu Dec 06 00:24:19 +0000
2018","SCREEN_NAME":"Jayakum08952401","TEXT":"RT
@gdinesh111: #SarvamThaalaMayam Promotions  Start's From
Tomorrow  @ Coimbatore d @gvprakash @DirRajivMenon
https://t.co/NWrUhcFD8E","LANG":"en"}
```

Fig. 4. The transformed message.

B. *Analysis with Kinesis Analytics and AWS Lambda*

Kinesis is able to read data from Kinesis Streams and Kinesis Firehose. To use the data in Kinesis Analytics, the records (json/csv/text format) need to be transformed into Analytics' in-application streams.

1. Preprocessing with AWS Lambda

Aws lambda can be integrated with kinesis analytics. We can enable the stream transformation to preprocess the data for kinesis analytics. In our case, the KPL records in our input data stream is binary, so we need to transform it into json before it is used to detect the schema.

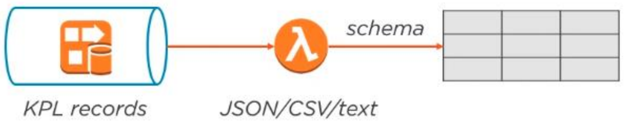
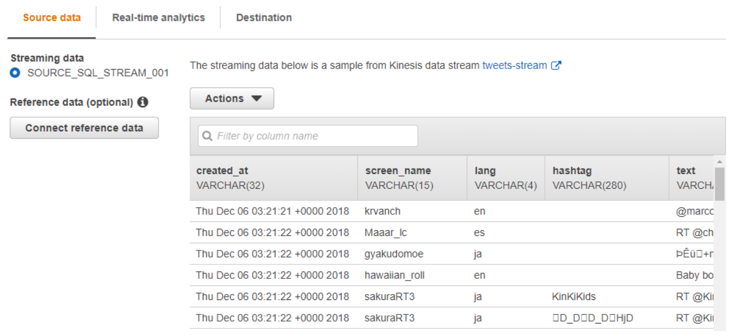


Fig. 5. Lambda preprocessing KPL records.

2. Configure the Schema and SQL Queries

The schema of data analytics application's input stream defines how data from the stream is made available to SQL queries in the application. It basically contains the keyword and its fields in the records, as is in Fig. 6. To configure the schema, we can use the schema discovery function or define the schema by ourselves.



The streaming data below is a sample from Kinesis data stream tweets-stream

created_at VARCHAR(32)	screen_name VARCHAR(15)	lang VARCHAR(4)	hashtag VARCHAR(280)	text VARCH
Thu Dec 06 03:21:21 +0000 2018	krvanch	en		@marcc
Thu Dec 06 03:21:22 +0000 2018	Maaar_lc	es		RT @ch
Thu Dec 06 03:21:22 +0000 2018	gyakudomoe	ja		ÞÊü□+r
Thu Dec 06 03:21:22 +0000 2018	hawaiian_roll	en		Baby bo
Thu Dec 06 03:21:22 +0000 2018	sakuraRT3	ja	KinKiKids	RT @Ki
Thu Dec 06 03:21:22 +0000 2018	sakuraRT3	ja	□D_D□D_D□HjD	RT @Ki

Fig. 6. Schema keywords example.

In Analytics, the input stream is transformed with SQL queries. In SQL Editor, we also need to create a "pump", with which transformed stream can be "pumped" to external destinations such as Kinesis Streams, Kinesis Firehose, and AWS Lambda. The example code looks like this:

Code5: *SQL queries for filtering the English tweets*

```
 -- create destination stream
CREATE OR REPLACE STREAM "FILTER_STREAM" (
    created_at VARCHAR(32),
    screen_name VARCHAR(15),
    text VARCHAR(280),
    lang VARCHAR(4)
    );
-- create pump
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO " FILTER_STREAM "
-- SQL query
SELECT STREAM "created_at", "screen_name", "text",
"lang"
FROM "SOURCE_SQL_STREAM_001"
WHERE "lang" = 'en';
```

In this section, we created analytics functions to filter tweets and get top hashtags. The results are shown in Section IV.

C. *Analysis with EMR*

AWS EMR runs Hadoop clusters using EC2 instances. With EMR, we can launch clusters without having to maintain the hardware, and pay for the computation capacity we need. In this section, we run Spark jobs on EMR. Since Spark stores data in memory, its hardware requirements can be demanding depending on the workload. On EMR, we can size the clusters to meet the needs. Spark on Amazon EMR can process data from S3 bucket or Kinesis Streams, and the results can be delivered to data store destinations after analysis.

In this part, the following works are done in time sequence:

1) Build Spark Project with Scala

The development is done in IntelliJ IDEA with sbt, including running spark job using spark-shell, generating jar, and some try outs in YARN. This part is mainly focused on getting familiar with Scala and Spark

2) Run Spark Application on EMR

The procedure achieved in this part is fetching the input files from S3, submit the jar file to EMR cluster and run the spark jobs using "spark-submit", and optionally store the output back to S3. Part of the work is configuring the EMR cluster, including the input and output, master and slave nodes and security settings. The other part is modifying the Spark program in previous section to be compatible with EMR.

3) Kinesis Stream and Spark Integration

This part mainly modifies the API of the data source to Kinesis Stream, and import AWS libraries. Since this section validates that EMR can be employed as a method to run data analysis jobs, we only wrote the program to print tweets and count hashtags. The results are shown in Section IV.

IV. Demonstration and Results

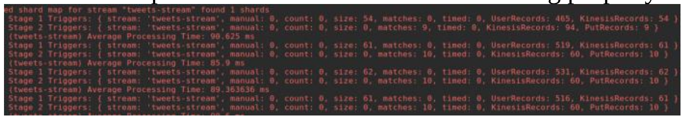
In this section, we present the demonstration of the data streaming and processing system. With several experiments, we will have a closer look at each implementation and its performance.

A. *Twitter Application*

In this section, we show the running results of the producer and consumer.

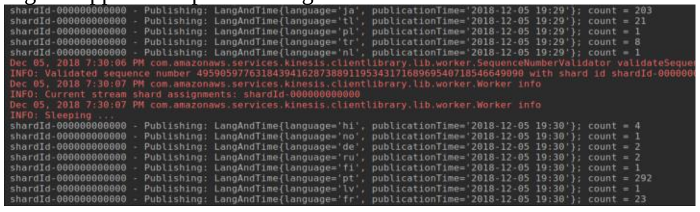
1) Fig. 7 is the log showing that the data is successfully ingested into the data stream. Each UserRecords is the number of tweets, and the result shows that Kinesis aggregates the data into KPL records and write into the stream.

Fig. 8 is the log and output information showing that the consumer can extract the records from the stream, and the Language Analysis function (refer to Section III-A-4) is able to cluster the tweets by the language. We can see from the results, that Language Analysis function is performed every minute, and the checkpoint function of consumer is working properly.



```
ed shard map for stream "tweets-stream" found 1 shards
 Stage 1 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 54, matches: 0, timed: 0, UserRecords: 465, KinesisRecords: 54 }
 Stage 2 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 0, matches: 9, timed: 0, KinesisRecords: 94, PutRecords: 9 }
 (tweets-stream) Average Processing Time: 90.625 ms
 Stage 1 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 61, matches: 0, timed: 0, UserRecords: 519, KinesisRecords: 61 }
 Stage 2 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 0, matches: 10, timed: 0, KinesisRecords: 60, PutRecords: 10 }
 (tweets-stream) Average Processing Time: 85.9 ms
 Stage 1 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 62, matches: 0, timed: 0, UserRecords: 531, KinesisRecords: 62 }
 Stage 2 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 0, matches: 10, timed: 0, KinesisRecords: 60, PutRecords: 10 }
 (tweets-stream) Average Processing Time: 89.363636 ms
 Stage 1 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 61, matches: 0, timed: 0, UserRecords: 516, KinesisRecords: 61 }
 Stage 2 Triggers: { stream: 'tweets-stream', manual: 0, count: 0, size: 0, matches: 10, timed: 0, KinesisRecords: 60, PutRecords: 10 }
```

Fig. 7. Application producer ingest the data into Kinesis Streams.



```
shardId-000000000000 - Publishing: LangAndTime{language='ja', publicationTime='2018-12-05 19:29'}; count = 203
shardId-000000000000 - Publishing: LangAndTime{language='tl', publicationTime='2018-12-05 19:29'}; count = 21
shardId-000000000000 - Publishing: LangAndTime{language='pl', publicationTime='2018-12-05 19:29'}; count = 1
shardId-000000000000 - Publishing: LangAndTime{language='tr', publicationTime='2018-12-05 19:29'}; count = 8
shardId-000000000000 - Publishing: LangAndTime{language='nl', publicationTime='2018-12-05 19:29'}; count = 1
Dec 05, 2018 7:30:06 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.SequenceNumberValidator validateSeque
INFO: Validated sequence number 49590597763184394162873889119534317168969540718546649090 with shard id shardId-000000
Dec 05, 2018 7:30:07 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker info
INFO: Current stream shard assignments: shardId-000000000000
Dec 05, 2018 7:30:07 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker info
INFO: Sleeping ...
shardId-000000000000 - Publishing: LangAndTime{language='hi', publicationTime='2018-12-05 19:30'}; count = 4
shardId-000000000000 - Publishing: LangAndTime{language='no', publicationTime='2018-12-05 19:30'}; count = 1
shardId-000000000000 - Publishing: LangAndTime{language='de', publicationTime='2018-12-05 19:30'}; count = 2
shardId-000000000000 - Publishing: LangAndTime{language='ru', publicationTime='2018-12-05 19:30'}; count = 2
shardId-000000000000 - Publishing: LangAndTime{language='fi', publicationTime='2018-12-05 19:30'}; count = 1
shardId-000000000000 - Publishing: LangAndTime{language='pt', publicationTime='2018-12-05 19:30'}; count = 292
shardId-000000000000 - Publishing: LangAndTime{language='lv', publicationTime='2018-12-05 19:30'}; count = 1
shardId-000000000000 - Publishing: LangAndTime{language='fr', publicationTime='2018-12-05 19:30'}; count = 23
```

Fig. 8. Application consumer reads the records, and the Language Analysis function is performed.

2) Fig. 9 and Fig. 10 is the monitoring results of Kinesis Streams. Due to paper page limited, we only show the sum and latency of Put Records and Get Records.

3) Fig. 11 is the monitoring results of Kinesis Firehose. Due to paper page limited, we only show the monitoring results of data connection to S3. It should also be noted that it also display the incoming record counts and succeed processing counts.

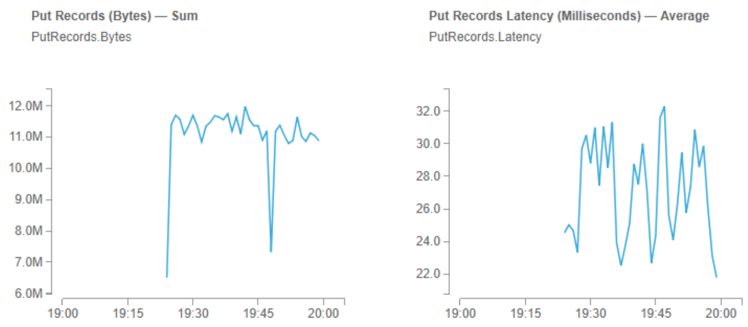


Fig. 9. Bytes Sum and Average Latency of Put Records.

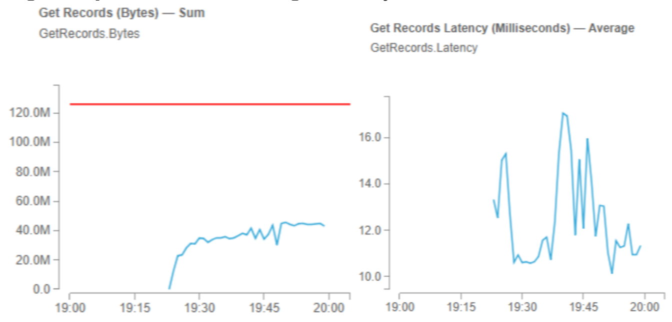


Fig. 10. Bytes Sum and Average Latency of Get Records.

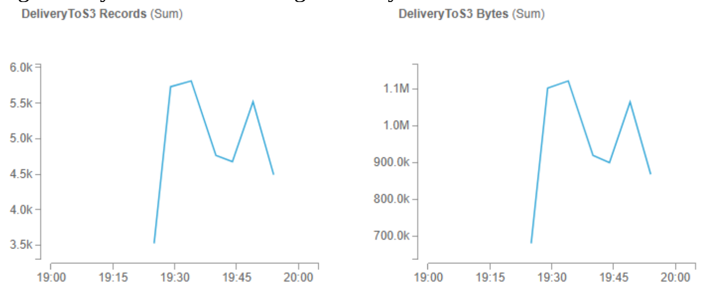
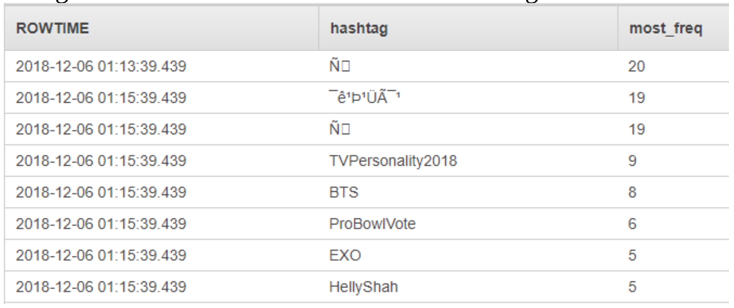


Fig. 10. Bytes Sum and Average Latency of Get Records.

B. *Analysis with Kinesis Analytics and AWS Lambda*

In this section, two functions are run on Kinesis Analytics. One is the twitter filtering, the SQL code is provided in Code5, and the result format is similar to get top hashtags function. Get top hashtags function displays the most frequent hashtags during a 120s windows. The result is shown in Fig. 11.

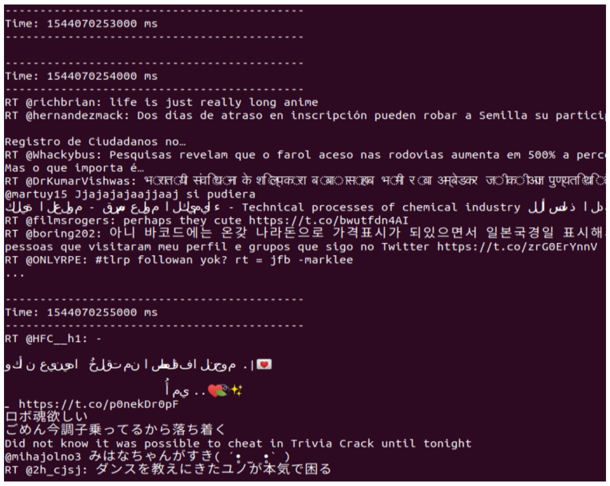


ROWTIME	hashtag	most_freq
2018-12-06 01:13:39.439	Ñ	20
2018-12-06 01:15:39.439	¯ê¹Þ¹ÜÃ¯¹	19
2018-12-06 01:15:39.439	Ñ	19
2018-12-06 01:15:39.439	TVPersonality2018	9
2018-12-06 01:15:39.439	BTS	8
2018-12-06 01:15:39.439	ProBowlVote	6
2018-12-06 01:15:39.439	EXO	5
2018-12-06 01:15:39.439	HellyShah	5

Fig. 11. Get top hashtags function running on Kinesis Analytics.

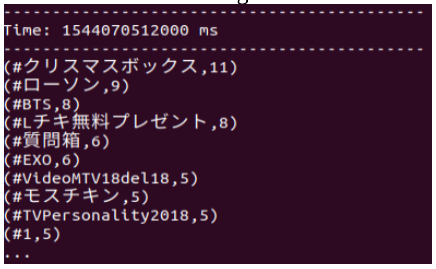
C. *Analysis with EMR*

In this section, two functions are run on EMR. One is the twitter printing, the other is displays the most frequent hashtags. The results are shown in Fig. 11 and Fig. 12.



```
-------------------------------------------
Time: 1544070253000 ms
-------------------------------------------

-------------------------------------------
Time: 1544070254000 ms
-------------------------------------------
RT @richbrian: life is just really long anime
RT @hernandezmack: Dos días de atraso en inscripción pueden robar a Semilla su partici

Registro de Ciudadanos no…
RT @Whackybus: Pesquisas revelam que o farol aceso nas rodovias aumenta em 500% a perc
Mas o que importa é…
RT @DrKumarVishwas: भारतीय संविधान के शिल्पकार बाबा साहब भीम राव अम्बेडकर जी की आज पुण्यतिथि
@martuy15 Jjajajajaajjaaj si pudiera
[illegible] - Technical processes of chemical industry [illegible]
RT @filmsrogers: perhaps they cute https://t.co/bwutfdn4AI
RT @boring202: 아니 바코드에는 온갖 나라돈으로 가격표시가 되있으면서 일본국경일 표시해
pessoas que visitaram meu perfil e grupos que sigo no Twitter https://t.co/zrG0ErYnnV
RT @ONLYRPE: #tlrp followan yok? rt = jfb -marklee
...

-------------------------------------------
Time: 1544070255000 ms
-------------------------------------------
RT @HFC__h1: -

[illegible] .|💌

                    [illegible] ..❤️✨
_ https://t.co/p0nekDr0pF
ロボ魂欲しい
ごめん今調子乗ってるから落ち着く
Did not know it was possible to cheat in Trivia Crack until tonight
@mihajolno3 みはなちゃんがすき( ´•̥  •̥` )
RT @2h_cjsj: ダンスを教えにきたユンが本気で困る
```

Fig. 11. Print tweets function running on EMR.



```
-------------------------------------------
Time: 1544070512000 ms
-------------------------------------------
(#クリスマスボックス,11)
(#ローソン,9)
(#BTS,8)
(#Lチキ無料プレゼント,8)
(#質問箱,6)
(#EXO,6)
(#VideoMTV18del18,5)
(#モスチキン,5)
(#TVPersonality2018,5)
(#1,5)
...
```

Fig. 12. Get top hashtags function running on EMR.

V. Conclusion

In this brief paper, we designed a data streaming and processing system based on AWS Kinesis. The system consists of two integrated modules: the data streaming module using Kinesis Streams, and the data processing and analyzing module implement 1) on Kinesis Analytics and AWS Lambda, and 2) on AWS EMR. Compared with non-real-time non-cloud methods for data processing, our system has features of high data availability, high service uptime, high scalability, and cost effective.

There are many future considerations or extensions. For example: 1) Similar frameworks for data streaming such as Apache Kafka should be investigated and compared. 2) In our data streaming application, the multi-shard problem is not considered, related problems such as resharding and fault-tolerance should be investigated.

References

[1] D. Namiot, "On Big Data Stream Processing," *Intl. J. Open Info. Tech.*, vol. 3, no. 8, pp. 48-51, 2015.

[2] Datafloq, “Streaming Analytics Platforms For Real-time Applications,” *https://datafloq.com/read/streaming-analytics-platforms-real-time-apps*, Mar 2018.

[3] L. Querzoni, and N. Rivetti, " Data Streaming and its Application to Stream Processing," *Proceedings of the 11th ACM Intl. Conf. Distributed Event-based Sys.*, June. 2017.

[4] S. Perera, " A Gentle Introduction to Stream Processing," *https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97*, Apr. 2018.

[5] “Amason Kinesis Overview,” *https://aws.amazon.com/kinesis/*.

[6] D. Jankov, *et. al.*, "Real-time High Performance Anomaly Detection over Data Streams," *Proceedings of the 11th ACM Intl. Conf. Distributed Event-based Sys.*, June. 2017.

[7] G. Chen, *et. al.*, "Realtime Data Processing at Facebook," *Proceedings of the 2016 Intl. Conf. Management of Data*, 2016.

[8] A. Toshniwal, *et. al.*, "Storm@Twitter," *Proceedings of the 2014 ACM SIGMOD Intl. Conf. Management of Data*, 2014.