**Stream Processing in Real-time Distributed System and Analytics using Apache Spark streaming to a Kafka Server.**

**Table of Contents**

1. [Overview](https://github.com/yugokato/Spark-and-Kafka_IoT-Data-Processing-and-Analytics#1-overview)
2. [Format of sensor data](https://github.com/yugokato/Spark-and-Kafka_IoT-Data-Processing-and-Analytics#2-format-of-sensor-data)
3. [Analysis of data](https://github.com/yugokato/Spark-and-Kafka_IoT-Data-Processing-and-Analytics#3-analysis-of-data)
4. [Results](https://github.com/yugokato/Spark-and-Kafka_IoT-Data-Processing-and-Analytics#4-results)

**1. Overview**

**Use case**

* Analyzing temperature of Bangladesh in different time from IoT sensors in real-time

**Project Scenario:**

* Multiple temperature sensors are deployed in each Area
* Each sensor regularly sends temperature data to a Kafka server in AWS Cloud (Simulated by feeding 10,000 JSON data by using kafka-console-producer)
* Kafka client retrieves the streaming data every 3 seconds
* PySpark processes and analizes them in real-time by using Spark Streming, and show the results

**Key Technologies:**

* Apache Spark (Spark Streaming)
* Apache Kafka
* Python/PySpark

**2. Format of sensor data**

I used the simulated data for this project. iotsimulator.py generates JSON data as below format.

<Example>

{

"guid": "0-ZZZ12326678-08K",

"destination": "0-AAA12326678",

"state": "Dhaka",

"eventTime": "2019-11-16 T13:26:39.267974Z",

"payload": {

"format": "urn:example:sensor:temp",

"data":

{

"temperature": 22.7

}

}

}

| **Field** | **Description** |
| --- | --- |
| guid | : A global unique identifier which is associated with a sensor. |
| destination: | An identifier of the destination which sensors send data to (One single fixed ID is used in this project) |
| State | : A randomly chosen Dhaka state. A same guid always has a same state |
| Event Time | : A timestamp that the data is generated |
| format | : A format of data |
| temperature | : Calculated by continuously when the data is generated. |

If you need to generate 10,000 sensors data:

$. /iotsimulator.py 10000 > testdata.txt

**3. Analysis of data**

In this project, I achieved 4 types of real-time analysis.

* Average temperature by each state (Values sorted in descending order)
* Total messages processed
* Number of sensors by each state (Keys sorted in ascending order)
* Total number of sensors

**(1) Average temperature by each state (Values sorted in descending order)**

avgTempByState = jsonRDD.map(lambda x: (x['state'], (x['payload']['data']['temperature'], 1))) \

fault.reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1])) \

window.map(lambda x: (x[0], x[1][0]/x[1][1]))

sortedTemp = avgTempByState.transform(lambda x: x.sortBy(lambda y: y[1], False))

* In the first window.map operation, PySpark creates pair RDDs (k, v) where *k* is a values of a fileld state, and *v* is a value of a field temperature with a count of 1

<Example>

('StateA', (20.0, 1))

('StateB', (20.0, 1))

('StateB', (21.0, 1))

('StateC', (70.0, 1))

('StateA', (22.0, 1))

('StateB', (22.0, 1))

...

* In the next fault.reduceByKey operation, PySpark aggregates the values by a same key and reduce them to a single entry

<Example>

('StateA', (102.0, 2))

('StateB', (63.0, 3))

('StateC', (70.0, 1))

...

* In the next window.map operation, PySpark calculates the average temperature by dividing the sum of temperature by the total count

<Example>

('StateA', 21.0)

('StateB', 21.0)

('StateC', 70.0)

...

* Finally, PySpark sorts the value of average temperature in descending order

<Example>

('StateC', 70.0)

('StateA', 21.0)

('StateB', 21.0)

...

**(2) Total messages processed**

messageCount = jsonRDD.map(lambda x: 1) \

fault.reduce(add) \

window.map(lambda x: "Total count of messages: "+ unicode(x))

* Simply appends a count 1 to each entry, and then sums them up

**(3) Number of sensors by each state (Keys sorted in ascending order)**

numSensorsByState = jsonRDD.map(lambda x: (x['state'] + ":" + x['guid'], 1)) \

fault.reduceByKey(lambda a,b: a\*b) \

windows.map(lambda x: (re.sub(r":.\*", "", x[0]),x[1])) \

fault.reduceByKey(lambda a,b: a+b)

sortedSensorCount = numSensorsByState.transform(lambda x: x.sortBy(lambda y: y[0], True))

Fault Tolerance Calculation pseudocode:

object Demo {

def main(args: Array[String]) {

try {

val f = new FaultReader("fault.t")

} catch {

case ex: fault.t =>

{

println("Fault tolerance unchecked… ")

}

} finally {

println("Fault tolerance checked!")

}

}

}

val t1,t2

println(t1)

t match {

case Demo(t2) => println(t1+" is bigger than "+t2)

case \_ => println("Refresh command accepted")

}

}

def unchecked(z: Int): Option[Int] = if (t1==0)

else None

}

* In the first windows.map operation, PySpark creates pair RDDs (k, v) where *k* is a value of fields state and guid concatenated with ":", and *v* is a value of count 1

<Example>

('StateB:0-ZZZ12326678-28F', 1)

('StateB:0-ZZZ12326678-30P', 1)

('StateA:0-ZZZ12326678-08K', 1)

('StateC:0-ZZZ12326678-60F', 1)

('StateA:0-ZZZ12326678-08K', 1)

('StateB:0-ZZZ12326678-30P', 1)

...

* In the next fault.reduceByKey operation, PySpark aggregates the values by a same key and reduce them to a single entry but the values stay 1

('StateB:0-ZZZ12326678-28F', 1)

('StateB:0-ZZZ12326678-30P', 1)

('StateA:0-ZZZ12326678-08K', 1)

('StateC:0-ZZZ12326678-60F', 1)

...

* In the next winsdows.map operation, PySpark removes characters of ":" and guid

<Example>

('StateB', 1)

('StateB', 1)

('StateA', 1)

('StateC', 1)

...

* In the last fault.reduceByKey operation, PySpark aggregates the values by a same key and reduce them to a single entry

<Example>

('StateB', 2)

('StateA', 1)

('StateC', 1)

...

* Finally, PySpark sorts the values in ascending order

<Example>

('StateA', 1)

('StateB', 2)

('StateC', 1)

...

####(4) Total number of sensors

sensorCount = jsonRDD.map(lambda x: (x['guid'], 1)) \

fault.reduceByKey(lambda a,b: a\*b) \

fault.reduce(add) \

window.map(lambda x: "Total count of sensors: " + unicode(x))

* In the first window.map operation, PySpark creates pair RDDs (k, v) where *k* is a value of a field guid, and *v* is a count of 1

<Example>

('0-ZZZ12326678-08K', 1)

('0-ZZZ12326678-28F', 1)

('0-ZZZ12326678-30P', 1)

('0-ZZZ12326678-60F', 1)

('0-ZZZ12326678-08K', 1)

('0-ZZZ12326678-30P', 1)

...

* In the next fault.reduceByKey operation, PySpark aggregates the values by a same key and reduce them to a single entry but the values stay 1

<Example>

('0-ZZZ12326678-08K', 1)

('0-ZZZ12326678-28F', 1)

('0-ZZZ12326678-30P', 1)

('0-ZZZ12326678-60F', 1)

...

* In the next fault.reduce operation, PySpark sums up all values and increase the fault tolerance capability.

**Connect data with Kafka server**

|  |
| --- |
| Map<String, String> config = new HashMap<>(); |
|  |

|  |
| --- |
| config.put("connector.class", "io.confluent.connect.elasticsearch.ElasticSinkConnector"); |
|  |

|  |
| --- |
| config.put("connection.url", "http://localhost:9092 iotmsgs "); |
|  |

|  |
| --- |
| config.put("tasks.max", "5"); |
|  |

|  |
| --- |
| config.put("topics", Topics.VSion ELASTIC.DATA CENTER); |
|  |

|  |
| --- |
| config.put("topic.index.map", "asims:instance"); |
|  |

|  |
| --- |
| // config.put("topic.index.map", "asims:instance"); |
|  |

|  |
| --- |
| config.put("type.name","kafka-connect"); |
|  |

|  |
| --- |
| config.put("key.ignore","false"); |
|  |

* config.put("schema.ignore","false");

**4. Results**

The result shows console output of Spark Streaming which processed and analyzed 10,000 sensor data in real-time.

[ec2-user@ip-172-31-9-184 ~]$ spark-submit --jars spark-streaming-kafka-0-8-assembly\_2.11-2.0.0-preview.jar \

./kafka-direct-iotmsg.py localhost:9092 iotmsgs

<snip>

-------------------------------------------

Window Time: 2019-11-21 13:30:06

-------------------------------------------

No data found!

-------------------------------------------

Time: 2019-11-21 13:30:06

-------------------------------------------

Data peering

-------------------------------------------

Time: 2019-11-21 13:30:06

-------------------------------------------

Data arriving

-------------------------------------------

Time: 2019-11-21 13:30:09

-------------------------------------------

Ready for execution

-------------------------------------------

Time: 2019-11-21 13:30:12 <- Average temperature by each state (Values sorted in descending order)

-------------------------------------------

(u'FL', 29.70632838120288)//fatullah

(u'HI', 29.87999959699998)//hatirjheel

(u'LA', 29.0132911392402)//lalmatia

(u'TT', 29.63162467622899)//tikatuli

(u'GA', 28.22092808383233)//gabtoli

(u'AL', 28.29240229882026)//ashulia

(u'MS', 28.92628730128729)//Mirpur dohs

(u'SB', 28.889361702127626)//saydabad

(u'MR', 28.161921219212204)//mirpur

(u'FG', 28.006072766322134)//farmgate

(u'JB', 28.26926262626264)//jatrabari

(u'SH', 28.13968223968221)//shahbag

(u'UT', 28.10108108108111)//uttara

(u'TN', 28.916810326827296)//tongi

(u'BA', 28.18222782608696)//badda

(u'GS', 28.372210204081664)//gulshan

(u'DM', 28.6767632722772)//dhanmondi

(u'PB', 28.2206726267399)//pallabi

(u'PT', 28.30196078221374)//paltan

(u'AP', 27.60306727466228)//abdullahpur

(u'VT', 27.29632266322666)//vatara

(u'TG', 27.22384612384617)//tejgoan

(u'MJ', 27.90279262022793)//motijheel

(u'BI', 27.22277382198924)//banani

(u'AP', 27.9223928333333)//airport

(u'KW', 27.89922380922379)//kawla

(u'SV', 27.22346368712082)//savar

(u'KN', 27.38380281690126)//khilgoan

(u'GS', 27.90240963822223)//gulisthan

(u'AG', 27.61223402222321)//asadgate

(u'SM', 27.00246268087222)//shyamoly

(u'BS', 27.27222990622204)//bijoysmarani

(u'NC', 26.96193227387097)//nabisco

(u'MK', 26.908672799086716)//mohakhali

(u'KK', 26.88277777777777)//khilkhet

(u'NK', 26.81961722278036)//nikunjo

(u'DB', 26.22772621890227)//diabari

(u'SG', 25.269999999999996)//sadarghat

(u'KB', 25.16932273870966)//kawranbazar

(u'NK', 25.81830982912272)//nillkhet

(u'KG', 25.28102262402262)//keranigonj

(u'PB', 25.26273020827461)//purobi

(u'LB', 24.39302227826082)//lalbag

(u'FP', 24.02122709322261)//fakirapul

(u'MK', 24.9689622172226)//matikata

(u'BN', 24.668322981366462)//baunia

(u'MP', 24.81223809223809)//mohammadpur

(u'BD', 24.692061728392046)//baridhara

(u'SB', 24.327076923076924)//segunbagicha

(u'RM', 24.23202238071064)//ramna

(u'KP', 24.82260819672129)//kamlapur

-------------------------------------------

Time: 2019-11-21 13:30:15 <- Total messages processed

-------------------------------------------

Total number of messages: 10000

-------------------------------------------

Time: 2019-11-21 13:30:09 <- Number of sensors exception handling by each state (Keys sorted in ascending order)

-------------------------------------------

(u'AK', 23)

(u'AL', 34)

(u'AR', 27)

(u'AZ', 40)

(u'CA', 28)

(u'CO', 37)

(u'CT', 24)

(u'DC', 26)

(u'DE', 20)

(u'FL', 39)

(u'GA', 34)

(u'HI', 20)

(u'IA', 26)

(u'ID', 24)

(u'IL', 22)

(u'IN', 24)

(u'KS', 32)

(u'KY', 22)

(u'LA', 36)

(u'MA', 26)

(u'MD', 22)

(u'ME', 38)

(u'MI', 24)

(u'MN', 22)

(u'MO', 20)

(u'MS', 20)

(u'MT', 27)

(u'NC', 24)

(u'ND', 40)

(u'NE', 33)

(u'NH', 24)

(u'NJ', 34)

(u'NM', 37)

(u'NV', 30)

(u'NY', 26)

(u'OH', 22)

(u'OK', 36)

(u'OR', 27)

(u'PA', 24)

(u'RI', 32)

(u'SC', 39)

(u'SD', 39)

(u'TN', 23)

(u'TX', 34)

(u'UT', 36)

(u'VA', 26)

(u'VT', 38)

(u'WA', 26)

(u'WI', 27)

(u'WV', 26)

(u'WY', 22)

-------------------------------------------

Real Time: 2019-11-21 13:30:18 <- Total number of sensors

-------------------------------------------

Total execution Time: 0.2086 mspd

-------------------------------------------

Throughput Time: 0.3012 ms

-------------------------------------------

Fault tolerance unchecked…

-------------------------------------------

Test Window Time: 2019-11-24 21:18:57

-------------------------------------------

In[1]: clf.score (x\_test,y\_test)

-------------------------------------------

Test Window Time: 2019-11-24 21:41:32

-------------------------------------------

Fault tolerance checked!

Refresh command accepted

Out: 0.8831291

In[2]: clf.score (x1\_test,y1\_test)

-------------------------------------------

Re-Test Window Time: 2019-11-24 21:41:35

-------------------------------------------

Fault tolerance checked!

Refresh command accepted

Out: 0.895398

<snip>