Integrating Hadoop and Elasticsearch – Part 2 – Writing to and Querying Elasticsearch from Apache Spark

Posted by Prasanth Kothuri on Tuesday, 24 May 2016

Introduction

In the part 2 of 'Integrating Hadoop and Elasticsearch' blogpost series we look at bridging Apache Spark and Elasticsearch. I assume that you have access to Hadoop and Elasticsearch clusters and you are faced with the challenge of bridging these two distributed systems. As spark code can be written in scala, python and java, we look at the setup, configuration and code snippets across all these three languages both in batch and interactively.

Before reading this blogpost, I recommend you to read the part 1 in this series - Loading into and querying Elasticsearch and Apache Hive

Spark

Apache® Spark™ is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. Spark is becoming the defacto processing framework for analytics, machine learning and tackling a range of data challenges. Spark has several deployment modes like Standalone, Mesos and YARN, in this post we only focus on Spark on YARN (Hadoop) cluster deployment mode.

ElasticSearch

Elasticsearch together with Logstash for log tailing and Kibana for visualisation is gaining a lot of momentum as it is fairly easy to setup and get started with Elastic and its near real-time search and aggregation capabilities covers lots of use cases in the area of web analytics and monitoring (log & metric analysis).

Bridging Apache Spark and Elasticsearch

Spark lets you write applications in scala, python, java AND can be executed interactively (spark-shell, pyspark) and in batch mode, so we look at the following scenarios, some in detail and some with code snippets which can be elaborated depending on the use cases.

	spark-shell	scala	pyspark	python	java
read from ES	Y	Y	Y	Y	Y
write to ES	Y	Y	Y	Y	Y

Setup and Configuration

Following is the required setup and configuration for pilot and production purposes

Pilot

If you are getting started with testing ES-Hadoop connector then the following setup is suitable

- Download and Copy Elastic-Hadoop connector to /tmp on HDFS

```
wget -P /tmp http://download.elastic.co/hadoop/elasticsearch-hadoop-2.3.2.zip
unzip /tmp/elasticsearch-hadoop-2.3.2.zip -d /tmp
cp /tmp/elasticsearch-hadoop-2.3.2/dist/elasticsearch-hadoop-2.3.2.jar /tmp/e
hdfs dfs -copyFromLocal /tmp/elasticsearch-hadoop-2.3.2/dist/elasticsearch-ha
```

Production

For the production purpose you can copy the jar file into the SPARK classpath

Configuration

For most cases the following configuration is sufficient

```
es.nodes            # list of elasticsearch nodes, defaults to localhost
es.port             # elasticsearch port, defaults to 9200
es.resource         # es index, e.g; filebeat-2016.05.17/log
es.query            # es query, defaults to match_all
```

And most importantly if you ES cluster allows access only through client nodes (which is recommended) then the following parameter is necessary

```
    es.nodes.client.only     # routes all requests through the client nodes
```

The full list of ES-hadoop connector configuration can be found here

All the following code snippets are tested and valid for CDH 5.5 and Spark 1.5.1

spark-shell

reading from ES index

The following code reads elasticsearch index and creates spark RDD, by default the documents are returned as a Tuple2 with document id as first element and the actual document as second element

```
spark-shell --jars /tmp/elasticsearch-hadoop-2.3.2.jar \
            --conf spark.es.nodes="ela-n1" \

// import elasticsearch packages
import org.elasticsearch.spark._
// load elasticsearch index into spark rdd
val fbeat_rdd = sc.esRDD("fmem/log")

// print 10 records of the rdd
fbeat_rdd.take(10).foreach(println)
// count the records in the rdd
fbeat_rdd.count

// map it to different tuple, with each tuple containing list of values only
val fmem_t_rdd = fbeat_rdd.map{ case(id, doc) =&gt; (doc.get("dt").get, doc.get("s
// display the maximum memory consumed on each server
mapped.map{case (x,y,z) =&gt; (y,z)}.reduceByKey((a,b) =&gt; math.max(a,b)).collec
```

Alternatively, you can do the same thing in a much more efficient way using spark dataframe and spark SQL

```
spark-shell --jars /tmp/elasticsearch-hadoop-2.3.2.jar \
            --conf spark.es.nodes="ela-n1"

// load the elasticsearch index into spark dataframe
val fbeat_df = sqlContext.read.format("org.elasticsearch.spark.sql").load("fmem/lo
// inspect the data
fbeat_df.show(10)
// display the maximum memory consumed on each server
fbeat_df.groupBy("server").max().show()
```

writing to ES index

In the following code snippet, we read the csv (containing the memory consumption of flume agent on our servers) into rdd, map it to the schema and persist the rdd as elasticsearch index. As mentioned earlier use **es.nodes.client.only=true** if your elasticsearch cluster is configured to route traffic through client nodes

```
spark-shell --jars /tmp/elasticsearch-hadoop-2.3.2.jar \
            --conf spark.es.nodes="ela-n1" \
            --conf spark.es.nodes.client.only=true

//import elasticsearch packages
import org.elasticsearch.spark._

//define the schema
case class MemT(dt: String, server: String, memoryused: Integer)

//load the csv file into rdd
val Memcsv = sc.textFile("/tmp/flume_memusage.csv")

//split the fields, trim and map it to the schema
val MemTrdd = Memcsv.map(line=&gt;line.split(",")).map(line=&gt;MemT(line(0).trim.

//write the rdd to the elasticsearch
MemTrdd.saveToEs("fmem/logs")
```

scala

In the previous examples, we have been running our code from the interpreter, lets now put it in a file and run it with spark-submit. This process is more involved than it appears, is also a generic way of submitting spark applications.

writing to ES index

In the following code snippet we calculate the logging rate of our sensors on an hourly basis and persist this into elasticsearch index

Step 1) create the following directory structure

```
src
|-main
|---scala
```

Step 2) copy the following code to src/main/scala/

```
/* Write to ES from Spark(scala) */
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.elasticsearch.spark.sql._
import org.apache.spark.SparkConf

object WriteToES {
  def main(args: Array[String]) {

        val conf = new SparkConf().setAppName("WriteToES")
        conf.set("es.index.auto.create", "true")

        val sc = new SparkContext(conf)

        val sqlContext = new org.apache.spark.sql.SQLContext(sc)

        val sen_p=sqlContext.read.parquet("/user/hloader/SENSOR.db/eventhistory_00
        sen_p.registerTempTable("sensor_ptable")
        sqlContext.sql("SELECT cast(date_format(ts,'YYYY-MM-dd H') as timestamp) a
  }
}
```

Step 3) Copy the following to SparkToES.sbt

```
name := "Write to ES"
version := "1.0"
scalaVersion := "2.10.5"
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.5.1"
libraryDependencies += "org.elasticsearch" % "elasticsearch-spark_2.10" % "2.3.2"
```

Step 4) Package it with sbt (scala build tool)

```
sbt package
```

Step 5) Submit the spark application with spark-submit

```
spark-submit \
  --jars /tmp/elasticsearch-hadoop-2.3.2.jar \
  --class "WriteToES" \
  --conf spark.es.nodes="ela-n1" \
  --conf spark.es.nodes.client.only=true \
  --master yarn-cluster \
  target/scala-2.10/write-to-es_2.10-1.0.jar
```

reading from ES index

The following code allows you to read from elasticsearch index, again you need to follow the steps mentioned above to package it and submit to yarn.

```
/*read ES index from Spark(scala) */
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.elasticsearch.spark.sql._
import org.apache.spark.SparkConf

object ReadFromES {
  def main(args: Array[String]) {
        val conf = new SparkConf().setAppName("ReadFromES")
        val sc = new SparkContext(conf)
        val sqlContext = new org.apache.spark.sql.SQLContext(sc)
        val es_df=sqlContext.read.format("org.elasticsearch.spark.sql").load("file
        println(es_df.count())
  }
}

sbt package

spark-submit \
  --jars /tmp/elasticsearch-hadoop-2.3.2.jar \
  --class "ReadFromES" \
  --conf spark.es.nodes="ela-n1" \
  --conf spark.es.nodes.client.only=true \
  --master yarn-cluster \
  target/scala-2.10/write-to-es_2.10-1.0.jar
```

pyspark and python

reading from ES index (pyspark)

pyspark is the python bindings for the Spark platform, since presumably data scientists already know python this makes it easy for them to write code for distributed computing. The following code snippet shows you how to read elasticsearch index from python

```
pyspark --jars /tmp/elasticsearch-hadoop-2.3.2.jar

conf = {"es.resource" : "filebeat-2016.05.19/log", "es.nodes" : "ela-n1", "es.quer
rdd = sc.newAPIHadoopRDD("org.elasticsearch.hadoop.mr.EsInputFormat",\
&nbsp; &nbsp; "org.apache.hadoop.io.NullWritable", "org.elasticsearch.hadoop.mr.Li
rdd.first()
```

starting from the spark 1.4 you can also read elasticsearch index in more straightforward way using sqlcontext and dataframe

```
pyspark --jars /tmp/elasticsearch-hadoop-2.3.2.jar --conf spark.es.resource="fileb

# read in ES index/type "filebeat-2016.05.19/log"
es_df = sqlContext.read.format("org.elasticsearch.spark.sql").load("filebeat-2016.
es_df.printSchema()
es_df.show(10)
```

writing to ES index (python with spark-submit)

In the following example you can see how the python spark application can be run using spark-submit

Step 1) copy the following code to es_spark_write.py

```
# es_spark_write.py
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
if __name__ == "__main__":
    conf = SparkConf().setAppName("WriteToES")
    sc = SparkContext(conf=conf)
    sqlContext = SQLContext(sc)
    es_conf = {"es.nodes" : "ela-n1","es.port" : "9200","es.nodes.client.only" : "
    es_df_p = sqlContext.read.parquet("/user/hloader/SENSOR.db/eventhistory_000000
    es_df_pf= es_df_p.groupBy("element_id").count().map(lambda (a,b): ('id',{'elem
    es_df_pf.saveAsNewAPIHadoopFile(
    path='-',
    outputFormatClass="org.elasticsearch.hadoop.mr.EsOutputFormat",
    keyClass="org.apache.hadoop.io.NullWritable",
    valueClass="org.elasticsearch.hadoop.mr.LinkedMapWritable",
    conf=es_conf)
```

Step 2) submit the python application using spark-submit

```
spark-submit --master yarn-cluster --jars /tmp/elasticsearch-hadoop-2.3.2.jar es_s
```

Conclusion

This blog post goes into the details of how to query Elasticsearch from spark interactively and in batch using scala, python. Most importantly also shows how to write to Elasticsearch index as this opens the possibilities of performing distributed batch analytics on large scale using spark and visualizing the results using Kibana and Elasticsearch. You can refer to elastic-hadoop connector documentation for further options and possibilities.

Tags: spark Hadoop Elasticsearch scala python pyspark

Add new comment

Your name *

E-mail *

The content of this field is kept private and will not be shown publicly.

Homepage

Subject

Comment *

- No HTML tags allowed.
- Web page addresses and e-mail addresses turn into links automatically.
- Lines and paragraphs break automatically.

More information about text formats

CAPTCHA

This question is to prevent automated spam submissions. Please note that you are not authenticated and therefore your comment will be held in a moderation queue and will not be published until it is approved.

I'm not a robot

reCAPTCHA

Privacy - Terms

Save

Also by Prasanth Kothuri

Large Scale data reduction of AWAKE experiment data with Apache Spark and Notebooks

Benchmarking Apache Kafka on OpenStack VM's

Offline analysis of HDFS metadata

Real-time visualisation of Hadoop resources

Integrating Hadoop and Elasticsearch – Part 2 – Writing to and Querying Elasticsearch from Apache Spark

Integrating Hadoop and Elasticsearch - Part 1 - Loading into and Querying Elasticsearch from Apache Hive

Using SQL Developer to access Apache Hive with kerberos authentication