**Real-time sport event detection on Twitter**

**Overview**

In this project, there are two major parts. At the first part, data will be capture and filter by location and time in Kafka connect and sent to the Kafka. After being partitioned by Kafka, data from message topic will be consumed by Spark streaming and received data pre-processing in it. At the second part, a certain amount of data (we use data from one weekend) sent to Hadoop and Mahout to create initial clusters. To identify the local sport event in Chicago, both the output of Spark streaming and Mahout will be put into Spark engine to conduct clustering and classification. The result of event analysis will be sent back to Kafka and organized by event. Kafka will broadcast these real-time events to the user by Amazon SNS.

**Kafka**

Twitter is one of the most popular microblog in the world, which produce millions of tweets every second and requires sound technologies to store and process the data from it. Since our goal is to detect the real-time sport events from twitter instead of batch data, we are looking for a technique that provides a unified, high-throughput, low-latency platform for handling real-time data feeds. Therefore, in the first step of the event detection, we choose to use Kafka.

Kafka is a distributed publish-subscribe system that can decouple data pipelines. As a broker between producer and consumer, Kafka maintains feeds of data in topics, and partition and replica those topics across multiple nodes, which prevent data loss in the process. In this case, there are two topics in the Kafka – message topic and sport event topic. Data from event topic are the cluster events which come from the Spark core after cluster and classification in the core.

When Twitter users publish tweets on Twitter, the server will immediately send the message to Kafka stream. The message will be order chronologically and partitioned subsequently. Partitions from each server will be pushed to the consumer group of next component (we are using Spark streaming as next component) and each partition will only be delivered by one consumer instance in each group. Each consumer instance represents a separate machine.

**Spark Streaming**



In the next step, we need a component that provides scalable, high-throughput and fault-tolerant stream processing for real-time data streams and apply complex algorithm with the data. Here we choose to use Spark Streaming.

Spark Streaming is an extension of core Spark API and able to ingested data from many sources such as Kafka, Flume, HDFS and Kinesis. When live data goes in Spark, Spark Streaming first separate them into data batches and deliver them to Spark Engine to do further data processing and generate final result in batches.

As the consumer of Kafka, Spark Streaming use data from Kafka partition and divides the data stream into batches called DStreams, which is a sequence of RDDs internally, based on a certain time interval. DStreams represent the stream of input data received from the data source (Kafka in this case). Each input DStream is associated with a receiver which responsible for receive data from the source and stores it in Spark’s memory for processing. Each RDD is a distributed collection of element, and it is contained in one parallel computing cluster (one machine). In RDDs, data is partitioned and stored in memory instead of disk, which is the reason that Spark Streaming perform better than MapReduce since it minimizes the time in I/O.

After the data is read in the Spark Streaming, we can transform the data before putting data into Spark engine and clustering model. In this step, ‘flatMap’ function will be used to tokenized the message into words. We choose the specific event detection which is to detect the main three kind of sports events in Chicago. We will define the trending and interesting by the volume of the messages, which represents the number of people discussing the event.

**Mahout, K-Means clustering:**

This step is to discover the initial clusters. To detect specific events using features of tweets content (e.g., hashtag, term statistics, time references, a bag of keyword triggers of related sports and teams) and group similar content via clustering into initial clusters.

*Input:* Tweets of the recent weekend that be uploaded to HDFS: hashtag, time reference, terms and a bag of words including sports names and sports team names.

*Output*: Initial clusters consist of sport event-based clustering of messages

Here we firstly consider traditional text processing steps such as stop words elimination and stemming to the messages. However, these text processing steps will not be applied to the hashtag keywords.

We pick Mahout because we are mining the historical data but not the real-time ones in this step. Mahout is efficient enough to group the similar tweets together via K-means clustering, the similarity of two tweets is measured by calculating the distance between their vector representation: TF-IDF. But their algorithm only deals with the tweets with hashtag or containing the keywords from the bag of words. Clustering is guided by the term popularity which is based on a weighted combination of term similarity measures, including complete string matching, and adjacency and equality indicators scaled by the inverse document frequency(IDF). We choose K-Means because it is most popular and proven to be effective. We will decide the K according to the volume of tweets uploaded to the HDFS and the effectiveness of the clusters generated.

**Spark Engine**

This step is to associate real-time Tweeter streams with initial clusters of events to extract discriminating features of clusters and to build an event classifier

**Single-Pass incremental clustering**

*Input:* Existing clusters from Mahout and batches of real-time RDDs from the spark streaming

*Output:* Event clusters consist of discriminating features

We firstly also process text transformations such as stop words elimination and stemming to the messages.

We use the Spark engine in this step because we need to deal with the real-time Twitter streaming with the single-pass incremental clustering method. The Spark engine is the most efficient software. This single-pass incremental clustering method has been shown to be an effective technique for event detection in textual documents based on the element’s similarity to any existing clusters. It does not require a priori knowledge of K and groups the element similar to any existing cluster using the centroid similarity approach by aggregating and average the TF.IDF. If there is no cluster to which the element similar, A new cluster will be generated. We define a similarity metric for each feature as a TF.IDF weight vector and use the cosine similarity metric.

**Classifying the Real-World Event v/s Rest.**

Based on the different features of the clusters which are formed in the previous step, we can build a classifier model, which classifies real world event vs rest. The different types of features are:

* Social Interaction Features (@-Mentions, Retweet, Replies)
* Topic Coherence Features (Cluster will be revolving around one central topic)
* Trending Behavior (There will be exponential growth in top-words of clusters)
* Increase in Volume Over time (Velocity & Volume of Cluster growth)

We use these features to train the classifier which distinguishes event and non-event clusters by applying standard machine learning techniques. The classifier predicts which cluster corresponds to events at any point in time. We selected Supported Vector Machine from Weka toolkit for RW-event and we can fit the Logistic Regression model to the output of the SVM to get the probabilities of class assignment.

At the end of each hour, we select 20 fastest growing clusters per hourly volume and publish it to Kafka topics.

**Push Event Information on various devices, including phones, tablets, and desktop computers.**

We use Amazon’s Simple Notification Service over custom messaging solutions to broadcast newly detected real-world events to multiple destinations on multiple platform.

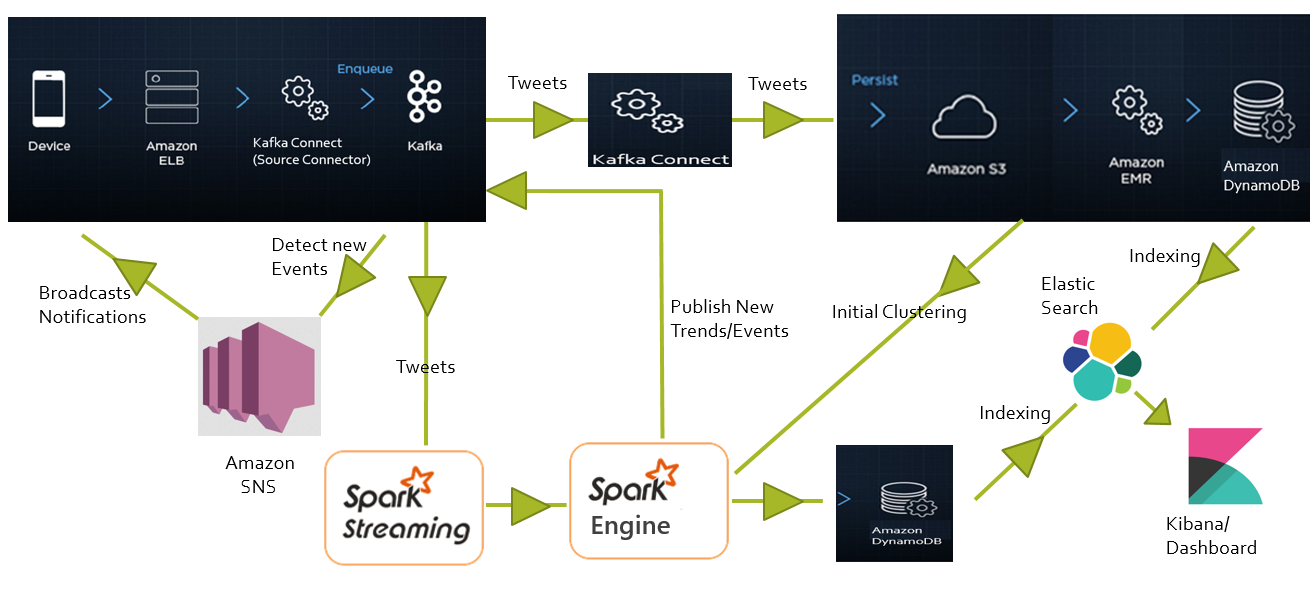
It usually takes 5 mins for a newly created topic to be visible to the consumers. So, Kafka consumers can poll on the topic metadata for every 5 mins to know about the newly formed clusters and its events. Once the consumer knows about the new event, we can set some rules and policies in Amazon SNS to broadcast messages to different clients. Reasons for using Amazon SNS are listed below.

* *Reliable***:** It runs within Amazon’s Datacenters and messages are stored redundantly across many servers and data centers.
* *Scalable***:** It handles unlimited number of messages at all the time.
* *Simple:* Just 3 API’s Create, Subscribe and Publish.
* *Flexible:* Allows applications and end-users on different devices to receive notifications via Mobile Push notification (Apple, Google and Kindle Fire Devices), HTTP/HTTPS, Email/Email-JSON, SMS
* *Secure:* It ensures that messages and topics are secured against unauthorized access.
* *Inexpensive:* Pay per use concept.

**Elastic Search and Kibana Dashboard.**

The output of Spark Engine is classified events and it will be Stored into Amazon’s DynamoDB and the data is indexed using Elastic Search, which is distributed Search engine which is built on top of Apache Lucene. We are using Kibana for real-time Dashboard which is a product of www.elasticsearch.co which integrates well with elastic search stack.

**Putting it All together we have come up with below Scalable Data Pipeline.**



**Advantages and Disadvantages of the Data-Pipeline:**

* Advantages
  + Scalable
  + Reliable
  + Inexpensive
* Disadvantages
  + Periodically update the initial input of ‘K’
  + Less customizable