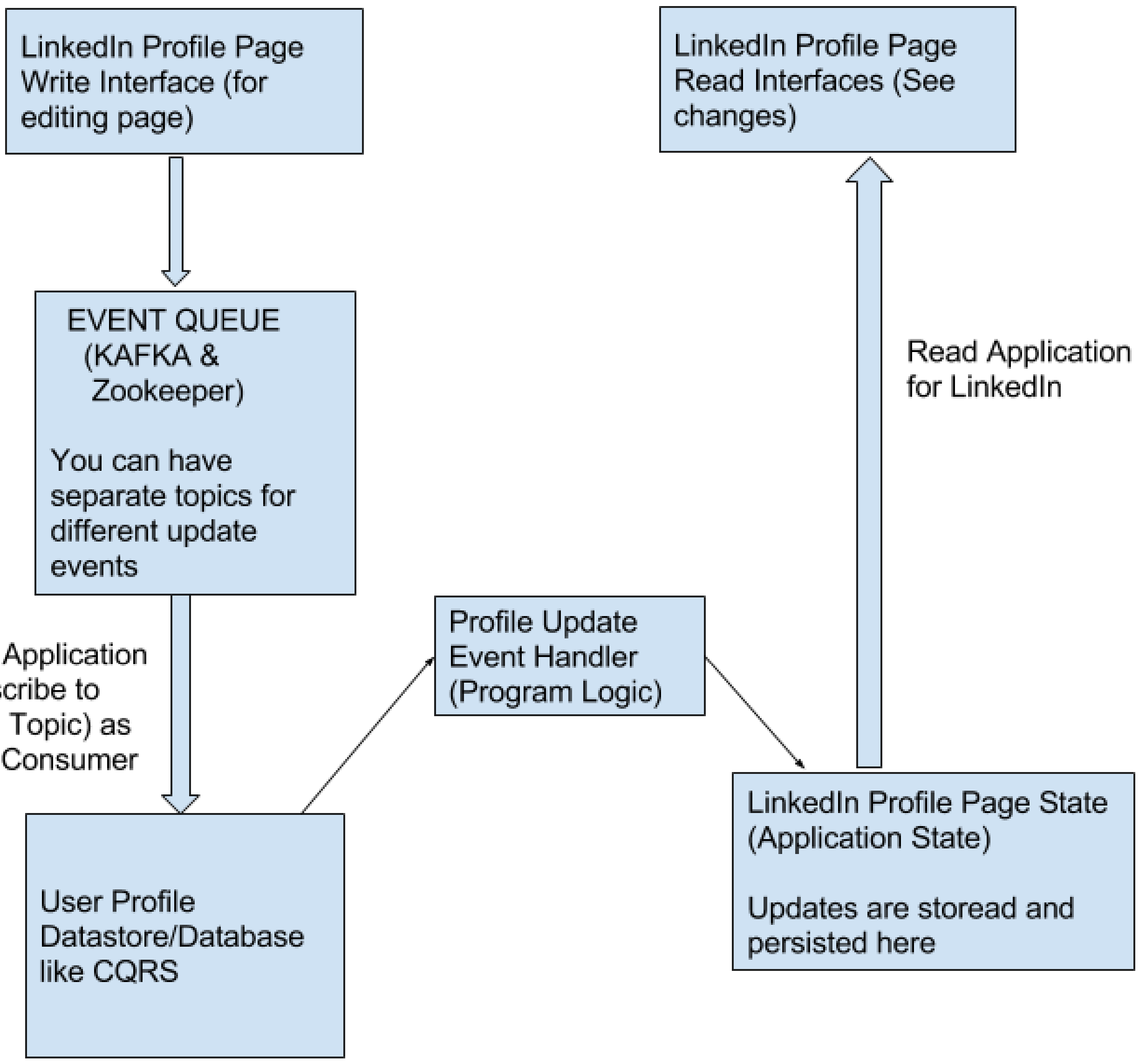
UNI: kb2896
Name: Kunal Baweja
COMS E6998 Cloud Computing and Big Data

1. LinkedIn Profile Page (Kafka Design)

a. 
LinkedIn Profile Page Write Interface (for editing page)
LinkedIn Profile Page Read Interfaces (See changes)
EVENT QUEUE (KAFKA & Zookeeper)
You can have separate topics for different update events
Read Application for LinkedIn
Write Application (Subscribe to Kafka Topic) as KafkaConsumer
Profile Update Event Handler (Program Logic)
User Profile Datastore/Database like CQRS
LinkedIn Profile Page State (Application State)
Updates are storead and persisted here

a) User makes and edit on their profile page
b) The user update is sent to the Kafka Server (with Zookeeper Service) which acts as an event queue
c) The server side application logic of our website (LinkedIn here) subscribes to different topics from kafka, each representing different type of profile update.
d) The server side logic extract profile updates as a KafkaConsumer and logs them into a Event Store database like CQRS
e) Next the server reads from the event store database and performs the update on user profile in the Profile Update Event Handler (Applies update to profile)
f) Stores update profile in permanent profile state storage/application state
g) User can see updates applied from the application read interface

2.
Links RDD

    Links = sc.map([(A,[C]), (B,[A]), (C,[A,D]), (D,[B,A])]).cache()

Ranks RDDs

    Ranks = sc.mapValues(lambda r: 1.0)

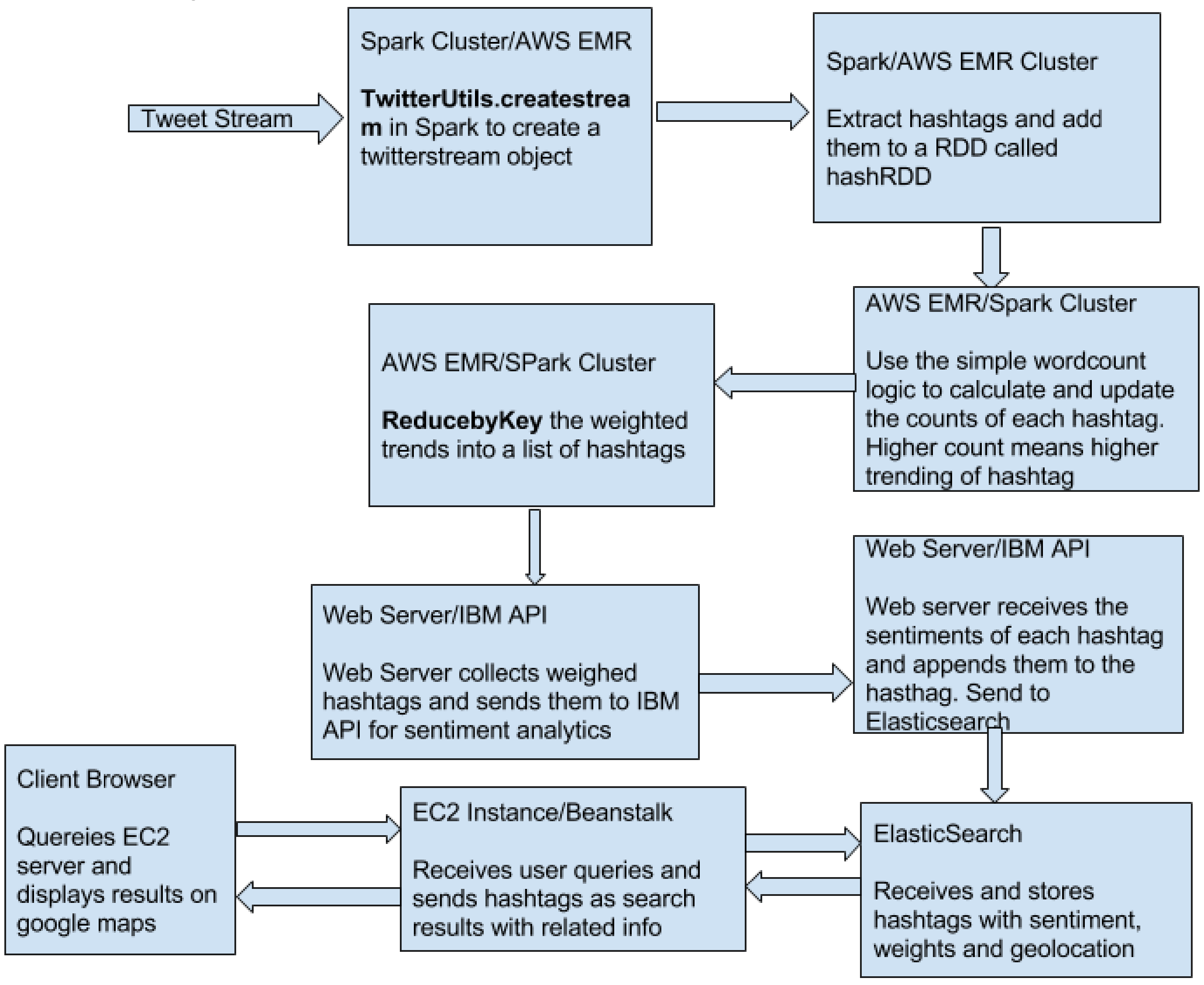
Steps:

1. User links.join with rank that generates RDD X as [(A,(C, 1)), (B, (A , 1)), (C, (A,1)), (C, (D,1)), (D, (A,1)), (D, (B,1))]
2. Use flatMap on the X RDD to flatten the data and pass to compute_contribs functions to calculate weighted contributions
3. Contribs RDD: [(C, 1), (A, 1), (A, 0.5), (D,0.5), (B, 0.5), (A, 0.5)]
4. Next we reduce on contribs using reduceByKey and sum the page ranks
5. ranks RDD = [(A, 2), (B, 0.5), (C, 1) ,(D, 0.5)] //after reduce by key operation
6. **ranks RDD = [(A, 1.85), (B, 0.575]), (C, 1) ,(D, 0.575)]**

At step 6 we get the ranks of each page which we can sort in descending order to get the relative ranking

3rd question on next page

3. Twitter Hashtags


Tweet Stream
Spark Cluster/AWS EMR
TwitterUtils.createstream in Spark to create a twitterstream object
Spark/AWS EMR Cluster
Extract hashtags and add them to a RDD called hashRDD
AWS EMR/Spark Cluster
Use the simple wordcount logic to calculate and update the counts of each hashtag. Higher count means higher trending of hashtag
AWS EMR/SPark Cluster
ReducebyKey the weighted trends into a list of hashtags
Web Server/IBM API
Web Server collects weighed hashtags and sends them to IBM API for sentiment analytics
Web Server/IBM API
Web server receives the sentiments of each hashtag and appends them to the hasthag. Send to Elasticsearch
ElasticSearch
Receives and stores hashtags with sentiment, weights and geolocation
EC2 Instance/Beanstalk
Receives user queries and sends hashtags as search results with related info
Client Browser
Quereies EC2 server and displays results on google maps

References:

1. https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/
2. http://spark.apache.org/docs/latest/streaming-programming-guide.html
3. https://spark.apache.org/docs/1.0.0/api/java/org/apache/spark/streaming/twitter/TwitterUtils.html