Static to Dynamic - Incremental Computations

Author: Anikesh and Sairamana
Mentor: Prof. K. Mani Chandy
Co-Mentor: Dr. Julian Bunn

Abstract: Internet of Things and 'Big Data' has become increasingly prevalent with a vast range of applications from cyber-security to health monitoring. The StreamPy package enables the easy application of functions and algorithms for static data to be extended to dynamically changing data streams. Thus it proves to be of high utility for real-time streaming analytics. We demonstrate this utility by the implementation common real-time analytics patterns with real life use cases done under Complex Event Processing and other such technologies. In the process, we developed a library for counting algorithms, streaming graph algorithms as well as an abstraction for database interaction among other pattern implementations. We also intend to implement a parallel processing version of these implementations to make it truly compatible for Big Data. Such a package would prove to be very powerful, as even though technologies such as Apache Storm and Spark Streaming are available for real-time analytics, it has been observed that users prefer to implement real-time analytics specific for their application from scratch due to the lack of coverage of the common analytics patterns.

Introduction:
There are mainly two types of problems that are of primary concern while working with real time analysis of data streams. One is memory efficient computation on the inflowing data streams. Quantities such as top-k nodes in a graph stream, or finding frequency of a particular element in a stream is very relevant in domains such as social networks or data packet streams on the web. However brute force approach is not an optimized approach for these computations due to time complexity involved and the huge amount of memory required. In this regard, we have worked on 'Approximation Algorithms' and the DEBS 2016 grand challenge problem set. The other type of problem is event detection where the challenge is to detect an anomaly in real time. In this regard, we have worked on 'Interaction with Database' and 'Data correlation, Missing events and Erroneous data'. The remaining portion of the paper describes these topics in more detail.

Approximation Algorithms: The importance of these approximate algorithms is that they provide a strong bound on the query output while being highly memory efficient . For example in a continuous data stream with million distinct elements it will not be efficient to maintain counters for each element so in such cases the sketching algorithms come into play. The motive behind selecting these particular algorithms is because of their wide range of applications.
Thus, these algorithms are useful where computation of the exact solution is too expensive both in memory and in time. Their solution has an (ε, δ) guarantee, i.e., they provide an ε bound on the output with probability $1 - \delta$. The algorithms we implemented are:

•Misra-Gries Algorithm: Returns elements with frequency greater than a certain fraction (specified by user) of length of stream.
•Count Min Sketch: Approximates non-negative frequencies for each distinct element
•Count Sketch: Approximates frequencies (can be negative) for each distinct element

•Flajolet-Martin Algorithm: Approximates the number of distinct elements in a stream
•Lossy Counting: Approximates frequencies of each distinct element
•Space Saving Algorithm: Keeps highly accurate frequency estimates for m elements (user specified), and also keeps track of errors

Interaction with Database: Often many real time data analytics applications require access to the data that is stored in our hard disks. Usually data is stored using Relational Database Management Systems which is queried using Structure Query Language (SQL). To facilitate the interaction of data stream with databases an abstract class called Databases is introduced. This abstract class was modelled along the lines of Continuous Query Language [10], and requires the programmer to provide only a function that contains the logic that needs to be executed when a new event arrives. However many practitioners are usually accustomed to using SQL for querying the database. So in order to ease programming this abstract class is designed such that it can handle even if the input function's logic is written in SQL. Another distinctive feature (optional) of this abstract class is that it also generates a database from the input data which is then made available to the user defined function. This Databases class is particularly relevant to big data analytics as big data whose one of the characteristics is unstructured data is usually stored in databases such as MongoDB. Also to illustrate the use of this class an application was developed that receives a stream of tuples consisting of IP's, urls being accessed, the date and time of access within a local network as input and searches in an SQL database containing blacklisted websites whether the website being accessed is blacklisted or not. If the website accessed is blacklisted one then this record is saved to a database.

Data correlation, Missing events and Erroneous data: It is a useful pattern into which many real time data analytics applications can be categorized into. This involves correlating the data between multiple streams and detecting if there are any missing events or if the input data is corrupt. An application called Transaction streams has been developed that demonstrates how such patterns could be implemented in StreamPy. This application involves an input stream that consists of requests from clients to a server whose clients are categorized into three groups 'Platinum', 'Gold 'and 'Silver'. Depending upon the category of the client, the system is supposed to respond within some fixed time. This application monitors the response stream and correlates these events with the request stream to verify if there are timely response to the requests and if case a request receives an untimely response then this request 's id is passed on to the output stream . A real life application of this application would be analyzing the quality of customer service.

DEBS 2016 Grand Challenge:

DEBS grand challenge provide an uniform evaluation criteria for Event based systems and this year's grand challenge aims to evaluate the event based platforms for real time analysis of large data in the context of graph models. The theme for the grand challenge was Social Network Analysis and the following were the problems that were proposed

(i) Identification of the post that presently trigger the most activity in social network
(ii) Identification of the large communities that are presently involved in a topic

A brief description is provided about the targeted problems below however complete description about them can be found in [1].

Top Three Scoring Posts:

The aim of this query is to identify the top three active scoring posts and to output them whenever there is a change. The following metric has been followed to calculate the total score of the post. Whenever a new post arrive an initial score of ten is assigned to it and this score decreases by 1 for every 24 hours. Similarly whenever a new comment arrives an initial score of ten is assigned to it, which again decreases by 1 for every 24 hours. The score assigned to posts and comments is nonnegative and the total score of the posts is the sum of the score of the post and the comments associated with it. Once the total score of a post becomes zero it is no longer considered active.

A general way to approach would be maintaining total score for each post and updating it whenever a new event arrives. This approach is not recommended as updating large number of posts in real time would require high computation power. Our solution to this problem is based on the fact that “When the score of a post decreases nothing needs to be done if the post is not in the top three.”

Data Structures:

SCORES: scores is a 1-D Array of sets, where score[j] is a set of posts with score of j or less than j. Note that the set, score[j], may include a post whose score is less than j, however score[j] will never include a post with a score higher than j.

POSTS:*posts* is hash table of posts which associate the following tuple with each post id.

1. The last updated score for the post.
2. The date at which the score for this post was last updated
3. The time at which this post was created
4. The set of comments associated with this post

COMMENTS: *comments* is a hash table of with comments which associate the following with the comment ids.

1. The last updated score for the comment
2. The latest day on which the score was updated
3. The time at which the comment was created
4. The post associated with this comment

HIGHEST_SCORES:*highest_scores* is a list of N integers such that for each element “j” that belongs to this list score[j] is a non-empty set.

TIMESTAMPS: Each one of the top k post’s and their associated comments time arrival is stored in this list. Each member of this list is a tuple of form (time, is_post, post_id)

So whenever a new event occurs which could be either arrival of new post or new comment the following steps are followed

Arrival of new post:

1. The scores of top k posts and their associated comments are updated using the list timestamps and if there is a change the new top k posts are updated.
2. An entry is created in the posts table with the post id as key
3. This post is added to the set corresponding to default score in the array scores
4. If number of scores in the highest_scores is less than N and if default score is not a part of these scores then default score is inserted into this and the timestamps list is updated according

Arrival of new comment:

1. The scores of top k posts and their associated comments are updated using the list timestamps and if there is a change the new top k posts are updated
2. An entry is created in the comments table and the score of the associated post is updated
3. If this post score is greater than the top k posts lowest score than corresponding changes are incorporated

Community Interest Query

The goal of this query is to identify the top k comments that interest the largest communities and are not created prior to d seconds. A community for comment is defined as the group of users such that each user is friend of all other user. So the relevant streams for this query are the friendship, likes and comments. This problem can modeled as a graph problem by considering all the users who liked a particular comment as nodes and friendship between them as the edges, once this graph of the range of the comment is the number of nodes in the largest connected component. However an important point to note is that the graph is dynamic and time required to find the largest connected component increases exponentially with the number of nodes. So in order to reduce the computations smart bounds are used to eliminate the outliers. The following facts have been used to prune the solution space.

1. For an user to be a part of clique of size m he should have at-least m friends
2. The user should have at-least m-1 friends who in turn have at-least m-1 friends
3. Also the user should have at-least m-1 friends who like the comment and satisfy the above two points

Implementation of Solution and data structures used:

1. User_comments: A hash table with user id as the key which is associated with the comment ids that the user liked

2. User_friends: A hash table with user id as the key which is associated with the user ids who are his friends (analogous to adjacency list) that is sorted in a decreasing order.
3. Cliquesize_to_comment : It is a 1D array of sets such that all the elements belonging to the set corresponding to index j have clique size j or less than j.
4. highest_clique: An ordered list of highest N non empty sets in cliquesize_to _comment.
5. Timestamps: A list of timestamps for each second in the day. The list will comprise of 86400 entries and each entry is a list of comments made at that time

New_Friendship:

1. Whenever a new friendship is established update the user_friends accordingly
2. Collect all the comments that are affected by this event and for each comment that is presently having a clique of size m-1 check if the above 3 conditions are met
3. If yes, then do the clique computation and update the results
4. For the duration from older clock to newer clock, update the duration of the comments lying in between these 2 indices in timestamps. If any highest clique comments exceed the duration d, remove from highest_clique and updatehighest_clique in efficient way.

New_like:

1. Whenever a new like is received by a comment there is a chance that its clique size could increase
2. For the new user who liked the comment should satisfy the three condition to increase the clique size of the comment
3. If the conditions are satisfied by the user then clique computation is carried out to check if there is an increase in the size of clique and the corresponding changes are incorporated

References:

[1] B. Boyer and J. Moore. A fast majority vote algorithm. Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas, Feb. 1981.
[2] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In ACM PODS, 2004.
[3] G. Manku and R. Motwani. Approximate frequency counts over data streams. In International Conference on Very Large Data Bases, pages 346–357, 2002.
[4] Amit Chakrabarti ,Dartmouth College CS85: Data Stream Algorithms Lecture Notes, Fall 2009.
[6] Data Stream Algorithms by S. Muthu Muthukrishnan
[7] Finding Frequent Items in Data Streams by Moses Charikar, Kevin Chen, and Martin Farach-Colton3
[8] Sketch Algorithms for Estimating Point Queries in NLP by Amit Goyal and Hal Daume III
[9]Srinath Perera and Sriskandarajah Suhothayan. Tutorial: Solution Patterns for Real time Streaming Analytics. In ACM DEBS -2015
[10]The CQL continuous query language: semantic foundations and query execution,

Acknowledgement:

We are grateful to our mentor Prof. K. Mani Chandy for the guidance he provided us during our fellowship as well as the work that has followed. We also would like to acknowledge the useful inputs and advice provided to us by our co-mentor Dr. Julian Bunn during the course of the fellowship, as well as Rahul Bachal and Ker Lee Yap (students at Caltech) for their suggestions early in our project. We would also like to acknowledge the SFP team for their help throughout our stay in Caltech.