DEBS Grand Challenge

Technical Approaches - 2016

The ACM DEBS Grand Challenge is a yearly competition where the participants implement an event-based solution to solve a real world high-volume streaming data problem. The focus of the Grand Challenge in 2016 is on processing of data streams that originate from social networks. Hence, the data represents an evolving graph structure.

This year’s grand challenge involves developing a solution to solve two real world problems by analyzing a social-network graph that evolves over time.

1. The identification of top-3 active posts according to their scores. The scores are increased with the arrival of new post-related comments and are decreased by the expiration of related-comments and own score. This suggests a non-linear window model.

2. The identification of social contagion in dynamic streaming settings. Given a window and a value of k, determine the top-k comments shared/liked between friends in the network. The size of the largest clique determines the influence of a certain comment posted by a person.

The ranking of the solutions is carried out by measuring their performance using two performance metrics: (1) throughput and (2) average latency.

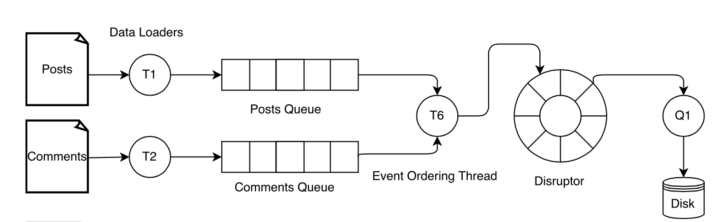
Some of the challenges encountered in this year’s competition were:

* non-linearity of the expiration of the elements, present in the first query
* in the second query, unlike traditional approaches where no persistent data is stored over the stream, the friendship graph must be persisted throughout the system execution

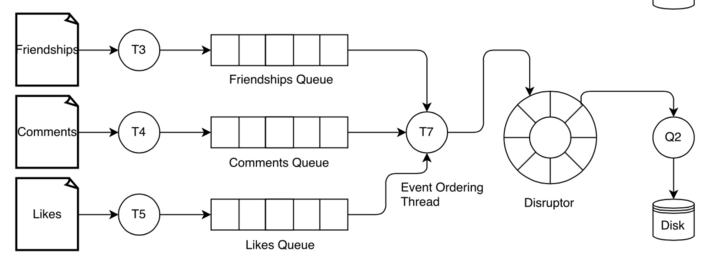
The top solutions presented at the DEBS Grand Challenge 2016 propose specific architecture and data structures in order to solve the problems. In what follows are presented some of the techniques and approaches used by the top teams from 2016.

* One of the solutions, proposed by Kammoun, Abderrahmen et al.[2], highlights the usage of bounds -- upper and lower --, in order execute expensive computations only when required. For Query 1, they use a bound based on score decay and lazy evaluation of the non-linear window elements. For Query 2 Turan's theorem is used to limit the clique computation.
* Another team proposes a graph-based Complex Event Processing system GraphCEP[4]. With this system, they try to overcome the limitations of graph-structured data by further parallelizing individual operator instances using modern graph processing systems. These systems partition the graph data and execute graph algorithms in a highly parallel fashion, for instance using cloud resources.
* One other solution proposes an original program to efficiently calculate 2 continuous top-k queries on real-time social-network graph data[5]. The implementation tries to prevent processing of unaffected events by designing the algorithms to efficiently maintain the spare list of candidates of the top-k results.
* In another solution StreamMine3G[6] is used, a distributed, highly scalable, elastic and fault tolerant event stream processing (ESP) system. It comes with a simple MapReduce like programming interface, however, supports real time computation in contrast to Hadoop. Innovative aspects of the implementation include highly optimized data structures that lower the amount of lookups and traversals, and a deterministic data partitioning and processing scheme that allows the system to scale without bounds in an elastic fashion while still guaranteeing semantic transparency. In order to better utilize nowadays many-core machines, they propose a pipelining scheme in addition to data partitioning.
* In another solution, WSO2 CEP[3], an open source Complex Event Processing Engine, was used to solve the problem. On a 4-core/8 GB virtual machine, the solution processed 90,000 events per second with a mean latency of 6 ms for query 1. For query 2 it processed 210,000 events per second with a mean latency of only 0.3 ms. WSO2 Complex Event Processor (WSO2 CEP) helps identify the most meaningful events and patterns from multiple data sources, analyze their impacts, and act on them in real time.

As this team provided a more detailed explanation of the solution, we will analyze in what follows the architecture that they used.



Query 1: Architecture[3]



Query 2: Architecture[3]

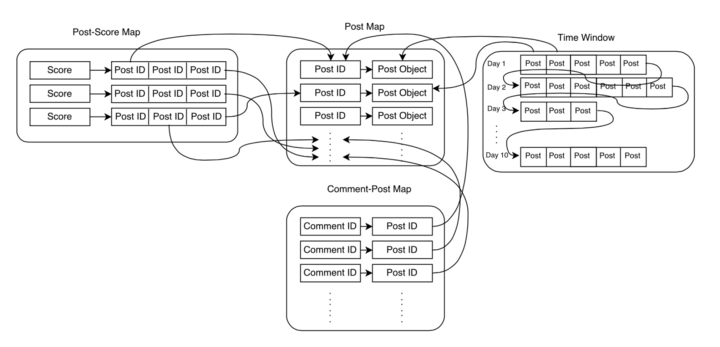
Each query is processed as a pipeline where the pipeline consists of three phases: 1) data loading, 2) event-ordering and 3) processing.

For each query there is a dedicated data-loader thread (T1, T2, T3, T4, T5) for reading each data stream. The data read by a data loader is placed in a blocking queue. The blocking queue provides a place for the data loader to store events without having to wait for the event-ordering threads to consume them. The loader threads also keep the event-ordering threads busy by providing a backlog of events to consume (note: the maximum capacity for a blocking queue can be specified when the queue is constructed.).

There are two dedicated event-ordering threads for ordering query 1 (T6) and query 2 (T7) streams. These threads fetch the events from queues and order them based on their event timestamps. Note that events in queues are already ordered based on their timestamp (the timestamp here refers to logical time). The purpose of the ordering done in this phase is to ensure that the merged event-stream that is sent to event processor is ordered based on their timestamps). For example, in the case of query 1, it orders comment and post streams and place these in the ring buffer. The event processing threads (Q1 and Q2) fetch the events from the ring buffer and process them according to the query logic. Note that event-ordering threads act as producer threads while the processing threads represent consumers.

Query 1: Data Structure

As illustrated below the query 1 data structure consists of three maps (post-map, post-score map and comment-post map) and ten time windows.



Query1: Data Structure[3]

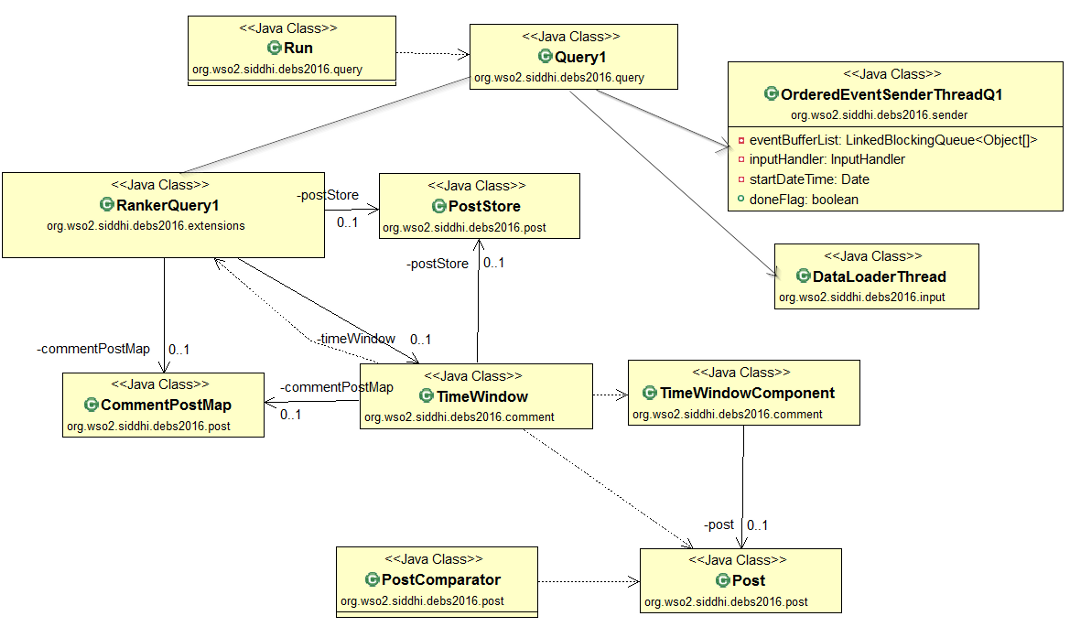
* Post-map (hashMap): mapping between post\_id and post
* Post-score map (bounded sorted multi-map of size 3): mapping between post score and post\_id
* Comment-post map (hashMap): mapping between comment\_id and post\_id
* 10 time windows: These time windows allow to model the expiration of posts and comments (note: the initial score of a post/comment is 10. This score gets decremented by 1 every 24 hours). The first time-window stores all the posts and comments whose ages are less than or equal to 24 hours. Similarly, the second window contains posts and comments whose ages are greater than 24 hours and less than or equal to 48 hours and so on.

Query 1: Algorithm

When a new post arrives, it is registered in the post and post-score maps. Then the time-windows are processed. Each time-window contains many time-window objects where each object stores information regarding a comment or post. When a post arrives the timestamp of the new post is evaluated against the timestamp of time-window objects. If the age of the time-window-object > upper limit of the time-window (i.e. 24, 48, etc.), the time-window object is transferred to the next time window and so on. The movement of a time-window object from one time window to another window indicates a change in a particular post score. When such a change is detected, the post-score map is updated. After processing the time windows, the new post is given a score of ten and it is placed in the first time-window. The post-score map now contains the top three scores and their corresponding posts. The final top three posts are obtained by processing this map.

When a new comment arrives, if the comment is for a post then it is registered in the comment-post map. If the comment is for a comment then post id of the comment is obtained from the comment-post map by doing a look up on the comment id. Then the new comment is registered in the comment-post map. As it was done for a post, the time windows are processed and the new comment is then placed in the first time window.

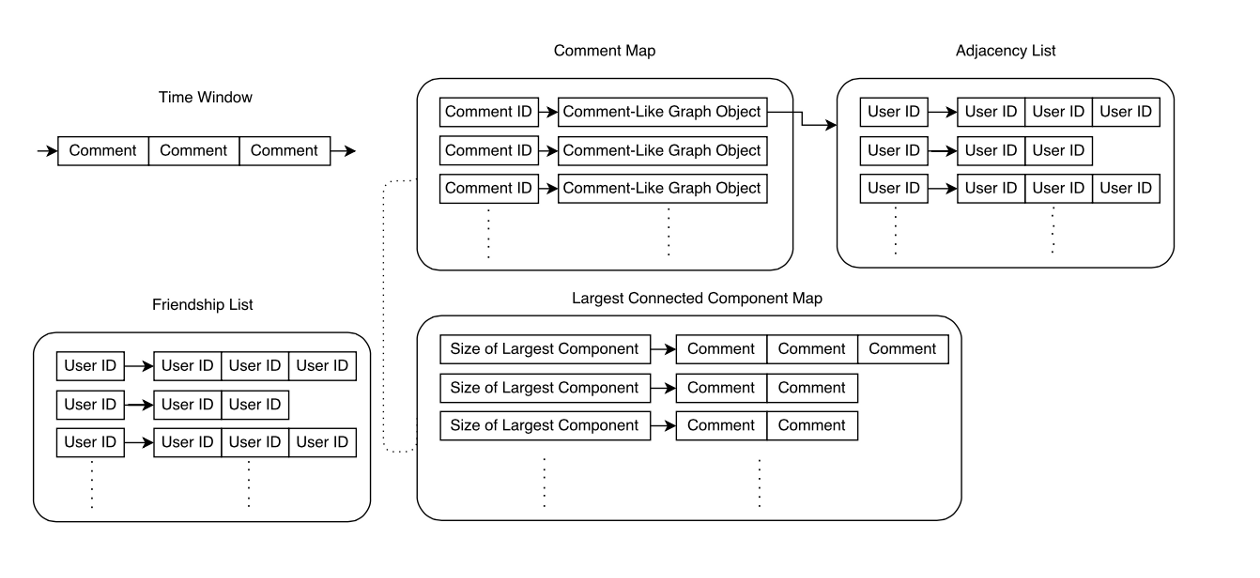
The main objective of the algorithm is to compute the top three posts by only processing a small percentage of active posts.



Query 1: UML

* org.wso2.siddhi.debs2016.extensions.Query.Run is the main entry point to the application and this is where the two queries start.
* The solution is based on the WSO2 CEP and it was implemented as an Siddhi extension. This specific extension is based on org.wso2.siddhi.debs2016.extensions.StreamFunctionProcessor. org.wso2.siddhi.debs2016.extensions.RankerQuery1 extends from this class.
* The org.wso2.siddhi.debs2016.comment.PostStore contains the post-map and post-score map and this class has methods which allow the client code to access post information and register new posts.
* The final top three posts are obtained by processing a hash-map which is based on a comparator (org.wso2.siddhi.debs2016.post.PostComparator)
* org.wso2.siddhi.debs2016.DataLoaderThread1 is responsible for reading the event streams from the text file.
* org.wso2.siddhi.debs2016.OrderedEventSenderThreadQ1 is responsible for the ordering events.

Query 2: Data Structure



Query 2: Data Structure[3]

* Comment Map (hashMap): mapping between the comment\_id and comment-like-graph (Comment-like-graph is adjacency list of users who have liked the comment . For each vertex i, it stores an array of the vertices adjacent to it)
* Largest connected component map (sorted MultiMap): mapping between the size of largest connected component and the comment
* Friendship List: graph of friendships represented using an adjacency list
* Time window of length d

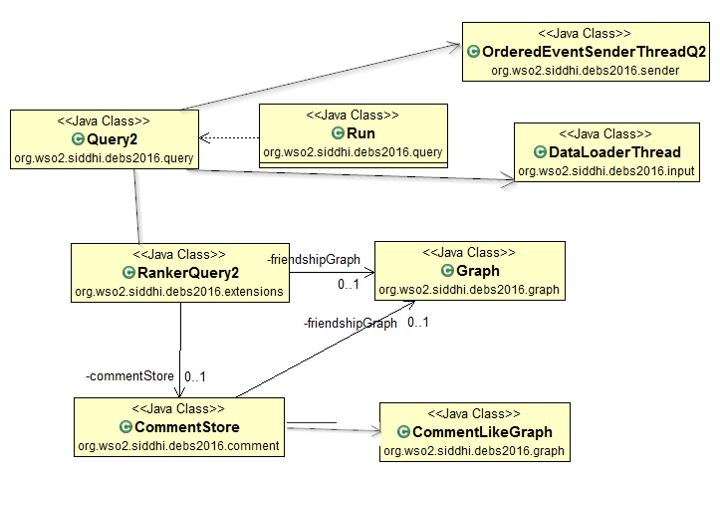
Query 2: Algorithm

When a new friendship event arrives, it is registered in the friendship list. This new friendship can have an effect on largest k communities. For example, a new friendship may result in two disconnected components in a comment-like-graph (refer to the previous section for the definition) to merge forming a single component. When this happens we update the largest connected component map.

When a comment event arrives, it is added to the time window as well as to the comment map.

When a “like” event arrives it registered in the related comment’s comment- like-graph object. The relevant entry in the largest connected component map is then updated with the updated size of the largest connected component.

After processing each event, a check is done on the largest connected component map to see if the order of the top k entries have changed and if so, an output is generated.



Query 2: UML

* Query 2 is also based on the org.wso2.siddhi.debs2016.extensions.StreamFunctionProcessor. org.wso2.siddhi.debs2016.extensions.RankerQuery2 extends from the StreamFunctionProcessor.
* The org.wso2.siddhi.debs2016.comment.CommentStore contains comment map and (mapping between comments and comment-like-graphs) and the time window. The class has methods which provide access to comment information (get top-k comments) and methods which allows you to register a new comment, register a new like and register a new friendship and update comment time-window.
* org.wso2.siddhi.debs2016.DataLoaderThread2 is responsible for reading the event streams from the text file.
* org.wso2.siddhi.debs2016.OrderedEventSenderThreadQ2 is responsible for ordering the events.

**Bibliography**

1. <http://www.debs2016.org/>
2. Kammoun, Abderrahmen et al. “High performance top-k processing of non-linear windows over data streams.” DEBS (2016).
3. Jayasinghe, Malith et al. “Continuous analytics on graph data streams using WSO2 complex event processor.” DEBS (2016).
4. Mayer, Ruben et al. “GraphCEP: real-time data analytics using parallel complex event and graph processing.” DEBS (2016).
5. Udomlamlert, Kamalas et al. “Monitoring top-k on real-time dynamic social-network graphs.” DEBS (2016).
6. Martin, Andre et al. “Real-time social network graph analysis using StreamMine3G.”DEBS (2016).
7. <https://github.com/KRAUT2016/DEBS2016GrandChallenge>
8. <http://wso2.com/products/complex-event-processor/>
9. <https://streammine3g.inf.tu-dresden.de/trac>
10. [https://medium.com/@malith.jayasinghe/debs-grand-challenge-2016-continuous-analytics-on-graph-data-streams-using-wso2-complex-event-43a1c27ec81d#.a5ron0xxq](https://medium.com/@malith.jayasinghe/debs-grand-challenge-2016-continuous-analytics-on-graph-data-streams-using-wso2-complex-event-43a1c27ec81d%23.a5ron0xxq)