Background and Introduction:
I planned to contribute to the open source community by implementing two enhancement features in a real time data processing framework called Apache Storm[1]. These features had been identified and raised by the Storm developer community and are detailed below:

- **STORM – 17: Making spout their own ackers[2]**
  A Storm topology has a set of special ‘acker’ tasks that tracks the tupletree generated for every spout tuple. The current implementation requires the number of ackers to be specified in topology configuration using Config.TOPOLOGY_ACKERS. Internally, spout sends ack/init message to the acker while it responds by sending the ack/fail message to the spout. Additionally, with the current implementation the birthday problem takes effect and occasionally two different spout tuples might have same id, causing them both to timeout and fail.

  The system relies on the fact that a bitwise-XOR applied to any list where every number appears twice (or an even number of times) always produces 0. In this way the acker can know when the tuple has been completed. A mistake could be made if the random 64-bit number causes the XOR to produce 0 before the tree is completed. I am trying to accommodate this logic within the spout.

- **STORM – 69: Adding UI visualization to Storm[3]**
  One of the feature that Storm community is looking forward to is the UI visualizations that would provide a visual indicator of how topologies are behaving. I plan to represent the topology and its components as a force directed graph where the size of the edges between nodes would indicate the amount of throughput passing between them.

  As existing visualization tools have certain limitations I’d utilize the D3.js library in my solution as it supports a vast array of visualization types. Since the library sits right between data and graphics, it is located in the ideal place for data visualization.

  A sample visualization is given below without any data.
Analysis:

- **STORM – 17** would make the Apache Storm project in the following ways:
  - The user would not have to specify the number of ackers anymore as each spout would take care of it’s own generated tuple and tupletree.
  - Storm could get rid of the birthday problem that may arise in the current implementation with a vanishingly small probability of guarantee a more accurate tuple tracking as spouts would be generating their own tuple id and could guarantee their uniqueness by checking if they existed.
  - This would also increase the throughput of Storm as the need for sending messages back and forth between spout and acker would be completely eliminated.

- **STORM – 69** would provide a quick UI on the makeup of the topologies. A force directed graph with indicators marking topologies, their spouts and bolts, presenting throughput and load would ease the life of Storm users. Additionally, it is far too easier than to look at graphs and statistics presented, or than to browse through the code making it invaluable even for Storm developers.
Implementation Details:

- **STORM – 17:**
  I implemented the create, update (as shown below), cleanup functionalities of acker daemon and implemented it in a spout daemon
  
  ```clojure
  (defn updateack [currentry val]
    (let [old (get currentry :val 0)]
      (assoc currentry :val (bitxor old val))
    )
  )
  ```
  
  The spout should then perform the basic functionalities as envisioned:
  - generate and assign a unique tuple id to each tuple it (or its successor bolt) generates.
  - XOR with pre-computed checksum with the tuple id
  - XOR again with the tuple id upon receiving the successfully completing event for the particular tuple or replays the message from queue upon failure event.
  - check if the checksum is 0. If so, send ack message to the tuple generating queue.

- **STORM – 69:**
  I implemented a class to extract the topology information from Storm in java.
  ```java
  backtype.storm.generated.ClusterSummary cs = new
  backtype.storm.generated.ClusterSummary();
  int i = cs.get_topologies_size();
  ```
  Depending on the size of the list of topologies, I obtained the various attributes of the topologies using the following code:
  ```java
  cs.get_topologies().get(i).get_name();
  cs.get_topologies().get(i).get_id();
  cs.get_topologies().get(i).get_num_tasks();
  cs.get_topologies().get(i).get_num_workers();
  cs.get_topologies().get(i).getuptime_secs();
  cs.get_topologies().get(i).get_status();
  ```

Challenges:

Inapt analysis and effort estimation was a major setback from my side, while establishing the scope and setting timelines of the project.

As Storm - 17 is a major design change concerning the underlying architecture of Apache Storm and the primary component that guarantees message processing, I observed modifying even the slightest part of implementation resulted in breaking many dependencies that it has from other components. Also, as Storm is a relatively new project that is still in incubating stage with Apache, it has very little documentation available with a relatively active, but small developer community.

For Storm – 69, I tried implementing this as a component in Java rather than a plugin to Storm. While the current UI of Storm is programmed entirely in Clojure, with all the required
relevant data being already acquired appropriately I realized the impossibility of implementing data visualization in real time in Java. Also, for real systems with 1000s of topologies and their components, condensing topologies in an intuitive manner within a force directed graph proved to be a major hinderance. Lastly, gaining a fair understanding of Clojure was very time consuming, only after which one would be able to take a peek at the underlying architecture supporting Storm.

Results and Discussion:
For Storm - 17, the code modifications could not work within the specified timeline and would take a lot more efforts in order to implement the intended functionality. On the other hand, for Storm - 69, trying to implement the UI visualization component in java proved to be futile. Having gained a fairly good understanding of the architecture and the various relevant components of Storm, and being now familiar with the requirements and the apt approach I will continue working on both of these enhancements in my own time, outside the scope of the project and course.

References: