System infrastructure and performance analysis of anomaly detection application (IndyCar)

Rohit Bapat
rbapat@iu.edu
Indiana University, Bloomington

Amit Makashir
abmakash@iu.edu
Indiana University, Bloomington

Sahil Tyagi
styagi@iu.edu
Indiana University, Bloomington

1 INTRODUCTION

IndyCar is one of the top-level car racing series in America. During the period of each race, different kind of events happen, including pit stops, crashes, mechanical breakdown, drivers ranking changes, and more. With the timing and scoring log data, the task is to detect events interested which can be categorized as the anomaly detection task.

The entire project is mainly divided into 3 phases.

- Event detection and prediction of race data.
- Performance analysis of anomaly detection application.
- Alternatives for message brokers and introducing a NoSQL based data persistence layer.

We analyzed the performance of the anomaly detection application. The data we used for this analysis was stored in a log file. We simulated the real-time conditions to analysis the behavior of the Storm application.

2 RELATED WORK

To get an insight into the Apache Storm structure we read "Real-time Anomaly Detection Over VMware Performance Data Using Storm" [4]. This paper is about the architecture of Apache Storm and its advantages over more recognized implementations like Hadoop or Spark. To learn more about the anomaly detection process we read about the use of Apache Storm to detect anomalies of VMWare node setup. The paper tells about the use of unsupervised clustering algorithms like K-means and Incremental clustering. This article especially made us familiar with terms like Spouts, Bolts, Supervisor, ZooKeeper etc. In the paper "HPC-ABDS High Performance Computing Enhanced Apache Big Data Stack" [3] we read about the "Apache Storm processing Data for the Internet of things". This structure was very similar to the current IndyCar Pipeline model.

3 METHODS

We are using Apache Storm to stream in data coming in at high speeds from the race cars. To enable Apache Storm we had to install some dependencies first. Following are the dependencies to have Storm up and running:

3.1 Zookeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.Storm uses Zookeeper for coordinating the cluster nodes.[2]

3.2 Nimbus

Nimbus is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures.[2]

3.3 Supervisor

Each worker node runs a daemon called the "Supervisor". The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it. Each worker process executes a subset of a topology; a running topology consists of many worker processes spread across many machines.[2]

3.4 Apache ActiveMQ Apollo

Apache ActiveMQ Apollo will be used as a message passing (MQTT) broker. The data from the race cars

will be published onto a topic in this broker and the corresponding spouts are subscribed to this topic.

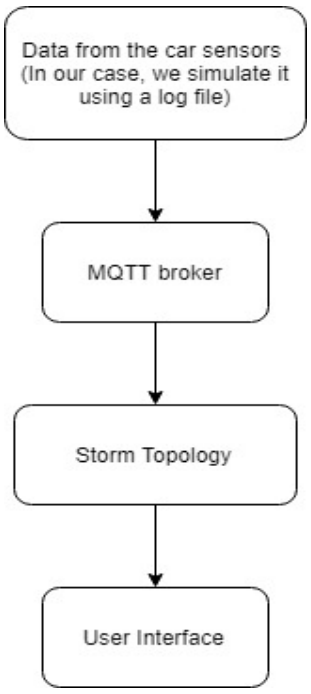

Figure 1: Data flow for this application

3.5 HTM

Hierarchical Temporal Memory (HTM) is a biologically inspired machine intelligence technology that mimics the architecture and processes of the neocortex. It offers certain advantages over traditional approaches like:

- No separate training or manual intervention required.
- Individual models are automatically built for any number of inputs.
- Continuous online learning allows the application to learn new patterns without manual intervention.
- The neocortex uses the same learning principles for vision, hearing, touch, movement, language and planning. HTM algorithms are neocortical algorithms and can similarly be used in many different applications.

These advantages make HTM suitable for detecting anomalies in our streaming application. [1]

4 ARCHITECTURE AND IMPLEMENTATION

4.1 Setup Configurations

The Juliet node has the following configurations:

(1) CPUs : 48
(2) Threads per core : 2
(3) Cores per socket: 12
(4) Model : $Intel(R)Xeon(R)CPUE5{-}2670v3@2.30GHz$

4.2 Experiment Settings

The current IndyCar infrastructure is being implemented for 33 cars which simulates the actual Indy car race. Initially, under the guidance of our mentor, we were able to simulate the sensor data and anomaly detection for a single car (Car #20). We executed this experiment to better understand the infrastructure in terms of storm topology and related deployments. We then extended this analysis to 2 cars, 4 cars and 8 cars. The experiment steps can be explained as follows.

The minimum number of nodes required to simulate "n" cars can be given as follows:
no. of nodes = roundup(no. of tasks/ 24)

There are 5 tasks per car (1 spout, 3 metric bolts and 1 sink bolt). Therefore, the no. of tasks would be "5n". We have 12 cores per socket and 2 sockets per node. Therefore, we have 24 cores per node.

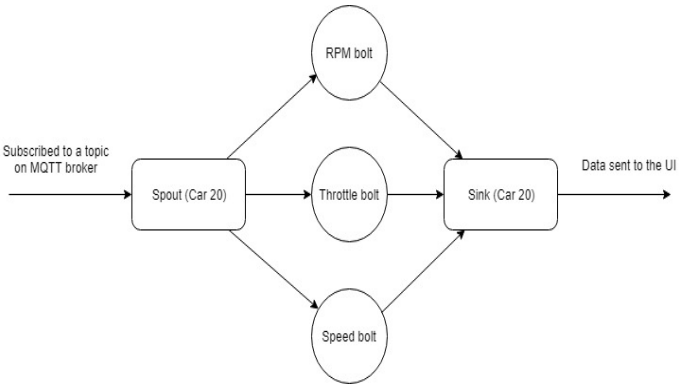

Figure 2: Storm Topology for one car

4.2.1 Log File. The single log file contains the sensor data which needs to be delivered to the Apache Storm

which in turn streamlines it to the Hierarchical Temporal Model (HTM) module. This log data is published on to the next component which is a publisher subscriber broker.

4.2.2 Apache MQTT. After installing the Apache MQTT we created a broker, named as 'mybroker'. We made some configuration changes in the apollo.xml files to point the broker to our IP address. This broker will have logs published to it while being subscribed by next component.

4.2.3 Apache Storm. We ran the Zookeeper server which is a pre-requisite for Apache Storm. We used the Apache Storm 1.0.4 to carry out the experiment for single car. We then started the nimbus, supervisors and UI for Storm.

4.2.4 Creating Tunnel for Storm UI. We created an ssh tunnel to view the Storm UI through a PuTTY Client and check running state of Apache Storm UI.

4.2.5 Storm Topology. Based on the source code shared by our mentor, we made changes to Sink and Bolt classes which can be used by the Storm topology file (YAML file). The YAML file describes a structure of mapping the different bolts, sinks and spouts. It contains the constructor calls with specific car numbers. Because of the recent modification the sink number is also mapped from Bolt20 to Sink 20. We give the YAML file and the jar file compiled by maven.

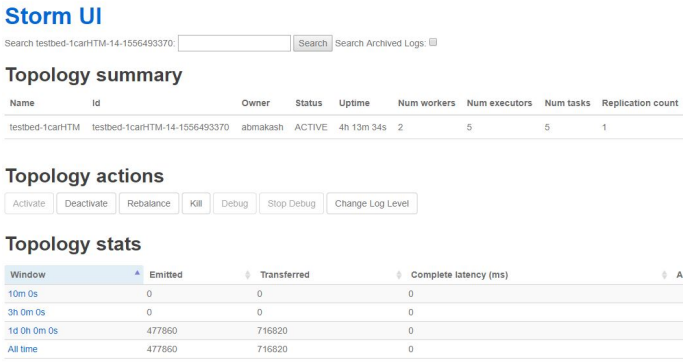

Figure 3: Storm Topology for one car

4.2.6 Publishing the Logs. We used the logs for Car #20 and started the RealPublisher class to publish the log data to mybroker (MQTT) queue. These logs are subscribed by the Spout20 which inturn sends it to Bolt20. We can see the different bolts for metrics - Speed, Throttle and RPM with statistics for emitted and transmitted entries.

The above components are implemented in a sequence. Once receiving the messages from the queues the tuples consisting of the car number and three metrics are sent to the bolts. These bolts make asynchronous calls to HTM module to get an Anomaly score which are dumped onto a sink. Here each sink is separate for each car.

4.3 2 Node Cluster Experiment Setting

To set-up a cluster of 2 nodes we made the following changes in our Zookeeper config:

- Assign an id to each node by creating a file called "id" in data directory
- Register this id in zoo.cfg and list the corresponding IP address.
- Restart Zookeeper server and check status (one leader and one follower)

Following are the changes in Storm yaml:

- List the IP addresses of both the nodes in Zookeeper servers
- Specify the nimbus host and seeds on both the nodes
- Restart all Storm services and check the number of supervisors on the UI

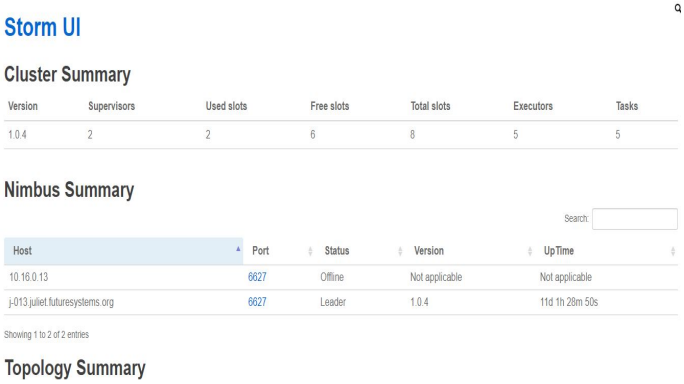

Figure 4: Storm UI showing 2 supervisors

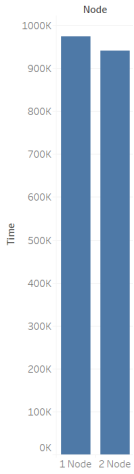

Figure 5: Processing Times for Node Comparison on Car#20

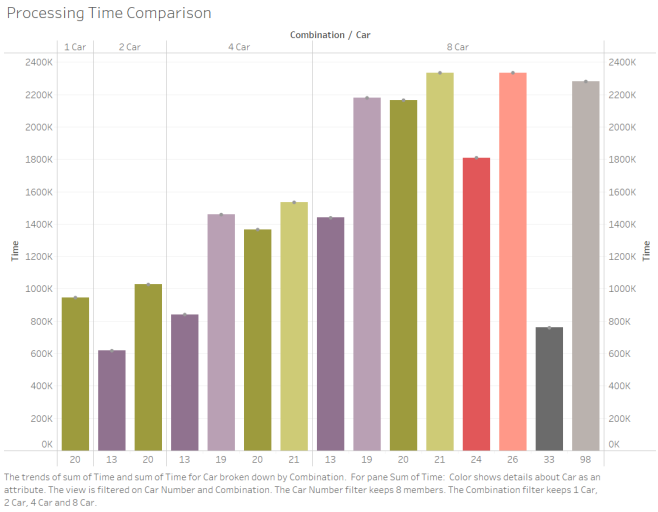

Figure 6: Processing Time Comparison

Figure 7: Speed-Anomaly Score Plot for Car#19

5 EXPERIMENTS

5.1 Word Count Program Comparison

"Word count" is considered to be the "Hello world" equivalent in Apache Storm. The aim here is to build a storm topology that counts the number of words in the sentences that are being streamed.

We developed a simple storm application that streamed sentences from a corpus into a spout called "SentenceSpout". The "SentenceSpout" was followed by a "SplitSentenceBolt" that was responsible for splitting the sentences into corresponding words. These words were then passed to "WordCountBolt" that incremented the counter of the words it sees. Finally, we had a "ReportBolt" that was used for aggregating (or reduce step) to combine all words and their corresponding counts and displayed the results. We deployed this topology on a single node.

5.2 Node Comparison

As shown in Figure 5 We compared the time required to parse a single car(Car#20) file on a single node as well as a 2 node cluster.

5.3 Processing Time Comparison

We used an incremental approach to analyze the processing times for the 8 cars in the race. Based on the two node cluster we experimented on the 4 combinations - 1 Car, 2 Cars, 4 Cars, 8 Car. As seen from Figure 6, we can see that the general trend for the same car in each combination was increasing. The number of records for processing remained the same. The cars used for this analysis were 13, 19, 20, 21, 24, 26, 33, 98.

5.4 Metric Based : Speed Analysis

In the above visualization we compared the anomaly scores with the 8 Car Data. We plotted the Speed metric with respect to time of the day. The X axis starts generally at 3:50 pm and ends at approximately 7:30 pm. Car#19 shows the variations in the Speed Metric and the anomalies detected by HTM module. We have designed a Tableau Dashboard where the Anomaly Score

Figure 8: RPM-Anomaly Score Plot for Car#19

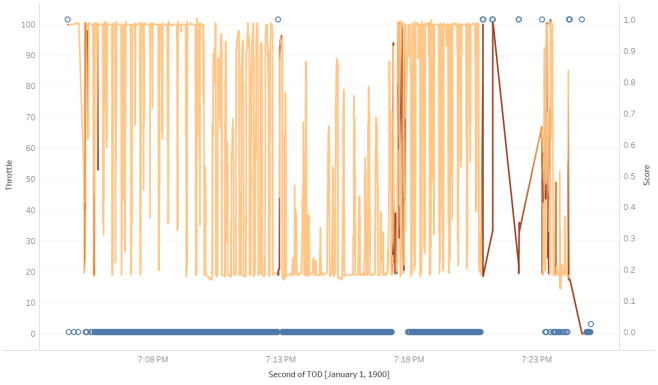

Figure 9: Throttle-Anomaly Score Plot for Car#19

filter is configurable to analyze the behavior of metric at each time of the day at the seconds interval.

5.5 Metric Based : RPM Analysis

In the above visualization we compared the anomaly scores with 8 car data. The RPM metric is described as 'engineSpeed' in the IPBroadcast Log File. From the above figure we can see the RPM plots for the same car number NUMBER

5.6 Metric Based : Throttle Analysis

In the Figure 9 we plotted the Throttle values(Throttle %) fetched from IPBroadcast logs against a time of the day X axis. The figure shows an example for Car#19. The circles denote anomaly score points. Most of the points are 0 level anomaly points. We used a gradient to show change in anomalies based on each throttle level. Our general understanding of this experiment was that many anomalies were seen at the near end of

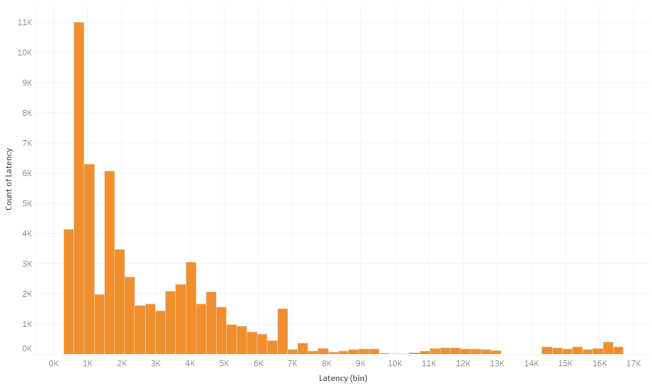

Figure 10: Throttle-Log Latency Distribution for Car#20 in 2 Cars

the race. The same plots can be seen for each car in the combination on the Tableau Dashboard.

5.7 Latency Analysis

The Speed, RPM and Throttle metrics and their anomalies were universal irrespective of the 1, 2, 4 or 8 car combinations. We wanted to experiment the effect of latency for all the 4 combinations. For this experiment we plotted a Histogram of latency for all the combinations. The Figures show the distribution of latency frequency for each combination. It is evident from Figures that the latency bins increased when more cars were added for processing on the 2 Node cluster. We have the latency distribution of each metric over all the 4 combinations. For visualization purposes, to show the effect on Latency for all the combinations we have shown Car#20 Latency distribution which occurs in each of the combination at each Metric Filter(Speed, Throttle, RPM).

5.7.1 Throttle Metric Latency. We can see from the Figure 10 and 12 the latency values increase when 7 other cars are running on the 2 Node Cluster. The further description of Latency for Throttle Metric is as follows:

	2 Car	4 Car	8 Car
min	3	3	4
max	16609	83420	446405
mean	2315	211	1665
median	1087	30	19

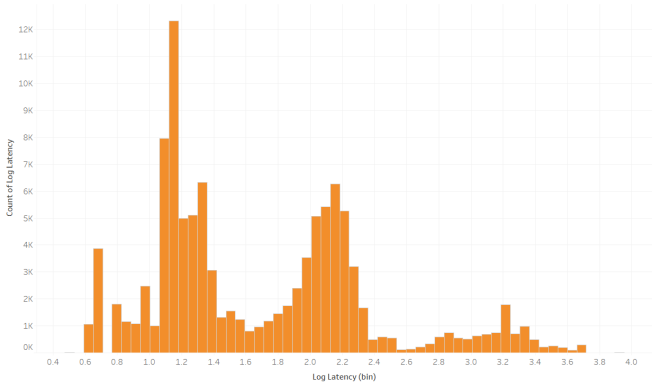

Figure 11: Throttle-Log Latency Distribution for Car#20 in 4 Cars

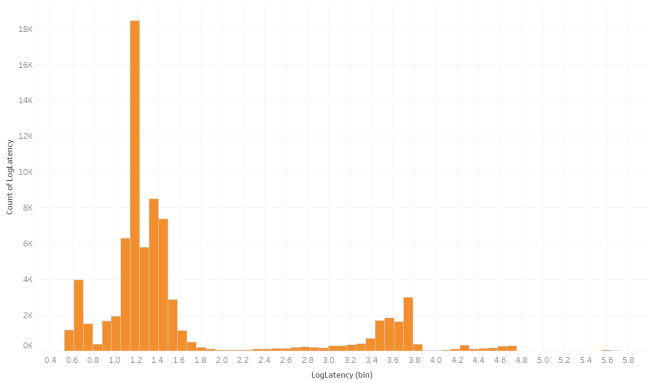

Figure 12: Throttle-Log Latency Distribution for Car#20 in 8 Cars

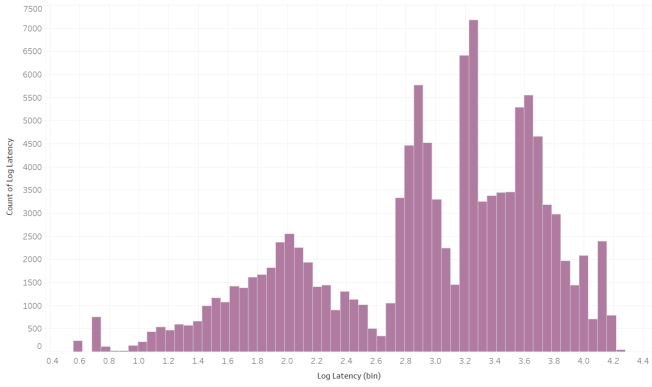

Figure 13: Speed-Log Latency Distribution for Car#20 in 2 Cars

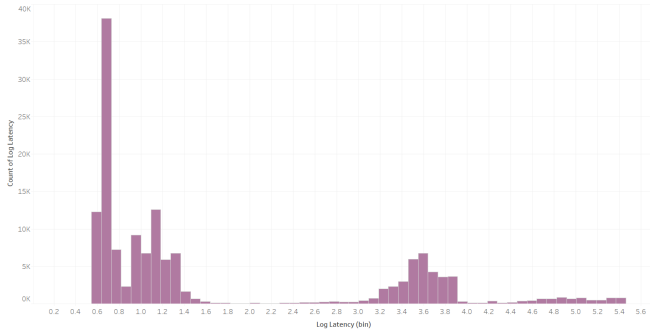

Figure 14: Speed-Log Latency Distribution for Car#20 in 4 Cars

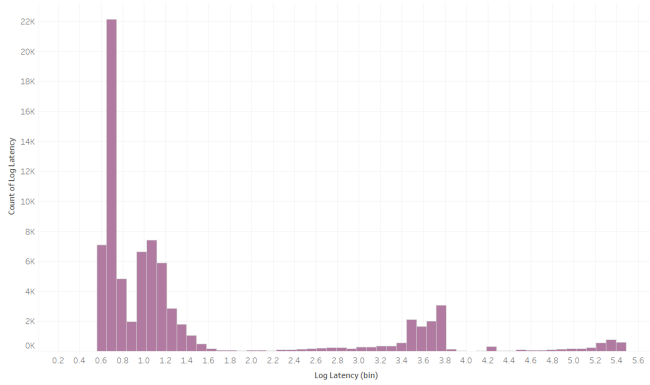

Figure 15: Speed-Log Latency Distribution for Car#20 in 8 Cars

5.7.2 Speed Metric Latency. The trend seen for the Throttle Metric continues for Speed Metric. We can see from Figures 13 , 14 and 15 the latency frequency and the over bins of Latency increase as we move from 2 car to 8 car combinations. The further description of Latency for Speed Metric is as follows:

	2 Car	4 Car	8 Car
min	4	2	2
max	16505	277100	286105
mean	2446.812	6056	7415.197
median	1400	30	10

5.7.3 RPM Metric Latency. The description of Latency for RPM Metric is as follows:

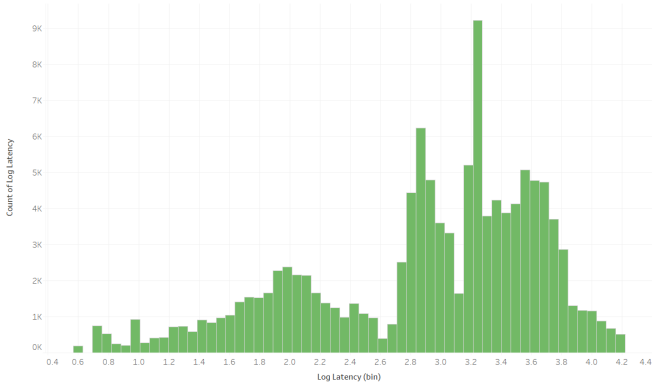

Figure 16: RPM-Log Latency Distribution for Car#20 in 2 Cars

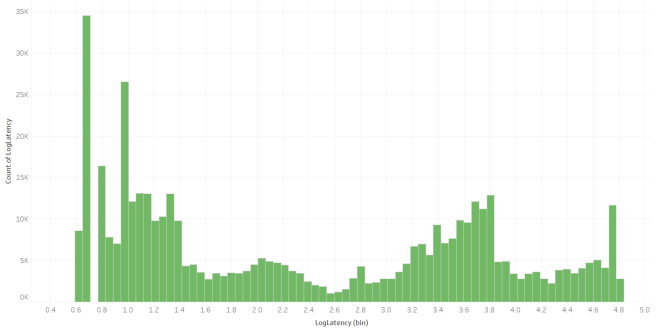

Figure 17: RPM-Log Latency Distribution for Car#20 in 4 Cars

	2 Car	4 Car	8 Car
min	4	3	2
max	16500	8142	357881
mean	2112	220	11457
median	1182	16	12

6 CHALLENGES

Some of the challenges we faced while working on this project are as follows:

(1) The sink files generated after every simulation had latency and anomaly score corresponding to each metric (vehicle, engine and throttle) for each record in the sink file. However, the "Time of the day" value for each of these metrics was the same. We wanted to maintain the sequence of anomalies generated for a particular Time of the Day metric at seconds levels. We used the counter in the sink file to join these records and get the corresponding Anomaly scores and Latency for for each metric.

(2) While Parsing the IPBroadcast Logs which had the sensor results, we came across obstacles like file parsing and the broken pipe separator which is Operating System specific delimiter. The telemetry document helped us to parse only the $P Logs which were for metric based values. We combined the parsed files with the sink files to get a metric based car independent scores and latency.

(3) After forming a 2 node cluster, we found that the Bolt, sink files are dumped in the local file location in our case. However, being a 2 Node cluster, the Nimbus divides the File streaming based on the car level. Hence, for 2 car combination only 1 car sink was created on the leader node, while the other file was lost as leader node did not have permission to write in the follower node file hierarchy. We resolved this problem with creation on similar directories on all nodes and all permissions to the leader node. This generated the desired output.

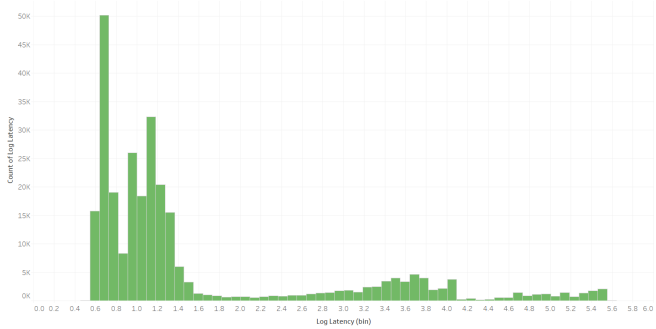

Figure 18: RPM-Log Latency Distribution for Car#20 in 8 Cars

7 OBSERVATIONS

7.1 Missing Data points after High Latency Event

Based on parsing of the IPBroadcast Logs we completely removed the entries with invlid timestamps which were overshooted by the HTM module as anomalies. This made the data discrete for that time frame till we got the next valid anomaly score. For Car#24 we found that this data was not present for Time range of 4:50 pm to 5:10. This is the same time where Car#13 crashed. The data to study the relation between the Car#24 Throttle metric

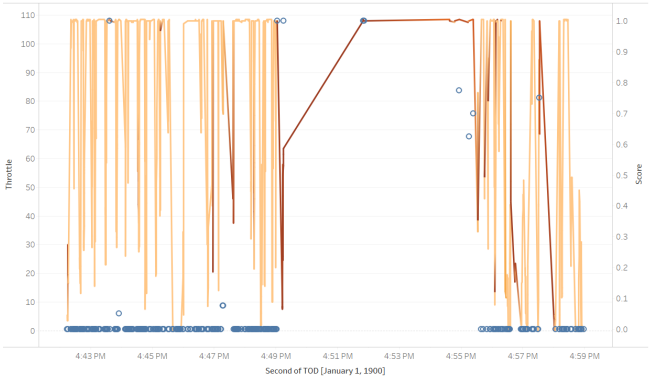

Figure 19: Throttle Data Observation for Car#24

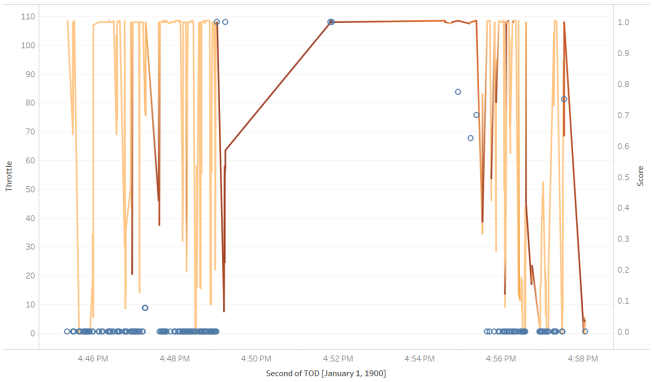

Figure 20: Throttle Data Observation for Car 24 for Crash Event

and the crash event was not available. This missing data can be seen in Figure 19

7.2 Analysis for Car 13 Crash

We observed that the Car 13 did not have data after 5:10pm for Speed Metric, 4:57 for Throttle Metric and 4:53pm for RPM metric. During Peer evaluation we understood that Car 13 crashed during the race. This explained the limited data. We checked to see if we got any after effects of this crash on other cars. Such analysis was correlated with the Yellow flag instance in the F1 race. One analyzing other car we found that: In throttle metric for Car 24 there was a dip in the Throttle at many points after 4:55pm as shown in Figure 20 In the Speed metric for Car 20 there was a dip in Speed at 4:56pm which is in similar time frame as Car 13 crash as shown in Figure 21. In the RPM metric for

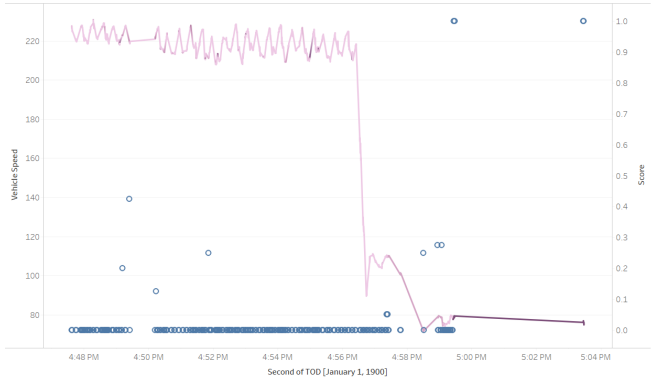

Figure 21: Speed Data Observation for Car 20 for Crash Event

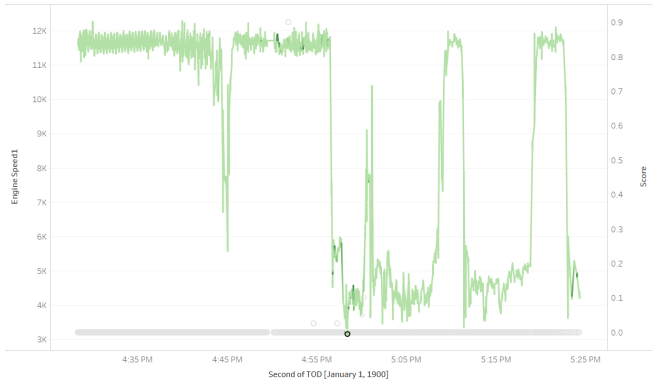

Figure 22: Engine Data Observation for Car 24 for Crash Event

Car 20 the there was a dip in the RPM at 4:58pm as seen in the Figure 22

8 CONCLUSION

- Through this project we learnt the Storm Configurations and Topology architecture.
- We understood the importance of streaming analytics with a tool like Apache Storm.
- After doing Time comparisons for 1 and 2 node clusters we understood that performance speeds up but never doubles given the communication and parallelization overhead.
- Through experiment observations we can infer that an anomaly detected on certain car, affects the latency of the system and thus also affects the metrics of other cars in the race.

- The overshooting of latency for a car can lead to data loss by dropping certain metric values.
- The last metrics values at the end of the race are always generating higher anomaly scores.

9 ACKNOWLEDGEMENTS

We are grateful to Prof. Judy Qui for providing us with the opportunity to work on such an exciting project. We appreciate the effort taken for providing us with the data and other resources for this project. We would also like to thank our mentor Sahil Tyagi and Associate Instructor Selahattin Akkas for guiding us in this process.

REFERENCES

[1] 2016. HTM. https://numenta.com/assets/pdf/whitepapers/Numenta%20White%20Paper%20-%20Science%20of%20Anomaly%20Detection.pdf

[2] 2019. Apache Storm Descriptions. http://storm.apache.org

[3] Supun Kamburugamuve Shantenu Jha Andre Luckow Geoffrey C. Fox, Judy Qiu. 2015. HPC-ABDS High Performance Computing Enhanced Apache Big Data Stack. Retrieved February 10 , 2019 from https://ieeexplore.ieee.org/abstract/document/7152592

[4] Bhavani Thuraisingham Mohiuddin Solaimani, Latifur Khan. 2014. Real-time Anomaly Detection Over VMware Performance Data Using Storm. Retrieved February 10 , 2019 from http://doi.ieeecomputersociety.org/10.1109/IRI.2014.7051925