Speedly: Aggregating Traffic Analysis with Stream Processing

Sallar Farokhi
University of California, Santa Cruz

Jason Mack
University of California, Santa Cruz

Ankur Ahir
University of California, Santa Cruz

1 Abstract

The issue of traffic congestion has long persisted in urban areas, dense populations, and poorly designed road networks. From these limitations, gridlock and carbon pollution have become apparent consequences from the extended periods of time cars spend on the road, harming everyone involved. Unfortunately, devising a solution for the rigid nature of traffic patterns is not so simple. One such reason is the static nature of traffic lights, relying solely on fixed intervals to switch between red, yellow, and green. As a result, present-day traffic applications have focused on optimizing shortest-path solutions as a convenient response. But as we invest more and more resources into existing applications such as Google Maps, we back ourselves into a corner. As geolocation services front the cost of supporting greater workloads over a wider area, traffic light infrastructure remains untouched and continues to serve as the overall bottleneck for cars on the road.

In this paper, we present Speedly, the proof-of-concept for a real-time traffic analysis cluster that collects, aggregates, and displays velocity and road density. Speedly is built upon a Flink cluster that processes incoming geolocation data from several cars at once, and aggregates their compute demands to effectively return actionable data for users, developers, and smart traffic devices. For the purposes of our model, we uploaded the location and velocity of any currently streaming vehicles into a postgis database, and display the real-time data with Grafana, an open-source analytics application.

2 Motivation

As urban and suburban traffic increases, we find that it becomes more and more difficult to prevent congestion in densely populated areas, let alone guarantee a certain speed of traffic. Government infrastructure and traffic rules are sparely changed, and so the onus of improving traffic conditions falls on the driver. To this end, many traffic solutions today offer easy visibility for drivers to navigate to their destination and be cognizant of any road incidents. Unfortunately, real-time data analysis engines face significant challenges satisfying Service Level Objectives(SLO) with minimal latency. Traditional methods of monitoring traffic conditions and enforcing speed limits have often relied on fixed infrastructure, such as speed cameras and road sensors, as vantage points for traffic data. As a result, there have been limitations to the scale at which speed of traffic and traffic congestion can be observed by local government and corporations. In order to increase the precision and scale, more cameras would have to be installed at interval locations on the roads, generating exponential costs and bulk data at the expense of the observer.

While it's impractical for the government to enact an observation framework for regional traffic data analysis, many corporations already have the means to track the roads and make informed claims about the state of traffic. Figure 1 displays Google Map's traffic feature, clearly displaying the bottlenecks where traffic is the worst. With constant access to the geolocation data of millions of mobile users, navigation apps such as Google Maps and Apple Maps allow users to find the quickest route anywhere, but despite these methods, the bulk of information regarding the speed of traffic still remains virtually inaccessible to the average driver. This often leaves users to question or even deviate from the correct path at times. Existing services now have a strong emphasis on the optimization of path construction, as drivers in an unfamiliar environment are more interested in that over the rate of traffic.

Yet, this poses an issue for another demographic of drivers that are decidedly more common: the population of users who don't need a path to their destination. Despite the expansive coverage of popular geolocation services, one must consider the possibility of a traffic analysis service which provides real-time updates on traffic speed, congestion, and speed limits without requiring drivers to follow a predefined route. This service could cater to a growing group of users who are familiar with their destination, but still benefit from traffic insights to adjust their driving decisions in real-time. By focusing on traffic flow and speed limit data, rather than navigation, this solution would address a gap left by traditional mapping ser-

vices, offering an enhanced driving experience for those who prioritize staying informed about current road conditions.

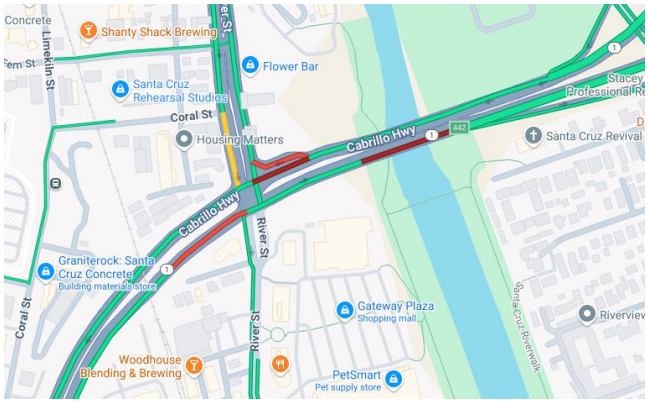

Figure 1: Google Maps traffic display

3 Question

As it currently stands, the majority of location-directed services offer little information regarding the velocity of traffic. There are a number of reasons why this might be the case. As mentioned before, there is a lack of a dynamic outlet by which vehicles on the road can be observed without the expansive installation of road cameras. Another possibility is the issue of anonymity, and more specifically, the willingness of a user to contribute their driving information to a third-party, with the worries of being held liable for speeding.

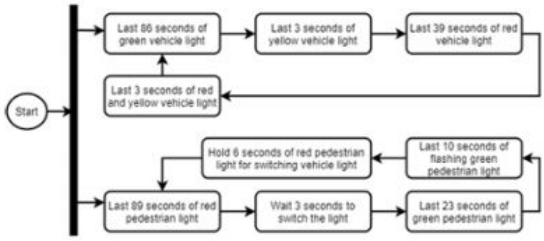

Figure 2: Flows of signal switching in fixed-cycle traffic lights (FCTL).

One such issue that drivers and developers share is the inflexible design of traffic lights, mainly their static intervals. Practically every stoplight in the United States operates on a fixed basis, similar to Figure 2, but this is a paltry effort to manage traffic of varying densities, especially towards the farther ends of the spectrum. [4] demonstrates an attempt from Hong Kong to improve the overall performance of urban traffic infrastructure by installing cameras and smart lights. Figures 2 and 3 show two proposed models that work interchangeably, based on the real-time congestion of the road, to

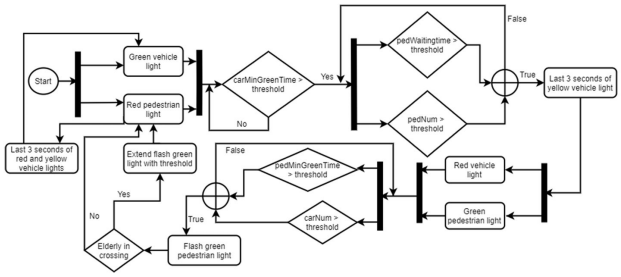

Figure 3: Flows of signal switching in intelligent traffic light system (ITLS).

deliver an optimized output for crosswalks and roads. The first flow graph demonstrates a fixed-cycle model, where every stage of the traffic light is determined based on duration or pedestrian presence. The second model offers a more adaptive logic for the traffic, which allows the intervals to change based on the presence of cars and people. This will be discussed further in 4.

Another issue that concerns the developer is the real-time processing of the streamed traffic data. While traffic-pathing services have the luxury of caching popular and recent routes, applications providing real-time updates of traffic state face latency demands more stringent than route planning. In order to provide accurate results for clients, vantage points must submit constant information to a system, and in the case of varying usage, said system needs resources available to consistently satisfy SLOs.

Popular services such as Google Maps and Apple Maps provide both route planning and traffic status, but like every traffic application, this function is ultimately decoupled from the workings of the traffic lights. Drivers and developers alike are forced to adhere to the fixed intervals of the current traffic model. The question now becomes: can a framework be devised to devote compute power to the sole purpose of traffic aggregation and optimization? We believe that such a solution is possible with Speedly and like systems.

4 Related Work

Previous attempts at this traffic solution have been made, with one notable paper from 1999 that attempts to create a system with nearly the same goal as us. [2] proposes an improvement to existing information systems built upon a real-time traffic simulator. Figure 4 gives a glimpse into the beginning of traffic analysis, with the datapoints presented on the map inspired by recent vehicle data. In the earliest stage, researchers simulated the traffic environment to provide hindsight, but this is not recent enough for our purposes.

At the time, this was a novel contribution to the state of information processing, even if today's society can easily substitute said work for machine learning. Nevertheless, "it

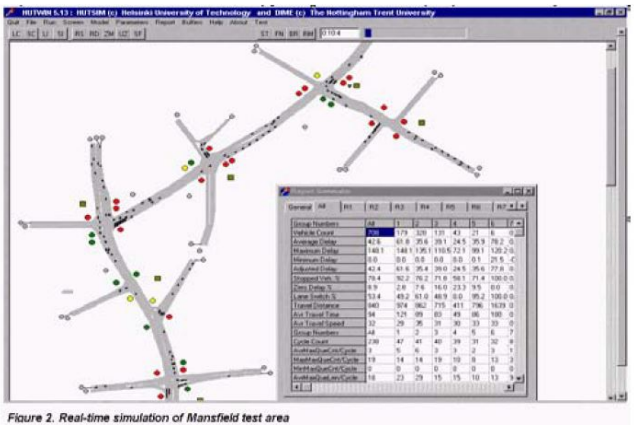

Figure 4: Real-time simulation of Mansfield test area

is believed that many more users and general public could benefit from the results of real-time simulations of urban traffic if these are made available in an accessible form on the World Wide Web (WWW)"[2]. This was a statement made six years after the publication of the Internet and the aim of the paper continues to hold merit, inspiring work like this paper.

Figure 5: ML Model to Detect Pedestrians & Cars

In the more recent realm of research, there has also been work that looks at the relationship between traffic lights and city infrastructure. One such article [4] attempts to bridge the gap between road congestion and metered velocity speeds in road networks through vehicle detection. These cameras were installed intermittently throughout the city, and a machine learning model would work to identify any people or cars in view. As opposed to the fixed architecture mentioned earlier, traffic lights receiving a feed of freshly generated data could adapt to a predetermined environment in real time, and achieve excellent performance even in moments of traffic congestion.

Author	Publication year	Device used	Parameters considered/focus area	Penetration rate
Friesen et al.[41]	2014	Bluetooth, GSM, and Xbee	Traffic count and traffic flow	4.50%
Mathew et al.[44]	2014	Bluetooth and traffic counter	Travel time and reliability	7.11 % and 10.04% (under heavy traffic)
Araghi et al.[46]	2015	Bluetooth and GPS	Reliability and accuracy of Bluetooth (BT) travel time estimation	Not given as a percentage of traffic volume
Remias et al.[43]	2017 (Study done in 2013)	Bluetooth	Travel time, device count, and reliability	7%
Khliefat and Shahnawi[45]	2017	Bluetooth	Optimal location of BT nodes for determining OD matrix	Not applicable
Advani et al.[22]	2020	Wi-Fi sensors and Bluetooth	OD matrix using Wi-Fi and BT devices at various locations	2%
Cevcik and Koçak[42]	2020	Bluetooth and video	Travel time and vehicle classification	3%–7%

Figure 6: Bluetooth penetration rate found during travel time studies by various researchers

This is similar to what present-day Intelligent Transportation Systems (ITS) do to optimize traffic flow in metropolitan areas. [5] is an example of cutting-edge work that looks to train a Latent Factor Analysis Model into better designing the traffic output. Unlike the papers mentioned in 4, an important decision has been made to use fixed infrastructure not as the basis for the traffic infrastructure, but rather as an aid to a machine learning model itself. [5] makes the observation that "Most studies overlook the extreme case where the missing rate of observed traffic data exceeds 90%, causing the existing traffic data imputation models to struggle in achieving satisfactory results". We see in figure 7 that the proposed dataset Z is transformed, based on time-of-day heuristics and real-time streamed traffic data.

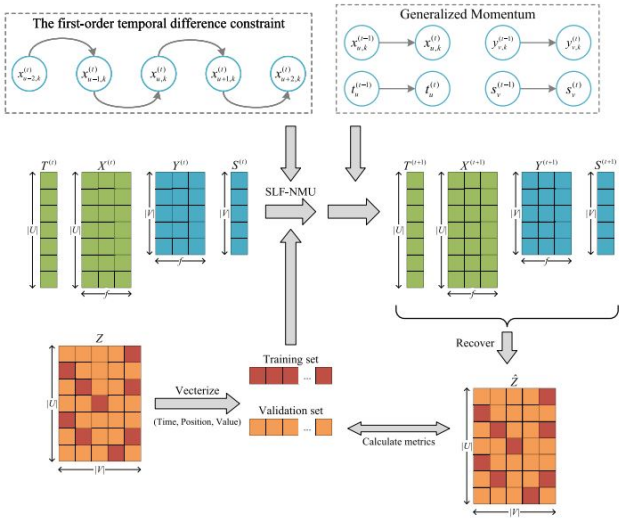

Figure 7: Proposed Latent Factor Analysis Model

Of course, similar work to ours has been published in the past, with the intent to track vehicles directly from the source, and apply the freshly acquired data to optimize traffic patterns. One notable paper [3] attempts to use the increasing presence of Bluetooth technologies onboard vehicles to passively track the movement of cars on the road. Figure 6 demonstrates prior

attempts at Bluetooth tracking, with a somewhat poor rate of penetration. [3] offers improvements over [4] by removing its dependency on localized cameras around traffic choke points. Unfortunately, this bottleneck was merely substituted for another, with the range limitations of Bluetooth also inhibiting application performance. Part of [3]s success was due in face to the dense mapping of India's road networks, and this ultimately circles back to the same limitations posed in [4]: fixed infrastructure will impede the system.

All of the ideas listed above made important strides in their own right, but they don't quite address the issue at hand. Within the first paper, we see the early stages of an information revolution, simulating traffic behavior and publishing it for the world to see. Unfortunately, even a modernized version of this application would lack the recency needed to offer traffic lights the potential benefit of real-time vehicle information. With the other two papers, we see a solution that offers real-time traffic optimization, both adaptive and static, but there are a number of issues with this. The first is a heavy reliance on cameras or routers at points of interest, generating upfront costs and overhead. secondly is the limitation of the devices' coverage, especially as covered areas becomes less and less urban. As the area between traffic lights grow, the information gathered by traffic cameras grows irrelevant, save for moments of apparent road congestion. Despite the appeal of the low-power, open-access Bluetooth protocol, we find that the range of Bluetooth is a significant flaw in the model presented by [3].

In addition, we find that selective participation is another issue for such an experiment. One of the strengths of current navigation apps is their penance to preserve the anonymity of users. Google Maps does not have to reveal anything about the user's location or identity, at least to the public. With the presence of such a solution, most users wouldn't bother to participate in the advancement of traffic analysis, let alone reveal personal information. Figure 8 gives a stark reminder about the limitations of wide-scale census, especially within research.

Ultimately, the architecture proposed by these are bottlenecked by their outdated methods or physical limitations, furthering our beliefs that such a traffic analysis system should be sparse, anonymous, and relatively decoupled from its physical environment.

5 System Design

To satisfy our proof-of-concept, we will need to satisfy the constraints listed in 6. In order to do this, we implemented the Datastream API from Apache Flink in Java [1]. We used the sockets library to supply our input stream, exposed on port 9999. With this, we connected our Flink architecture to a Python script that would simulate a traffic environment in Santa Cruz. This stream takes in four attributes, the id of the vehicle, the latitude, the longitude, and its current speed. We assume clients are able to generate all four data points since most mobile devices have this capability. Later iterations of this model should consider the addition of stateful operations, in order to limit the amount of transmitted vehicle data.

Figure 8: Attempts at Data Collection

```
DataStream<VehicleData> vehicleDataStream = env
.socketTextStream("python-socket", 9999)
.map((MapFunction<String, VehicleData>)
  value -> {
    String[] tokens = value.split(",");
    return new VehicleData(
        Long.parseLong(tokens[0]),
        Double.parseDouble(tokens[1]),
        Double.parseDouble(tokens[2]),
        Double.parseDouble(tokens[3])
    );
});
```

By default, we generate 100 clients but this is interchangeable. From our experiments, we can simulate over 1000 clients without performance hitches in our systems. This is an important detail to note, as infrastructure limitations are also a part of our real-world issue.

To simulate client data streaming in we have a python script that can simulate a variable number of clients driving on preset paths with some randomness in how it follows the path. The script works by creating an array of client objects. The script then repetitively runs the update function on the client objects. This update function updates the location of the client locally, and then every few update calls it sends its location to our flink servers.

We implemented a postgres database using the postgis extension. This allowed us to efficiently query a group of latitude and longitude points along a certain 10 mile region. Under Flink, we connected postgres as our output stream using the JDBC API, and arbitrarily set a batch interval of

200ms with a batch size of 1000. This has two purposes in our system. First, if 200ms has exceeded, Flink will send the batch of data in postgres. In the second case, if over 1000 events happen within the 200ms, Flink will send the batch early to save downtime in the system. There is no thread pool that's occurring within the database itself. From the database's perspective, only one connection is happening. All the complexity of handling multiple clients is performed by Flink itself.

```
vehicleDataStream.addSink(JdbcSink.sink(
    "INSERT INTO traffic_data
      (vehicle_id, latitude, longitude, speed)
       VALUES (?, ?, ?, ?)",
    (statement, vehicle) -> {
        statement.setLong(1,
          vehicle.getVehicleId());
        statement.setDouble(2,
          vehicle.getLatitude());
        statement.setDouble(3,
          vehicle.getLongitude());
        statement.setDouble(4,
          vehicle.getSpeed());
    },
    JdbcExecutionOptions.builder()
            .withBatchIntervalMs(200)
            .withBatchSize(1000)
            .withMaxRetries(5)
            .build(),
    new JdbcConnectionOptions
      .JdbcConnectionOptionsBuilder()
            .withUrl
              ("jdbc:postgresql://db:5432")
            .withDriverName
              ("org.postgresql.Driver")
            .withUsername("postgres")
            .withPassword("postgres")
            .build()

));
```

Finally, we used Grafana to visualize all of these results. Grafana receives data from the Postgres database with the SQL query shown here. All that the query does is requests the most recent location of each user.

```
SELECT DISTINCT ON (vehicle_id)
    vehicle_id as id,
    ST_Y(location::geometry) AS latitude,
    ST_X(location::geometry) AS longitude,
    timestamp
FROM traffic_data
ORDER BY id, timestamp desc
```

To simplify our workflow, we store the configuration on postgres connection in its own folder. Every part of this was written into a docker-compose file, meaning that only a few commands are necessary to run everything listed above.

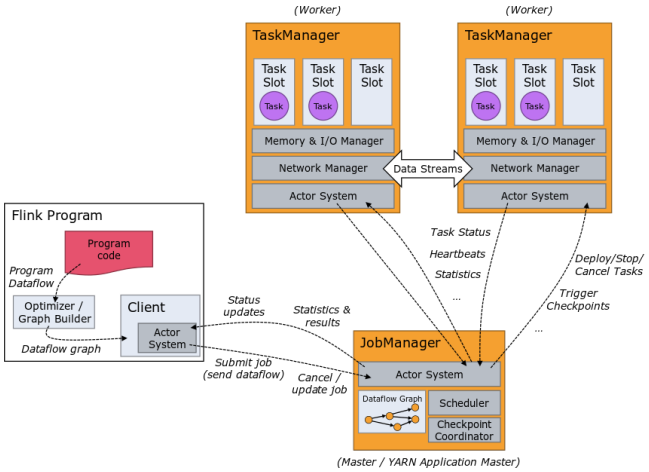

Figure 9: Flink Architecture

Regarding the framework of our Stream Processing Engine(SPE), we chose to implement Apache Flink v1.20.0 to manage the stream of data generated by drivers on the road and perform stateful computations. Figure 9 displays the Flink architecture, which is a key factor in the implementation of Speedly. The JobManager component, responsible for the deconstruction and distribution of jobs across TaskManagers, allows us to aggregate the stream of geolocation data across several users at a time. As users continually send updates regarding their displacement, the information will take up Task Slots, delegated by the TaskManager, and be processed to construct several rolling averages of velocity from that user and other clients in the area. With this information, a map visualization can be generated based on information from a large quantity of clients, containing broad-spectrum information about the state of traffic with minimal lag. Lastly, an important point to be made about the Flink framework are the added benefits of fault-tolerance, exactly-once consistency, and availability. In addition, the Speedly model addresses the previous issue of government surveillance and user anonymity, since the only transfer of data between the client and analysis tools are map coordinates.

Each client is sending a data stream to our system, which is comprised of tuples containing a user ID so data associated with a user can be grouped, the location of the user, and a timestamp. The timestamp has multiple purposes, the first is to calculate speed and direction as already stated, but the timestamp is also used to filter out old data, as traffic is always moving, and if data is outdated, it is likely useless in calculating current traffic, and can thus be disregarded or discarded.

A computation model will also be necessary to describe the flow of data and operations performed on the data. An example of the computational model is shown in Figure 10, however this is currently subject to change. Even though

the computational model is subject to change, many of the tasks shown in the potential model will be needed, such as: Receiving data, filtering the data to the appropriate database or server, computing heat maps, and sending heat maps to clients.

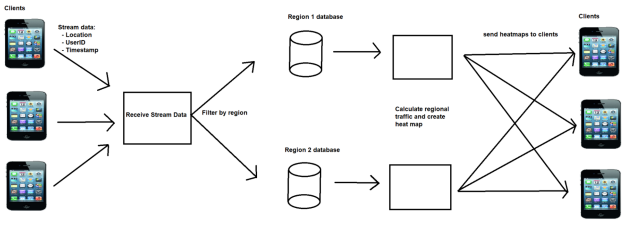

Figure 10: Potential Computational Model

As the potential quantity of data for a system like this is quite large, the system will have to be made in a way that allows for scalability. One way we may accomplish this is by dividing road maps into a grid, and only assigning individual database and operation servers to serve clients within their region. This could also help add fault tolerance to this system, as regions could potentially overlap, meaning all clients would fall into multiple server regions, and if one region server fails, one of the overlapped servers could serve those clients. An example of how this could work is shown in Figure 11, where a map is split into two overlapping grids, where the squares of one grid overlap with a quarter of four squares from the opposite grid.

Figure 11: Potential Grid for Region Based Servers

6 Evaluation

Due to the minimal scale of our initial model, our goals were limited to a handful of requirements in order to be considered a success. The first was a guarantee of performance, meaning the system had to work consistently, especially with varying workloads. This is a simple requirement, but regardless, many first-time issues were posed as we combined different services and applications.

The second major consideration was scalability. Supporting tens of thousands of simultaneous connections was crucial to maintaining an accurate, real-time view of the traffic network. Separate from the vehicle mapping framework, an entirely separate interface would have to be maintained to interact with and command the traffic infrastructure. One such limitation of Speedly is the current lack of this traffic interface, but we believed it to be outside the scope of our novel contribution. Ultimately, the current state of traffic systems in the United States leaves much to be desired, and we're likely decades away from any advancement in traffic light infrastructure. Nevertheless, the need for an information system that can theoretically supply this movement remains well within reach. Suffice it to say, the workload presented in this theoretical cluster is formidable.

The last requirement is latency. For the proposed system to have any chance of outperforming current solutions, an enforced maximum latency for the processing of vehicle data would be essential to the performance of the system. Stale data of cars on the road would inhibit the predicative capabilities of a traffic model, especially on shorter stretches of road between lights.

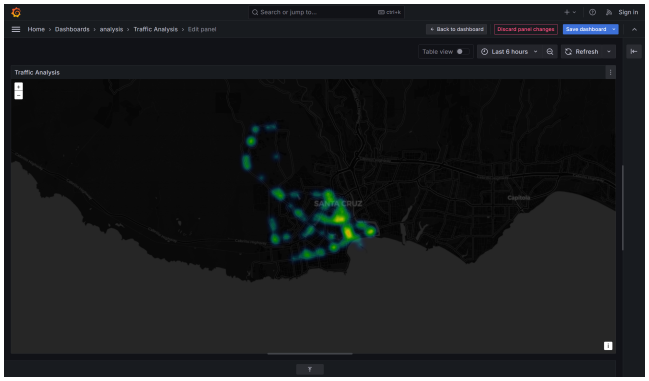

Figure 12: 128 connections

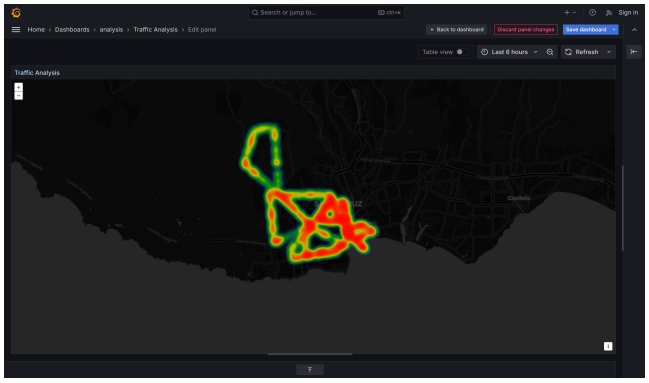

Figure 13: 1024 connections

We test our Speedly application against two different cases, one where the user pool size has 128 connections on the road and second has 1024. One of the reasons for the minimal

size of the test cases is due to the hardware limitations of the test machine, which contained an Intel i5-1135G7 and 16gib DDR4 memory. While we understand that the performance of Speedly is paramount to its applicability, we note that the underlying Flink architecture [1] has proven to offer sufficient throughput and scalability. As shown in Figure 12, the test model maintains 128 continuous, separate streams of data to the Speedly TaskManager. As each of these simulated vehicles ping their velocity data to the TaskManager, the individual data packets are batched and subsequently processed. For the demonstrative purposes of Speedly, we submitted the results of the serialized vehicle data to a postgis database. With the Grafana solution we crafted, the location of individual cars was displayed with a specific query on the destination database. Ultimately, we find that, even with the 1024 connection test case, Grafana was able to display the updated information with minuscule lag, which speaks lengths to the computing capabilities of the Flink system.

7 Future Work

Since this is the initial model of the program, there is still a lot of room for growth (as shown in our Evaluation section). There is many aspects that could have been done better, and there is also functionality and optimizations that we didn't have the time to fit into our project.

One optimization that could be made is to optimize the database for streaming live data. Since only fresh data in the database is going to be used in streaming, old data doesn't need to be considered in real time applications. This means old data doesn't need to be readily available. If we decide we don't ever need data we could just remove old data from the database. However if we decide we ever want to use historical traffic data for things such as training machine learning models we could decide to make improvements such as time-based partitioning. Time-based partitioning is just dividing the database into partitions based on a time stamp for more optimal querying of data if you know the time frame you are looking at. For us that would just be the most recent partition but it also allows us to keep past data without having to worry about it degrading query performance.

What we could also do is make data points snap to roads and have a set direction on that road. That would entail having a way for map data to be stored in the project, and then having a Flink operation find the nearest point on a road and sending that data to the database rather than the actual location. It would also have to find which direction the user is moving on that road. This would help deal with the inaccuracies of user data and help generate cleaner heat-maps.

We also have yet to introduce Flink's checkpointing system to our project. The checkpointing system is Flink's solution to fault tolerance in a stateful system. It works by saving the state of an operator in some persistent store, so in the event of a fault the state of that operator can be restored and execution can continue. This isn't particularly helpful currently in our project as we aren't performing any stateful operations, but if we are to implement statefulness this is the easiest way to make it fault tolerant. A picture explaining checkpointing can be seen in Figure 14.

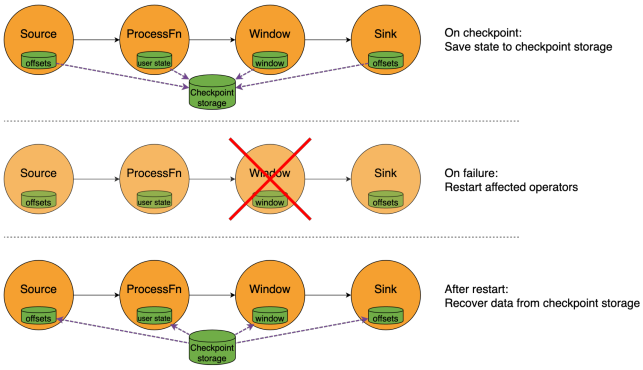

Figure 14: Flink Checkpointing

One other potential improvement that could be made is to have an actual website to generate heatmaps. Currently the project generates a heatmap using an instance of Grafana that is accessed through a local host. Ideally, we would have some website that could be generated that just provides the heatmap for users to see. This also would be helpful as other websites could potentially embed the heatmap for viewing there, similar to how Google Maps is often used. Since the goal of this project was a more open ended framework that could be used as a website for clients or for services such as intelligent traffic light controllers to receive live traffic data, having this website isn't that crucial as the end user of this would likely make the application that receives data from Speedly.

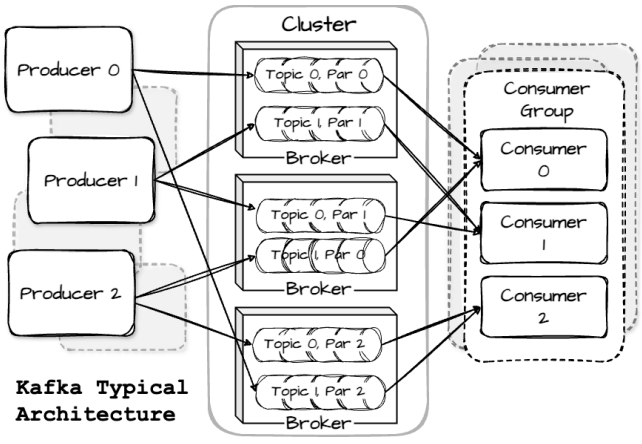

Figure 15: Kafka Example

Another improvement that we wanted to implement but ran out of time to for was Apache Kafka. Apache Kafka is

basically an input buffer for stream data. It works by having several producers pushing stream data to a cluster of Kafka instances. This data is placed into topics and then some consumer(s) take the data out of the Kafka instances for them to use. We could use Kafka to buffer our client input by placing it right before our Flink instances in our architecture. We also could place them between Flink and postgres in our architecture if we wanted to. Figure 15 shows a diagram of how Kafka is typically used in a streaming system.

8 Conclusion

Speedly exemplifies the benefits of integrating stream processing into traffic information systems to address the limitations of present-day traffic infrastructure. Speedly provides fresh traffic data with the capacity to support large amounts of data coming in at various rates. As of now, Speedly is not a fully fleshed out model ready for deployment, but serves its purpose as proof-of-concept for further development of traffic analysis. We can see that this model is a functional framework for stream-based traffic processing, and that it fulfills our goals of a backend service that expressly stores and serves data on the current conditions of traffic. There are still functionalities and optimizations that could be added as made clear by the future work section, but the groundwork has been laid.

9 Team Contributions

Ankur Ahir Ankur was responsible for the integration of the Grafana interface and the deployment of the postgis database. He also contributed a great deal to the code surrounding the Flink architecture, building the streaming logic that aggregated and batched information from the Python socket.

Jason Mack Jason worked together with the others in the team to figure out which packages and tools to use for developing Speedly, with some of the tools not being used in the end, such as Kubernetes, Helm, and FyFlink. He wrote most of the client simulation script with some help from Ankur. Jason also created the SQL query that's used by Grafana to receive data from the database. He also did much of the work on making the initial iteration of the slideshow for the presentation, with each teammate then fleshing out their respective parts. He also worked on debugging the project throughout development.

Sallar Farokhi Sallar was responsible for a majority of the documentation and presentation. He also helped to set up the initial logic for the Flink JobManager, albeit using PyFlink. After we switched to JDBC, the majority of his efforts were focused on debugging the application and compiling all separate modules together into a single docker-compose. He also made Apache Flink work within a docker container which the team had heavily struggled to do.

References

[1] Asterios Katsifodimos and Sebastian Schelter. "Apache Flink: Stream Analytics at Scale". In: *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)* (2016), pp. 193–193. DOI: 10.1109/IC2EW.2016.56.

[2] Iisakki Kosonen, Andrzej Bargiela, and Christophe Claramunt. "A distributed Traffic Monitoring and Information System". In: *Journal of Geographic Information and Data Analysis* 3 (Jan. 1999), pp. 30–39.

[3] Ashish Rajeshwar Kulkarni, Narendra Kumar, and K. Ramachandra Rao. "Efficacy of Bluetooth-Based Data Collection for Road Traffic Analysis and Visualization Using Big Data Analytics". In: *Big Data Mining and Analytics* 6.2 (2023), pp. 139–153. DOI: 10.26599/BDMA.2022.9020039.

[4] Sin-Chun Ng and Chok-Pang Kwok. "An Intelligent Traffic Light System Using Object Detection and Evolutionary Algorithm for Alleviating Traffic Congestion in Hong Kong". In: *International Journal of Computational Intelligence Systems* 13.1 (2020), pp. 802–809. ISSN: 1875-6883. DOI: 10.2991/ijcis.d.200522.001. URL: https://doi.org/10.2991/ijcis.d.200522.001.

[5] Hengshuo Yang et al. "Latent Factor Analysis Model With Temporal Regularized Constraint for Road Traffic Data Imputation". In: *IEEE Transactions on Intelligent Transportation Systems* (2024), pp. 1–18. DOI: 10.1109/TITS.2024.3486529.