Final Report: Public Transport Data Lake with Analytics

CS4225 Big Data Systems for Data Science

Group members:
Royce Ho (A0199477A),
Ong Wei Sheng (A0206042X),
Liu Yuhui (A0225308L),
Jodi Choo (A0222236R),
Tan Shee Hui (A0197153Y),
Lim Zi Xuan Jeremy (A0200402J)

Table of Content

Introduction and Literature Review

Motivation and Background

Traffic congestion is a significant problem in urban areas. Studies have shown and proven the threat of traffic congestion to the economy, as well as public health and well-being. Traffic jams and longer travel times culminate in loss of productivity and waste of fuel. The increase in vehicle tailpipe emissions also contributes to air pollution. If left unchecked, the traffic congestion issue slows economic growth and increases public health risks. An increase in population and the rise of various ride-sharing services further increase the number of vehicles on the road, worsening the traffic congestion problem.

Our project involves building a data lake for public transport data from the Land Transport Authority's (LTA) datamall APIs. We designed an end-to-end solution from data collection to data processing and system monitoring using a variety of platforms, including multiple services from Amazon Web Services (AWS). Data collected was processed using Apache Spark, an analytics engine used for large-scale data processing, and visualized in the form of graphs. Machine learning was also applied to some datasets from our data lake, to predict trends as well as to detect congestion through CCTV images via object detection algorithms.

These insights may be used by decision-makers to devise measures that alleviate the negative impacts of traffic congestion. Transport providers can use these insights to improve scheduling plans, increasing the efficiency of public transport to reduce lost time spent in traffic jams, while delivery companies can optimize delivery times based on traffic predictions to increase the productivity of workers.

Other related works and research

There are some previous works being done that use public transport data, traffic congestion data etc. and we will go review what each of them does. Firstly, TomTom's Singapore traffic report collects traffic data to visualize and show traffic congestion to users through a dashboard and allows users to select the relevant information that it wishes to view about traffic. However, this is merely just the collection and visualization of data as it does not analyze or learn the data obtained, therefore analyzing the data collected to learn some useful trends and insights along with the visualization of the data is one of our goals.

Some of the relevant research articles that analyze traffic data include a paper called FASTER: Fusion AnalyticS for public Transport Event Response (Sebastien Blandin, Laura Wynter, Hasan Poonawala, Sean Laguna, Basile Dura), which involves methods ranging from statistical machine learning to agent-based simulation to monitor and predict commuter movement in order to effectively respond to unforeseen public transport incidents and to keep up the level of efficiency of public transports. Real-time data processing using Spark/Hadoop is used here, alongside Redis for fast operations.

A Data-Drive and Optimal Bus Scheduling Model With Time-Dependent Traffic and Demand (Yuan Wang, Dongxiang Zhang, Lu Hu, Yang Yang, Loo Hay Lee) involves the collection and analysis of traffic data to efficiently determine whether more bus services are required at

perhaps a certain time of the day. The model also takes into account operating expenses to see if the benefits of increasing the frequency of bus services outweigh the negatives of the increase in operating expenses. Our main aim for our project is to combine the interactive front end of current works with insightful analysis of the data being collected.

Methodology and Experimentation

Data Lake Architecture

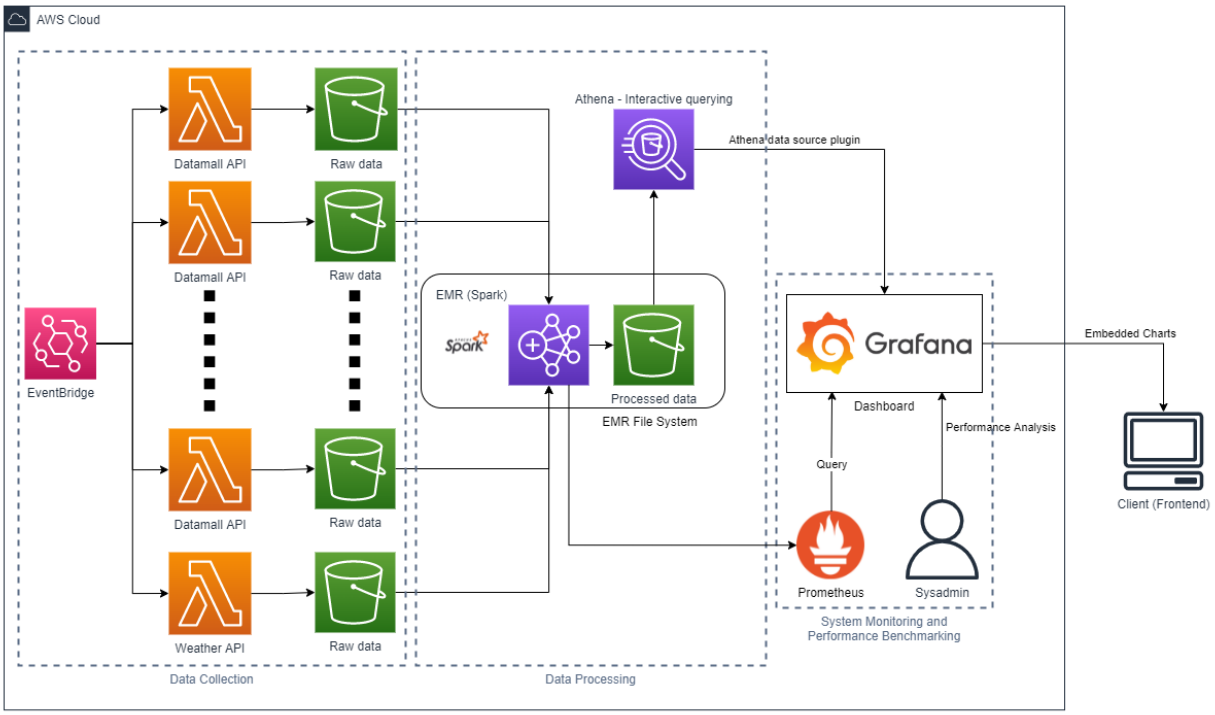

Figure 1: Cloud Architecture

Figure 1 describes the cloud architecture implemented, broken into 3 main components, data collection, processing and system monitoring. Below is a summary of the 3 components, which will then be further elaborated on in the subsequent sub-sections.

Data collection: Lambda functions are triggered by EventBridge to pull in data via API calls. The data is stored in their respective S3 buckets.

Data processing: Spark application in EMR reads from S3, and writes output back to S3. Athena queries out the data in S3 to be visualized through Grafana dashboards.

System monitoring: Node exporters export out metrics to Prometheus (time-series database) and Grafana queries and visualize these metrics.

Datasets and Data Collection

9 LTA Datamall APIs and 1 NEA weather API for rainfall were used to pull in raw data. We chose them as they were Real-time APIs without any strict thresholds on the number of calls. Out of all the raw data (10 APIs) that were collected, we shortlisted them and ended up processing data on car park availability, speed bands, rainfall and estimated travel times.

Amazon EventBridge triggers the lambda functions (written in node js) to be called every 5 and 10 minutes, depending on which group they are in, which uploads the results of the API call to its respective S3 bucket as JSON objects, with the filename being the timestamp in ISO 8601 string format.

Data Processing with Spark

We did our data processing entirely on the cloud through the use of Amazon ElasticMapReduce (EMR). Amazon EMR was used as it helps remove the need for managing clusters and simplifies big data processing. The Spark clusters being created via Amazon EMR would then allows us to process raw data from S3, utilizing Amazon's EMRFS, which is an implementation of HDFS that all Amazon EMR clusters use for reading and writing regular files from Amazon EMR directly to Amazon S3. We also used PySpark, which is an interface for Apache Spark in Python. This allows us to write Spark applications using Python which is more user friendly. The Spark application would then be uploaded to Amazon EMR to be run, with the output written back into S3.

Querying with Athena into Grafana

Once the output of the spark application is stored back in S3, we have to query out the data. Our tool of choice is Amazon Athena, a serverless interactive query service built on top of Presto, an open-source distributed SQL query engine. Athena is able to query out of S3 using SQL.

To visualize our data in charts, we use Grafana, which allows for dashboards to connect to data sources. Using the Athena data source plugin for Grafana, we were able to write our SQL queries directly in Grafana, which connects to Athena and visualize our S3 buckets. Figure 2 shows exactly how this works on Grafana's UI.

Figure 2: SQL queries in Grafana connect directly to Athena to query S3 data (EMR output)

System monitoring and performance benchmarking

To monitor the performance of our system, we used Prometheus, a time-series database, and Grafana, analytics and interactive visualization web application, for this use case. As shown in Figure 3, it involves three parts:

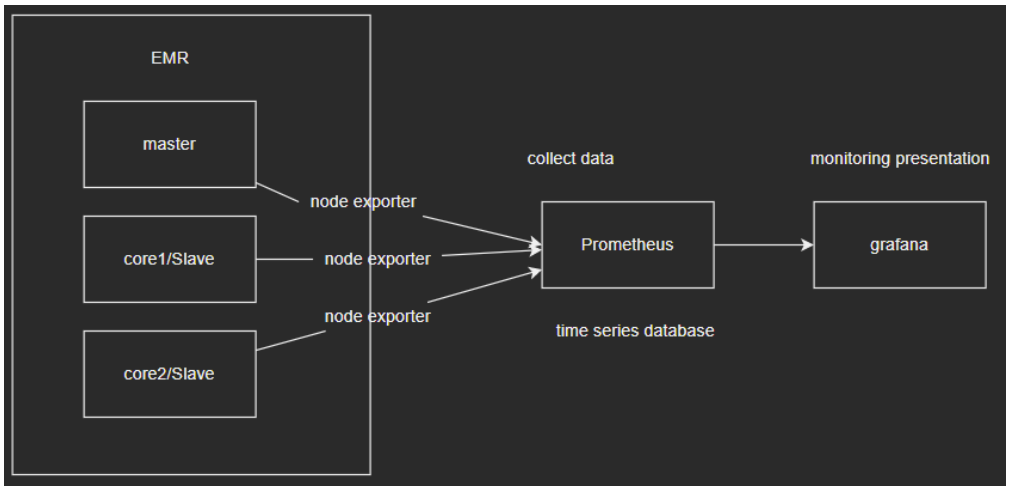

Figure 3: Data pipeline for analytics

Firstly, we need to install a “node exporter” at all the target servers to be monitored. Our EMR setup involves a master node and two core nodes. As a result, we need to install “node exporter” and run the exporter in each instance. The exporter will keep running and listen on port 9100. Whenever there is an incoming request, the “node exporter” will check the system statistics and respond.

On Prometheus’ side, it retrieves the system information from the “node exporter”. The scrape interval is set to 5 seconds, meaning that every 5 seconds, the system stats will be updated. Prometheus is a time-series database and it can handle lots of queries to be used to query insights of the data.

Lastly, Grafana provides the graphic interface and presentation dashboards to show the monitoring data. Grafana can use Prometheus queries to retrieve data and further generate chars based on the retrieved data.

Machine Learning

To add value to our analytics, we utilised different machine learning models to analyse several data and predict the possible outcomes for the day ahead. Firstly, we wanted to detect how many cars there are on the road at any given point in time and at any point in Singapore. Since there was no data related to the number of cars on the road, we had to train a model to detect and count how many cars were in the images as that was the only data provided by the API. This was done using a kind of neural network called Faster R-CNN which is able to detect objects in images. We then utilised a pre-trained backbone, ResNet50 to do transfer learning and to train on the model itself. This helped to save time as we did not have to train for a long time to yield a decent result. (insert image here)

Figure 4: Trained faster R-CNN performance with Non-Maximum Suppression applied

Spark was used to aggregate some data so that it would be easier to train the neural network. Pytorch was the underlying python library and we trained it on Google Collab as they provided a free GPU that is essential in computation.

Secondly, we did a simple time series prediction using a fully connected network. Like before, we wanted to predict some outcomes such as the available carpark lots at certain locations based on the past results. This was once again carried out by firstly using spark to clean out any data that is not important and to aggregate them so that it would be easier for the network to work on them. For this, we changed to the TensorFlow library as it provided many readily available time series models ranging from simple linear models all the way to multi-step models that provided prediction 24 hours ahead of time.

In order to keep the model updated, we would train the model with new data once every week. This allows for the model to be trained in mini-batches which is important so that data noises do not affect the model. In addition, this will reduce the load on the system as there is a lot of pre-processing and training which will eat up the CPU usage etc…

Frontend

To visualise our results meaningfully, we built a map-based frontend using the react-leaflet library as the base of our interactive map. Afterwhich, two custom layers are created (i.e. congestion layer and taxi availability layer).

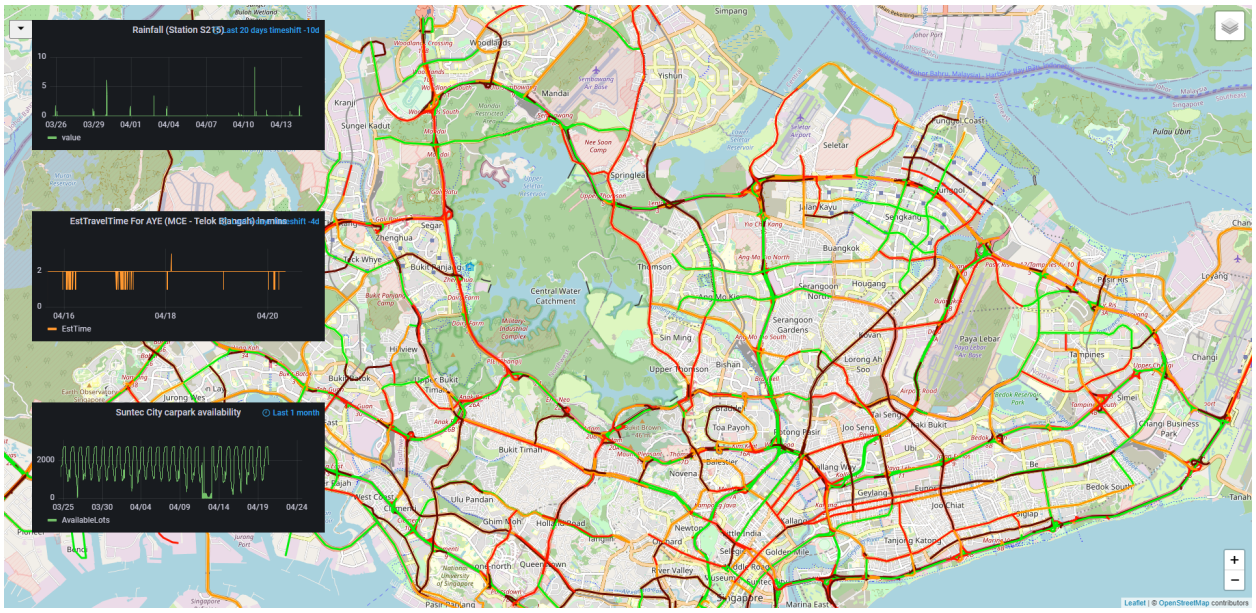

Figure 5: Congestion Layer

The congestion layer is used to visualise the predicted level of traffic congestion on most major roads and expressways in Singapore. The layer colour-codes these roads according to the level of congestion, which is categorised on a scale of 1-4.

The scale is as follows: 1(green) - clear roads, 2(orange) - light congestion, 3(red) - moderate congestion, 4(maroon) - heavy congestion. We represented each road section with an array of coordinates and created our own GeoJSON file for this layer.

Figure 6: Taxi Availability Layer

To represent the latest data on taxi availability in Singapore, we actively pull from our Amazon S3 bucket and render a layer with taxi icons for users to view.

Embedded graphs are rendered from a local Grafana instance to further consolidate our data and observe the trends across various metrics over time. The metrics plotted are the amount of rainfall, estimated travel time and carpark availability.

Discussion of results

Public transport data

Figure 7: Carpark availability at Suntec City

One of the datasets of interest was the carpark availability dataset. Figure 6 shows a particular carpark's availability in Suntec city for over a month. The graph allows us to note abnormalities such as the low availability of lots from 12/4/2022 to 13/4/2022 at a glance.

Other data such as rainfall and estimated travel times were similar, in the sense that they followed daily trends that were repeated, which is expected.

Performance benchmark results

The final monitoring dashboard result is shown in Figure 7:

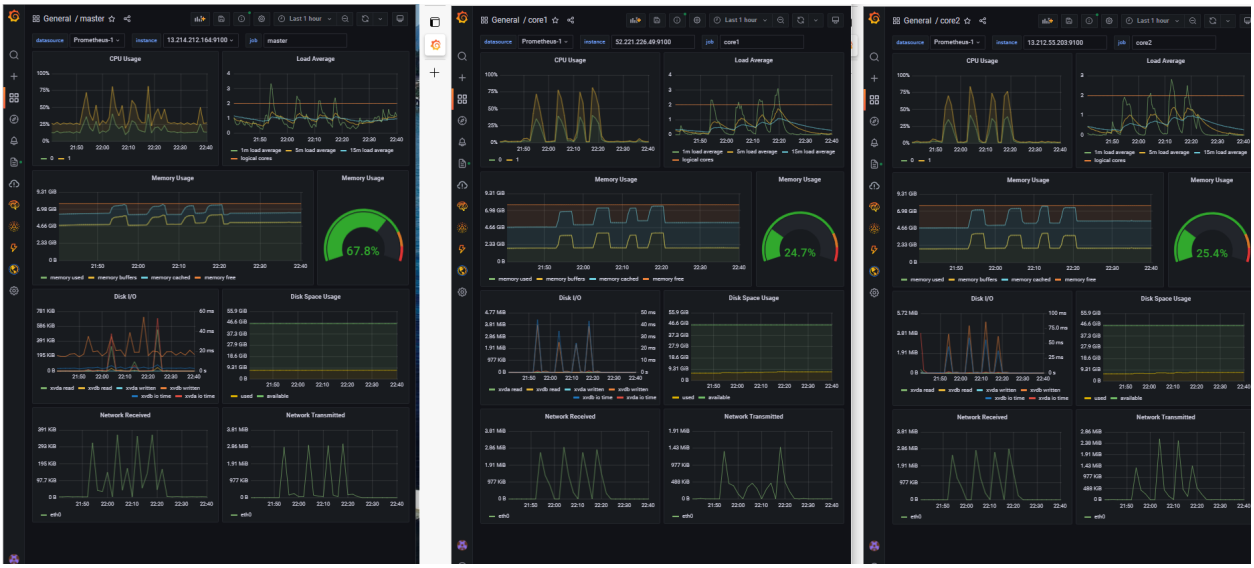

Figure 8: 3 Grafana dashboards

From right to left, it is the dashboard for the master server, core server 1 and core server 2. The dashboard can clearly show most of the key performance parameters: CPI usage, Message usage, Disk I/O, Disk usage and Network I/O. From the graph, there are four peaks in most of the graph, and that is the four steps we have run in EMR, we can see a clear increase in consumption of different resources. During the run of the spark application, there is an average spike of about 15-25% in memory usage.

Problems encountered and lessons learnt

We used Grafana to plot out charts and embed them in the frontend, as well as having a dashboard for system monitoring to measure metrics. However, Grafana Cloud does not allow for easy embedding of charts, due to a need to request customer service to make changes to the config file to allow iframe embedding of charts and the fact that the frontend has to authenticate with Grafana Cloud in order to view the iframes. As a result, we had 1 Grafana Cloud instance for system monitoring and 1 self-hosted docker Grafana instance for chart embedding due to the need to enable embedding without the need for authentication (anonymous access). This setup allowed us to skip the tedious implementation of setting up authentication as well as making the embedding work.

We obtained the coordinates used for the frontend congestion layer using the LTA Datamall API. However, the data obtained from the API endpoint did not contain sufficient coordinates to plot and draw the roads meaningfully. To counter this problem, we obtained a separate dataset which contained a sufficient number of coordinates for each road. The existing coordinates we obtained from our analysis were matched to the respective road from this separate dataset.

We learnt that it was essential for the team to sync up on the full process of the data flow and how the format and structure of data should look at each stage. This was so that everyone could work on different stages of the data lake pipeline in parallel and that the system would work once we linked up the various stages.

Personal contribution

The workload was evenly distributed across all 6 members with their major tasks outlined as shown below:

1. Frontend development: Shee and Jodi
2. Machine learning: Jeremy
3. Spark application: Wei Sheng
4. AWS services: Royce
5. System monitoring: Carter

More details are highlighted in the “Workloads of each group member” section below.

Project summary

This end-to-end data lake project covers all the technical aspects of a data lake, from data collection and processing to system monitoring. The majority of this project utilised available cloud services from Amazon Web Services, which allowed us to do much in little time. This project aims to be a product that helps alleviate traffic congestion issues in Singapore and improve transport efficiency by processing historical data to create prediction models for crowd surges. Road-related data, such as taxi availability, is constantly collected through the LTA Datamall API. Weather data is also collected through the NEA weather API to allow us to draw relations between weather and road situations. Data from these two sources are the core of our data lake, which is then processed and visualized for users.

References

Singapore traffic report: Tomtom traffic index. report | TomTom Traffic Index. (n.d.). Retrieved March 9, 2022, from https://www.tomtom.com/en_gb/traffic-index/singapore-traffic/

Passman, D. B. (2013). Fusion analytics: A Data Integration System for public health and medical disaster response decision support. *Online Journal of Public Health Informatics*, *5*(1). https://doi.org/10.5210/ojphi.v5i1.4522

Wang, Y., Zhang, D., Hu, L., Yang, Y., & Lee, L. H. (2017). A data-driven and optimal bus scheduling model with time-dependent traffic and demand. *IEEE Transactions on Intelligent Transportation Systems*, *18*(9), 2443–2452. https://doi.org/10.1109/tits.2016.2644725

Tan, D., & Lang, F. (2008, August 10). *Monitor and Optimize Analytic Workloads on Amazon EMR with Prometheus and Grafana*. Amazon. Retrieved March 9, 2022, from https://aws.amazon.com/blogs/big-data/monitor-and-optimize-analytic-workloads-on-amazon-emr-with-prometheus-and-grafana/

Kranc, M. (2017, November 2). *Amazon fills a big data hole with Athena*. Datanami. Retrieved March 9, 2022, from https://www.datanami.com/2017/10/17/amazon-fills-big-data-hole-athena/

Workloads of each group member

This breakdown includes the major and minor task breakdown of each individual, as there were some overlapping tasks.

Name	Tasks
Shee	UI Design of application, frontend web development with React, libraries used include react-leaflet, literature research, Report: frontend, problems encountered, discussion of results
Jodi	
Jeremy	Data analysis and machine learning on datasets, frontend web development and converting data required by the frontend to the correct format., Report: machine learning, frontend
Wei Sheng	Spark application development, testing of spark application on EMR, Report: introduction and literature, spark
Royce	Design of architecture, provisioning and maintenance of AWS resources, setup and testing of Lambda, S3, EventBridge, Athena, EMR and local docker instance of Grafana for chart embedding, Report: AWS services breakdown, discussion of results
Carter	Setup and testing of node exporters in EMR, Prometheus in EC2 and Grafana Cloud dashboards for system monitoring, Report: Monitoring results, provide feedback for architecture