VU Advanced Internet Computing (184.269)

Project Topic 1 – Stream Processing for Spatially-Distributed IoT Systems

stream processing, kafka, redis, storm, data-intensive

TU Vienna – Distributed Systems Group

Keywords: Stream Processing, Data Topologies, Real-time System

Description

IoT systems which operate within a dynamic spatial environment are becoming ubiquitous; think of a taxi fleet within a smart city optimizing spatial distribution with respect to passenger demand, or a manufacturing floor co-habited by humans and robots. Those represent an important class of cyber-physical systems, which are faced with the manifold challenges that a dynamic spatial environment brings. They demand operational management to handle large amounts of data, often in real time.

This project entails realizing a monitoring dashboard, which is updated in real-time based on spatial streaming data obtained from IoT devices. To provide information in real-time, data needs to be processed by a sequence of different stream processing operators, such as filters or aggregators.[1] These individual stream processing operators form a stream processing topology, as you can see in Figure 1.

The goal of this stream processing topology is to transform a series of geographical locations by different taxis into value-added information that can be visualized on a dashboard to monitor the taxi fleet. In this task, you are going to combine several data processing frameworks, such as Apache Storm,[2] Apache Kafka,[3] and Redis.[4]

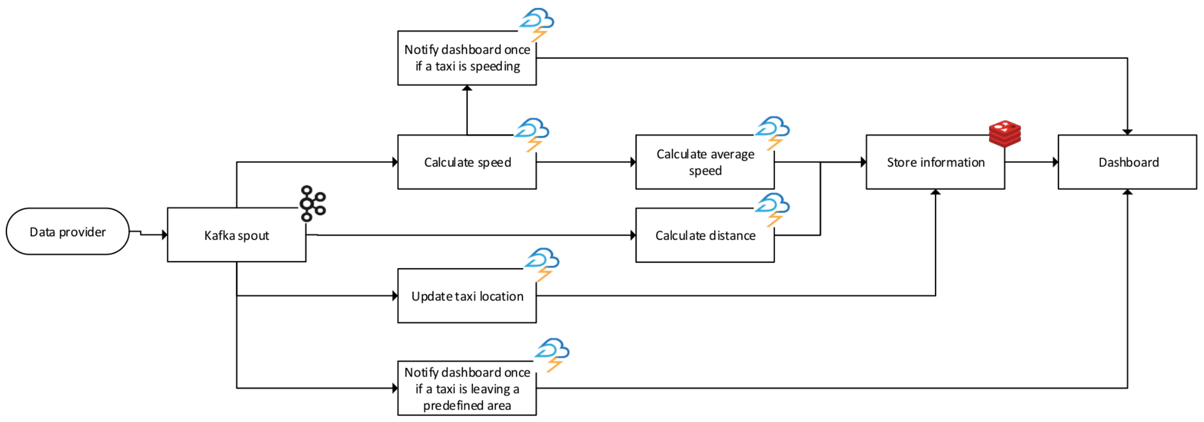


Figure 1: Stream Processing Topology.

The preliminary task for this project is to populate an Apache Kafka instance with the location information of the taxis from the T-Drive dataset[5]. This data set contains the location information of different taxis and your task is to replay the location information according to their timestamps, and to submit the replayed data stream to Apache Kafka. Apache Kafka acts then as a spout for Apache Storm, which processes the location information to obtain the value-added information and forwards the value-added information to a data storage

[1]Bugra Gedik et al. "SPADE: the system s declarative stream processing engine". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* ACM. 2008, pp. 1123-1134.
[2]https://storm.apache.org
[3]http://kafka.apache.org
[4]http://redis.io
[5]https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

as well as dashboard. The processing operators of this stream processing topology are implemented by several stream processing operators, which are described in detail below:

The **Calculate speed** operator calculates the speed between two successive locations for each taxi, whereas the distance between two locations can be derived by the Haversine formula. This operator represents a stateful operator because it is required to always remember the last location of the taxi to calculate the current speed.

The **Calculate average speed** operator aggregates all speed information from the previous stream processing operator and calculates the average speed of all single speed information for a specific taxi. You may use this formula for calculating the average speed:

$$S_{avg} = \frac{1}{S_n} \sum_{i=1}^{S_n} S_i \tag{0.1}$$

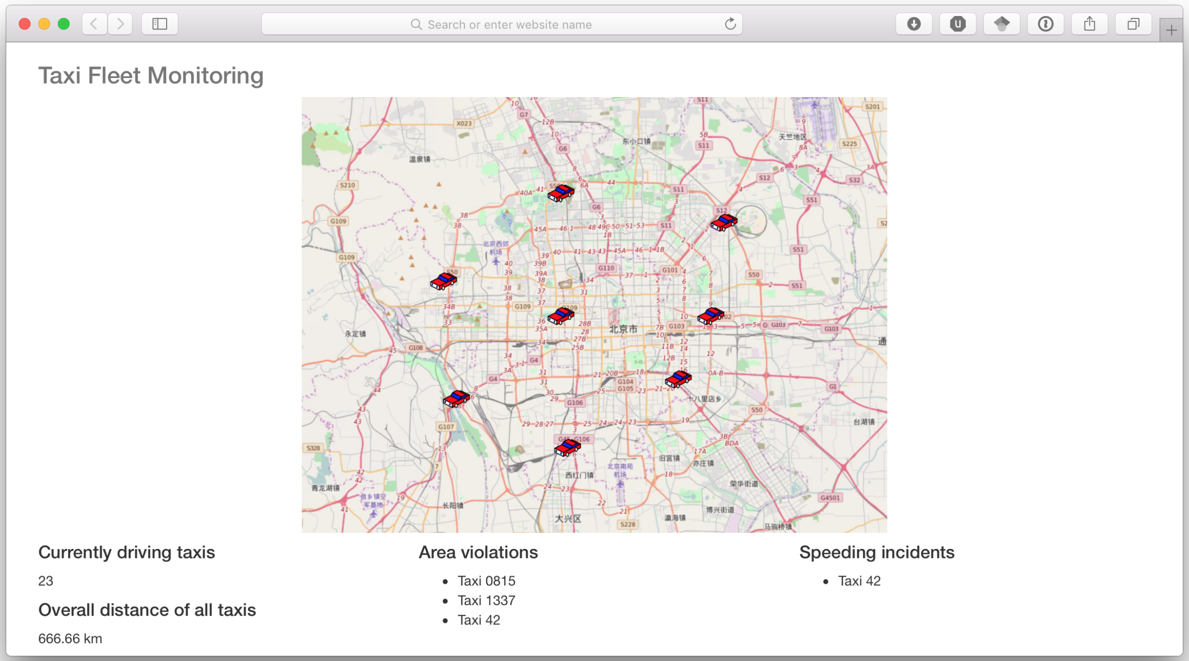
S_n represents the number of all individual speed information and S_i represents the individual speed information. This operator is stateful, since it has to collect all individual speed information to calculate the average speed for a taxi.

The **Calculate distance** operator calculates the overall distance for each taxi, whereas you can also use the Haversine formula.

The **Store information** operator stores the average speed for each individual taxi and the distance of the taxi ride so far to a Redis data structure store.

The **Dashboard** retrieves the data every 5 seconds from the data store, calculates the amount of all currently driving taxis, aggregates the overall distance covered by all taxis and displays the calculated numbers and current location of the taxis on the dashboard.

The **Update Taxi Location** operator forwards the current location of the taxi to the data storage.



Taxi Fleet Monitoring

Currently driving taxis

23

Overall distance of all taxis

666.66 km

Area violations

- Taxi 0815
- Taxi 1337
- Taxi 42

Speeding incidents

- Taxi 42

Figure 2: Fleet Monitoring Dashboard

The **Notify dashboard once if taxi is leaving the predefined area** and the **Notify dashboard once if taxi is speeding** operator analyze the speed respectively location of each taxi and whenever a taxi is faster than a predefined speed limit, i.e., 50 km/h or leaves a predefined area, this operator forwards the information to the dashboard. The dashboard should issue a warning whenever a taxi is leaving the area with a radius of 10 km around the center of the Forbidden City in Beijing. The dashboard should discard any information about taxis who leave the area with a radius of 15 km around the center of the Forbidden City in Beijing.

Stage 1

Your task in Stage 1 is to setup and configure Apache Kafka as well as Apache Storm. Furthermore, you have to implement the data provider to fill Apache Kafka with data. Therefore, you may need to preprocess the data provided by the T-Ddrive data set trajectory data sample[5] dynamically, i.e. using a script. The data in the text files are structured as follows: The taxiId represents a unique identifier for one taxi, across all locations, the timestamp the time when the location was recorded and latitude respectively longitude describe the geographical location. It is advisable that the data provider emits an end token as soon as the data for one taxi, i.e., one data file, has been submitted to the system, to signal that this taxi has stopped and does not emit any further information.

Tasks:

1. Setup and configure Apache Kafka.
2. Implement the data provider, which represents a producer for Apache Kafka. This data provider retrieves the location information from text files or an SQL database and submits it to Apache Kafka. The data needs to be replayed in the same order as it was recorded, i.e., all location information with the same timestamp need to be submitted at the same time. It is required to make the submission speed configurable to apply different submission speeds during testing and actual performance tests of the system. Furthermore, it is necessary to make the amount of displayed taxis configurable for demonstration purposes.
3. Setup and configure Apache Storm.
   - Implement a spout for Apache Kafka.
   - Implement the required operators for the topology. For those operators that maintain a state, you have to implement some kind of state management. To achieve this, you can either use the Trident State[6] or implement your own state management based on a simple key-value storage solution, e.g. based on Redis.
   - The processed data needs to be stored in a Redis in-memory instance.

Stage 2

In Stage 2, you have to implement the dashboard (see Figure 2), which retrieves the data from the Redis data storage or the stream processing topology and visualizes it for the user. You also need to evaluate the performance of your implementation, by determining the maximal submission speed your implementation can handle.

Tasks:

1. Implement a Web-based GUI, featuring all the elements in Figure 2.
2. Prepare a reduced dataset to visualize the movements on the dashboard.
3. Retrieve the processed information from the Redis data storage or stream processing topology and display the data in real-time, i.e., show taxis in the map and add a new incident for every violation retrieved from the stream processing topology.

Bonus Points

[6]https://storm.apache.org/documentation/Trident-state

- 1-5P (Optimization) Optimize your implementation to achieve a high throughput while maintaining valid results. For optimization ideas, you may take a look at the publication by Hirzel et al. [7] You are awarded for each successful optimization. However, you must document and adequately benchmark your applied stream processing optimizations and configurations for Apache Storm. Highlight their effects on the throughput, which can be monitored by the Apache Storm GUI. Lastly, briefly explain how each optimization improves throughput.

Technical Report

As lead engineers, you will enjoy higher pay and other privileges. However, you will bear more responsibilities and be held accountable for every code deployed under your leadership. See Hamlen's guidelines on technical writing [8]. Use the single-column Springer Lecture Notes in Computer Science latex template [9]. The page limit is 12 (excluding references). Make reasonable assumptions about how much text you need, and tone down on redundant wording. Take responsibility for your statements and decisions by avoiding *weasle words*. Be bold, but underpin your assertions with sufficient references. For example, a report littered with sentences such as, "One might argue that the analysis of flying pigs may be an effective way to obfuscate the main contribution of a possibly weak paper potentially," will lead to an insufficient grade.

Report Structure

The following describes the structure of your report. The page counts per section are suggestions.

Preamble, Responsibilities, and workload distribution (2 Pages)

Create profiles for each team member (you may roleplay). Clearly define roles describe your background (education, prior work or hobbyist experience, etc.), strengths, and familiarity with tech stacks. To grow together as a team, set learning outcomes and, on completion, reflect on whether you've hit your targets in the conclusion. A role only implies that the student is responsible for a part of the project and focuses on specific aspects. However, it does not mean they should work exclusively on their role-specific tasks. We highly encourage students to focus on topics they have no familiarity with.

Design Decisions and Technology Choice (1 Page)

Choosing a tech stack due to the preferences of the engineering team is a valid argument. Still, you should solidify your choices by highlighting the strengths of the tech stack. Moreover, you should also list the risks and weaknesses of your technology. Especially figure out how stable and "battle-tested" your libraries/programming languages are.

Code Generation (4 Pages)

It is **mandatory** for you to use code generation beyond simple completion. For system critical code (i.e., anything that is not basic boilerplate and service-agnostic utility code), clearly state which parts of your codebase are generated (At least three prompts). Include the tool/LLM used (e.g., Copilot X, GPT-4, etc.) in the queries. Generated code, which you had to fix or significantly alter, should be embedded in your report as syntax-highlighted code (Tip: LaTeX mint). Mark the parts you changed, explain what the issue with the code was, and try to reason whether your prompt was insufficiently clear or if the fault was with the generator. Note we will check the prompts, and we fully expect the LLMs to not generate the correct code on the first try for non-boilerplate code.

Tip: Implement/define interfaces and write tests against them. Include the interfaces (with docstring!) in your prompt, and instruct the generator to comply with the specifications.

[7]Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. ACM Computing Surveys (CSUR), 46(4):46, 2014.

[8]https://personal.utdallas.edu/ hamlen/hamlen-writingtips.pdf

[9]https://www.overleaf.com/latex/templates/springer-lecture-notes-in-computer-science/kzwwpvhwnvfj

Evaluation + Analysis (3 Pages)

Conduct experiments to benchmark your processing engine's throughput. Additionally, identify some bottlenecks (i.e., operations that take particularly long). Explain what the issues are, and propose some solutions. Provide references to support your claims.

Conclusion (1 Page)

Finalize the report with a summary, lessons learned, and issues you've faced. Additionally, dedicate a small paragraph to provide honest but constructive feedback for the exercise and how you think we could improve it. Please keep the feedback exclusively to the exercise's content, and use tiss if you wish to provide feedback on the general course structure and execution.

Theory Questions (1 Page)

Dedicate a page to answer the following:

- There are some glaring problems and needless redundancies in the architecture. Describe what they are, and sketch out an alternative system that alleviates the problems.
- Why can Storm not achieve exactly-once processing semantics?
- How does Trident address this limitation to support exactly-once processing?

Outcomes

The expected outcomes of this project are three-fold: (1) the actual project solution, (2) two presentations of the results, (3) your report.

Project Solution

The project has to be hosted on a git repository provided by the Distributed Systems Group – you will get instructions on applying for such a repository at the lab kickoff meeting. Every member of your team is assigned a separate Git account. *We will check who has contributed to the source code*, so please make sure you use your account when submitting code to the repository. Further, it is required to provide an easy-to-follow README that provides some details on the application.

For the submission (see below), you must package your services in Docker containers and create a docker-compose script, which starts your implemented solution with a **single** command. Projects requiring more than a single command will not be graded.

Presentations

There are two presentations with **mandatory** attendance. The first presentation is during the mid-term meetings and covers at least the tasks of Stage 1 (see below), the second presentation is during the final meetings and contains all your results. The actual dates are announced on TISS.

Each presentation consists of a (brief) slide part and a demo of your implementation. Think carefully about how you will demonstrate your implementation, as this will be part of the grading. You have 15 minutes per presentation. The 15 minutes will be strictly enforced, as we follow it up with a 10-minute Q&A. The first presentation is intended as a dry run to receive feedback. There, you present your current results, and the Q&A is less formal with room for discussion. The second presentation is graded, and the Q&A will be similar to an oral exam., i.e., you should demonstrate a profound understanding of your tech stack and the project's problem domain as a team and as individuals. The grade of the presentation is determined by your performance in the Q&A (75%) and the quality of your presentation (25%).

Report

Designate a first author responsible for coordinating and driving the writing process. However, every member must participate in writing the report. Include a separate markdown detailing how members contributed to the report (i.e., who wrote what (sub)sections, proofread, checked references for correctness, etc.). Place both files in a *report* subdirectory.

Grading

A maximum of 60 points are awarded in total for the project. Of this, up to 40 points are awarded for the implementation (taking into account both the quality and creativity of the solution as well as code quality), and 20 points are awarded for the presentations (taking into account content, quality of slides, presentation skills, and discussion). The report does not give any points. Instead, the report's grade acts as a multiplier (rounding up) to your points (min=0., max=1.). For example, if you were awarded 28 points for your project, 15 points for your presentation, and got 50%, on the report, your total project points are calculated as $\lceil (28 + 15) \times 0.5 \rceil = 22$.

A strict policy is applied regarding plagiarism. Plagiarism in the source code will lead to 0 points for the particular student who has implemented this part of the code. If more than one group member plagiarizes, this may lead to further penalties, i.e., 0 points for the implementation. Plagiarizing your report, will automatically lead to 0 points for the entire project for all team members.

Deadline

The hard deadline for the project is **January 15, 2024 18:00**. We will only consider commits **before** the deadline. Late submissions are not accepted.

Development Environment

You may develop and deploy the central server and the front end locally without cloud resources. However, we recommend deploying the application in a cloud environment. To this end, we recommend using the AWS free tier, which does not require you to apply for any student grant or similar. However, cloud deployment is not required and will not be taken into account for grading purposes.