The Meteor Project

Insightful, Efficient and Responsive App for Analyzing and Visualizing Hangzhou Metro

Chi Zhang (aka. Alex Chi) iskyzh@sjtu.edu.cn

Dec, 2019

Abstract

Meteor is an insightful, efficient and responsive app for analyzing Hangzhou metro. With records from automated fare collection system, Meteor helps metro passengers travel efficiently by visualizing per-station inflow and outflow data, and planning route based on segment pressure and historical travel time. It analyzes data efficiently with the help of SQLite and a task-based background scheduling system. And it exploits macOS features such as Touch Bar to provide a flawless data navigation experience. Meteor succeeded in maintaining a balance between analysis efficiency and analysis insightfulness. With Meteor, some interesting data patterns are observed.

1 Introduction

The aim of Meteor is to promote efficient travel in metro system. This paper is organized into three parts.

- **How does Meteor do analysis?**
  Section 2 discusses the implementation of Meteor in a bottom-to-top approach: the underlying database organization, the task scheduling system and the integration with macOS.
- **What analyses are done?**
  In section 3, analysis results are shown.
- **How fast and reliable is Meteor app?**
  Evaluation of the Meteor App is presented in section 4 to demonstrate that it provides efficient and reliable data analysis.

2 Implementation

A close integration with SQLite enables efficient data query and fine-grained data analysis in Meteor. And a task-based background scheduling system makes data analysis easy to implement, transparent to users and fault-tolerent. Meteor also supports Magic Trackpad and Touch Bar to provide a flawless data navigation experience.

2.1 Data Organization and Analysis

Storing data in database such as SQLite makes it efficient to filter, process and count data. Meteor exploits SQLite to provide efficient data anslysis.

In the database there're four tables: *dataset*, *flow-analysis*, *smart-travel* and *journal*. Details of these tables are discussed below.

dataset table

time int	lineID text	stationID text	deviceID text	status int	userID text	payType int

flow table

start_time int	enter_station_id int	exit_station_id int	time_block int	flow int

smart-travel table

start_time int	enter_station_id int	exit_station_id int	time_block int	flow_sum int	flow_n int

journal table

journal_id text	completed_at int

2.1.1 Reading Dataset

Meteor transforms data in `.csv` into records in database, and stores them in *dataset* table. All datetime strings are transformed to UNIX timestamp, therefore these records can be sorted and filtered by time efficiently. Meanwhile, an index is created to make filtering faster. This index builds on *time, stationID, lineID, status*.

2.1.2 Flow Analysis and Smart Travel Time

Meteor can estimate pressure of each metro segment by flow analysis. By pairing records of entering and exiting stations from a single user, Meteor estimates where the user should be in a 5-minute time window. Then number of users on a metro segment can be evaluated. This is so-called *flow analysis*. These data are cached in *flow-analysis* table.

Travel time estimation uses a 1-minute time window, and these data are cached in *smart-travel* table. The estimation depends on kNN (k nearest neighbours) algorithm. To make the process more efficient, departure time of each record is bucketed into a 1-minute window, which means only date, hour and minute are stored. Assume that ETA is only related to departure weekday, time, entry and exit station. Distance is defined as below, where I is entry station, O is exit station, D is departure time of one bucket and n is sample size of that bucket. Larger sample size means more credibility, thus reducing distance. $f(x, y)$ computes the shortest distance on map between station x and station y.

$$dist(X_i, X) = 10 \times f(I_i, I)^2 + 10 \times f(O_i, O)^2 + (D_i - D)^2 + \frac{5}{n_i^2}$$

This simple model gives reasonable travel time. As data are not sufficient to evaluate the model, in this paper, accuracy of this model won't be discussed.

2.1.3 Journal

The date and time when each task completes is stored in *journal* table. Each time when a task is to be scheduled, the scheduler will check if a task has been done before by querying *journal* table.

2.1.4 Route Planning

Meteor uses breadth-first search to plan the shortest route. After obtaining a sequence of station ID, Meteor will then find more detailed information, for example, where to transfer.

To obtain transfer data, all stations along one metro line should be known in advance. Stations on line A and line C can be found by obtaining the shortest route between the starting station and the terminal. And there're some special cases. Line B has a Y-shape. There're two terminal stations (station 33, station 27) on line B in one direction, and the shortest route between starting (0) and terminal station (33 or 27) requires a transfer to line A. Therefore, Meteor temporarily removes the connection between station 79 and 80 before finding the stations along line B, and split 0-33, 0-27 into two lines B1 and B2.

Therefore, if a station S in route isn't on the current metro line, it can be deduced that a passenger should transfer to current metro line at the station before S.

Combining flow analysis and smart travel time, Meteor shows estimated arrival time at each station, and calculates crowded rate of each segment. This functionality is called *Meteor Adviser*.

2.2 Task Scheduling

All computational works are organized in tasks. A task is a thread emitting signals such as *success*, *progress* and *message*, which can be managed by task scheduling system. This abstraction simplifies the way to implement scheduling system and retains developer's freedom to run their own task on Meteor.

A custom task-scheduling system is implemented in Meteor to make data loading and processing fast, efficient, seamless and fault-tolerent. Examples of tasks include *initialize database*, *flow analysis of Jan 9*, and *read dataset of Jan 10*.

The scheduler runs in a standalone thread. It acts as a bridge between tasks and GUI. Inside the scheduler is a task queue, which is indeed a double-ended queue. GUI thread requests the scheduler to run a task, and receive signals on currently-running tasks and their progress. The scheduler pushes newly-scheduled task to the end of the queue and its dependencies to the front of the queue. When a task completes (the *success* signal is emitted from the task), the scheduler will notify GUI thread, clean up previous task object, pick the next task in queue and run it. Only one task runs at a time.

Lazy loading technique is employed in the scheduling system, which means tasks will be executed on-demand. For example, if a user just requires flow analysis of Jan 9, only these tasks will be executed: *initialize database*, *read dataset of Jan 9*, *flow analysis of Jan 9* and *query data*. The system automatically resolves dependencies of these tasks, and schedules them in order in a background thread.

Tasks can be journaled, which means they will be executed only once on one's computer. For example, if a user requires inflow and outflow data from Jan 9 after flow analysis of that day, only *query data* task will be executed, as database has already been intialized and dataset is already in database.

All tasks are fault-tolerent as they clean up possible garbage data in database before executing. This technique is commonly referred as *rollback* in database systems. For example, if the app crashes when reading dataset, after restarting, the *reading dataset* task is not journaled. Therefore, it is in either of the two states: it never runs, or it crashed. By cleaning up all records with date Jan 9, the database stays consistent.

Meanwhile, all tasks support gracefully termination. Each task periodically check the atomic bool variable *cancel*. If it is true, the task will stop, and the program may shutdown gracefully.

2.3 macOS Integration

There're some exclusive features in macOS, such as Magic Trackpad with gestures support, and the Touch Bar which makes controls available at your fingertip. The former one is compatible with most GUI programming frameworks, and the latter one requires extra engineering effort.

Integrating with Magic Trackpad for flexible scroll is trivial as it triggers `mouseWheel` event in Qt.

However, much more effort is required to exploit the Touch Bar. It requires a building system with Objective-C target support, and a mechanism to call Objective-C functions from C++ code.

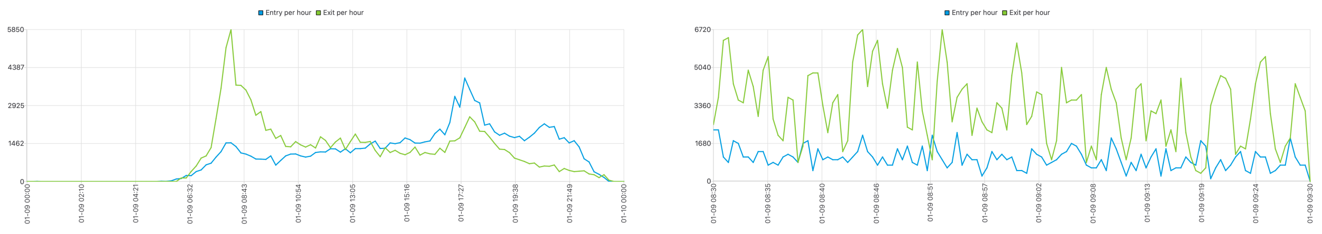
CMake is used to build the Meteor project. The building system recognizes Objective-C source files, which ends with `.mm`. In `touchbar.mm`, it's possible to call AppKit framework of macOS and implement `TouchbarProvider`.

Multiple global functions are decleared in common headers, hence enabling two-way communication between macOS APIs and Qt. This makes it possible to check if a button is pressed on Touch Bar in `MainWindow`, and to set the value of a slider on Touch Bar from `MainWindow`.

3 Analysis Result

3.1 Inflow and Outflow

From the chart we may easily observe the data pattern.



(a) Daily Data (b) Hourly Data

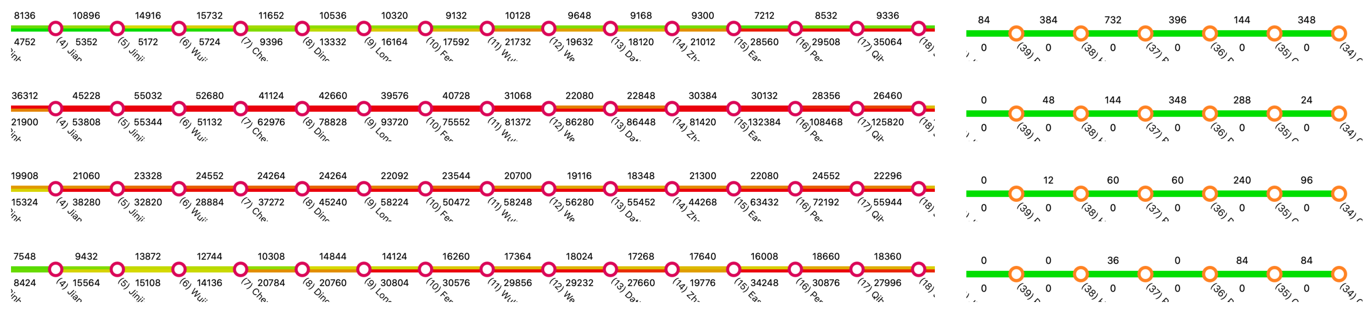
Figure 1: Inflow and outflow for Fengqi Road Station at 2019-01-10

As shown in Figure 1 (a), 8:30am and 6pm is generally peak hour for all stations. If a station has a larger outflow in the morning, it will have a larger inflow in the evening.

And in Figure 1 (b), for each station, there're regular peaks in outflow data. With this information we may deduce arrival time of each train.

3.2 Flow Analysis for Segment Pressure

With flow analysis, some data patterns are observed.



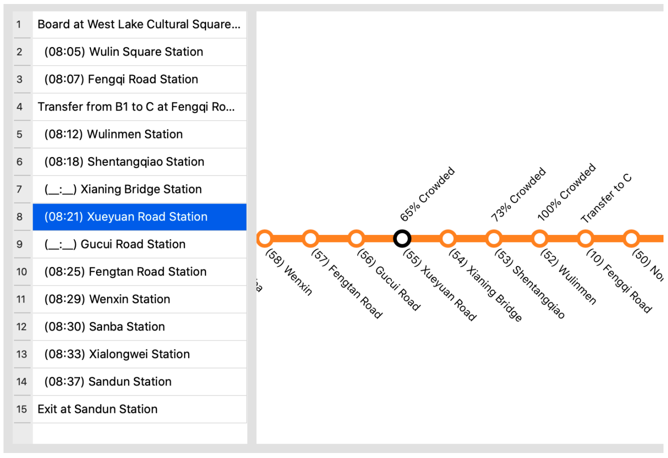
Left: Line 1 in 2019-01-09, (top-to-bottom) 7:00am, 8:00am, 9:00am, 10:00am
Right: Line 2 in 2019-01-12, (top-to-bottom) 11:56pm, 11:58pm, 12:00am, 12:02am
Upper lane for left to right pressure, lower lane for right to left.

Figure 2: Flow data for Line 1 and Line 2

As shown in Figure 2 (a), on weekdays, segment volume reaches it peak at 5pm and 8am. As shown in Figure 2 (b), significant flow appears at terminal stations at the end of service hours.

3.3 Route Planning and Smart Travel Time

Meteor will show where to transfer in route details. And it draws a map of metro lines with real world station names to make route more readable. With Meteor advisor, users may find estimated arrival time of each station in route. Generally the kNN model will output reasonable ETA. And crowded rate is shown in map. This can be seen in Figure 3.



1	Board at West Lake Cultural Square...
2	(08:05) Wulin Square Station
3	(08:07) Fengqi Road Station
4	Transfer from B1 to C at Fengqi Ro...
5	(08:12) Wulinmen Station
6	(08:18) Shentangqiao Station
7	(_:_) Xianing Bridge Station
8	(08:21) Xueyuan Road Station
9	(_:_) Gucui Road Station
10	(08:25) Fengtan Road Station
11	(08:29) Wenxin Station
12	(08:30) Sanba Station
13	(08:33) Xialongwei Station
14	(08:37) Sandun Station
15	Exit at Sandun Station

Figure 3: Route planning with Meteor Adviser enabled

4 Evaluation and Discussion

Evaluation is done on MacBook Pro (15-inch, 2017). Build Meteor with CMake Release mode, then run the program. Set Meteor to use in-memory database in wizard. Then, run benchmark. Run the benchmark for three times to obtain the evaluation result. Result is shown in Table 1.

Table 1: Evaluation Result

Task	Number of Tests per Epoch	Average Time (ms)
Read Dataset (1 day)	2	28301.17
Entry / Exit Query (Daily)	10	3820.76
Entry / Exit Query (Hourly)	10	515.06
Flow Analysis (1 day)	1	13809.67
Query Segment Pressure	10	280.83
Plan Route	50	2.91
Smart Travel Analysis (1 day)	1	16635.33
Smart Travel Predict	10	1766.03

After heavy data processing and analysis (e.g. *read dataset*, *flow analysis*, *smart travel analysis*), query tasks perform very fast. Therefore, Meteor succeeded in maintaining a balance between analysis efficiency and analysis insightfulness. With several optimization and caching techniques, Meteor exploits full potential of SQLite. In a nutshell, Meteor provides a variety of data analysis perspectives while keeping the analysis process relatively fast.

Meteor is licensed under GNU General Public License 3. This project will be open-sourced after the course ends. The Qt project laid a solid foundation for this project. And also thank ZQ Zhao for inspiring me use an in-memory database instead of a persist-to-disk one, hence providing better performance.