[PD-02-005] Design, Development, Testing, and Deployment of GIS Applications

Abstract

A systems development life cycle (SDLC) defines and guides the activities and milestones in the design, development, testing, and de ployment of software applications & information systems. Various choices of SDLC are available for different types of software applications & information systems and compositions of development teams and stakeholders. While the choice of an SDLC for building geographic information system (GIS) applications is similar to that of other types of software applications, critical decisions in each phase of the GIS development life cycle (GiSDLC) should take into account essential questions concern ing the storage, access, and analysis of (geo)spatial data for the target application. This article aims to introduce various considerations in the GiSDLC, from the perspectives of handling (geo)spatial data. The article first introduces several (geo)spatial processes and types as well as various modalities of GIS applications. Then the article gives a brief introduction to an SDLC, including explaining the role of (geo)spatial data in the SDLC. Finally, the article uses two existing real-world applications as an example to highlight critical considerations in the GiSDLC.

Keywords: AGILE, APIs, software, systems development life cycle, Waterfall, web mapping

Author & citation

Chiang, Y-Y. and Lin, Y. (2020). Design, Development, Testing, and Deployment of GIS Applications. The Geographic Information Science & Technology Body of Knowledge (4th Quarter 2020 Edition), John P. Wilson (Ed.). DOI: 10.22224/gistbok/2020.4.2.

Explanation

1. Introduction

Geographic information systems (GIS) refer to the information systems that manipulate (geo)spatial data. The data manipulation process can range from simple data aggregation to sophisticated spatial analysis and machine learning methods. An example of a data aggregation process is computing the daily average temperature of every location on a map. A spatial analysis method employs principles in spatial statistics to model data in space to understand the data generation process or make predictions, e.g., kriging (Cressie 1990) and geographically weighted regression (Brunsdon et al. 1996). A machine learning method, similar to spatial analysis methods, exploits (geo)spatial properties in the data to either make inference or prediction using existing data for unseen data. In this article, these data manipulation processes that use (geo)spatial data are called (geo)spatial

processes.

Traditional GIS (software) applications are often generic, all-purpose GISs, which support a variety of (geo)spatial processes for analyzing and visualizing (geo)spatial data in various formats on a local computer. Recent advances in computing power and software development tools have enabled a broad array of GIS applications, including powerful generic GIS applications and customized applications for specic missions (e.g., visualizing traffic data). In general, GIS applications can be categorized as either desktop or cloud-based applications based on the characteristics of their computing resources (e.g., data storage, computational power, network access). Desktop applications, such as Esri's ArcGIS Desktop and QGIS often do not require an internet connection but can only handle a small amount of data because of a single computer's limited computational power and storage. In contrast, cloud-based applications, such as Esri's ArcGIS Online can process large data (often on a large computer cluster) but require access to the network (either the internet or a private enterprise network).

In addition to the distinction in computing resources, GIS applications can have a variety of modalities based on the design of their human-computer interaction (HCI) strategies, such as mobile GIS applications, Web GIS applications, and online GIS APIs (application programming interfaces). Mobile GIS applications are software applications running on mobile devices and, most importantly, can exploit location based services (LBS) for solving a practical problem (e.g., recording moving trajectories). For example, Esri Collector is a mobile GIS application that enables efficient data collection using a mobile device. Web GIS applications run within a web browser, either on a mobile or desktop device. The main advantage of Web GIS applications is that web browsers ubiquitously exist on various types of devices, and hence the users do not have to install additional software applications. Web GIS applications often require a backend cloud GIS. A Web GIS supported by a cloud GIS can support a wide range of functionalities, such as visualizing existing or user-provided map data and performing complex (geo)spatial processes. An example of Web GIS applications is CARTO, which enables the users to customize datasets and publish interactive maps on the Web. Google Maps is both a mobile & Web GIS application that displays location information, including traffic, road networks, and points of interest, either on mobile devices or on the Web. Online GIS APIs are computing processes running on a server. They can be accessed on the internet using specific protocols (e.g., RESTful services conformed to the REpresentational State Transfer (REST) software architecture). For example, the University of Southern California's Spatial Sciences Institute provides an air quality prediction API, and Google makes their geocoding services available as an online API.

GIS applications can also be distinguished by their supported (geo)spatial data types, which often determine the capability of the application's (geo)spatial processes and the modality of the application. For example, suppose a GIS application requires producing weekly spatially aggregated air quality values from point locations. Then the best modality is a cloud (or server) based application using a spatial database. If the application involves more sophisticated (geo)spatial processes than data aggregation, the software architecture on the cloud should be able to host (geo)spatial programming libraries or machine learning frameworks, e.g., Tensor flow (Lin et al. 2018)

Two common (geo)spatial data types are the vector data type and raster data type. Vector data represent the geometry of (geo)spatial things using points, lines, and polygons. Also,

vector data often come with a tabular component describing the geometries' metadata (attributes). Popular vector formats include the Esri shapefile (Esri 1998), GeoJSON, and the Open Geospatial Consortium (OGC) Well-Known Text, Well-Known Binary, and KML (Keyhole Markup Language). The Esri shapefile stores object geometry and their attributes (e.g., location names, addresses). GeoJSON is a lightweight vector format mostly for web GIS and other GIS applications that handle small datasets. Well-Known Text and Well-Known Binary are common data formats for spatial databases (e.g., PostGIS) originated from the OGC. Their definitions are included in the ISO/IEC 13249-3:2016 standard. KML (currently maintained by the OGC) contains the object geometries and attributes and how these geometries should be visualized in an application (e.g., Google Earth and Google Maps Android Software Development Kit). Vector data allow a small file size and precise point, skeleton, and boundary representations of (geo)spatial things.

In contrast to vector data that explicitly store geometry information, raster data utilize a grid structure to represent spatial things using a continuous surface over the space. Each cell in the grid covers a piece of the space and can contain a set of variables as the cell's metadata. The most common raster geographic data type is imagery, such as satellite imagery and aerial photos. Standard raster formats include GeoTIFF and NetCDF (Network Common Data Form). GeoTIFF is a metadata standard for describing georeferenced images in the Tagged Image File Format (TIFF). NetCDF supports multi-dimensional data and includes a set of formats and libraries to represent and access the multi- dimensional data. NetCDF is commonly used in the geoscience community to store various spatiotemporal phenomena. Raster data have the benefit that its data representation is not limited to a predefined boundary compared with vector data. Most image processing libraries (e.g., OpenCV) can handle raster data by treating each raster cell as image pixels.

In sum, the modality of GIS applications has a wide range. The supported data types and the (geo)spatial processes can vary from simple point data visualization to sophisticated imagery recognition. The broad spectrum of modalities and capabilities makes it challenging to use a single methodology to design, develop, test, and deploy GIS applications (i.e., the systems development life cycle, SDLC, for GIS applications). Specically, SDLCs should include considerations of the instrument, problem, domain, and people (stakeholders) of a GIS application. Nonetheless, all GIS applications need to consume, process, and produce (geo)spatial data. Hence, this entry focuses on critical decisions in each phase of the GIS development life cycle (GiSDLC) concerning the requirements of storage, access, and analysis of (geo)spatial data for the target application. The remainder of this article aims to introduce various considerations in the GiSDLC, including application design, development, testing, and deployment, from the perspectives of data storage, access, and analysis. Section 2 gives a brief introduction to SDLC, including explaining the role of (geo)spatial data in each SDLC phase. Section 3 uses two existing real-world applications as an example to highlight critical considerations in each step of the GiSDLC. Section 4 provides an overall summary.

2. Systems Development Life Cycle

A systems development life cycle (SDLC) defines the activities and milestones in developing information systems. In general, a SDLC can include five to ten phases. For example, a seven-phase SDLC includes the phases of problem identication, plan, design, development,

testing, deployment, and maintenance. To maintain a concise discussion, without loss of generality, this entry focuses on a five-phase SDLC, including the phases of design, development, testing, deployment, and maintenance (where the phases of problem identication, plan, and design are combined into a single phase of design). In addition to the SDLC phases, an SDLC methodology or model guides the execution of these phases. While the choice of an SDLC for GIS applications is similar to that of other software applications, each phase of the GIS development life cycle (GiSDLC) should pay additional attention to the required (geo)spatial data and processes. This section first explain two representative SDLC methodologies (i.e., Waterfall and Agile model) that can be used for GiSDLC and then describes each SDLC phase focusing on (geo)spatial data and processes.

2.1 SDLC Methodologies

A variety of SDLCs has been proposed in the software engineering community to different organizations, development teams, and projects, such as Waterfall, Agile, Spiral, and V-shaped. Traditional methods include the Waterfall model, a sequential methodology that requires completing all activities in each phase before moving to the next phase (McCormick 2012), and the Agile model, an iterative methodology that contains many iterations, each completing a subset of activities for each phase (Edeki 2015). Other methods are often used in specific scenarios; for example, the V-shaped model is mainly for the small and mid-sized projects, where requirements are strictly predefined. This section introduces the process of Waterfall and Agile model and compares their advantages and disadvantages.

In the Waterfall model, each phase must complete in full before the next phase begins. A review process at the end of each phase determines if the project is on the correct path. The Waterfall model provides an easy way to manage the development project since each phase has a specific deliverable and a review
process. Therefore, it usually works well for projects where the requirements can be clearly defined and well understood. However, one weakness is that once a project is in the testing stage, it is difficult to change something that was not well-thought-out in the previous stages. There is no working software or demo produced until late in the life cycle, so the Waterfall method is not suitable for long-term and complex projects. Another drawback of the Waterfall model is that very few stakeholder interactions are involved in the entire life cycle, meaning that the product can only be demonstrated to the stakeholders when ready at the end of the cycle (Mahalakshmi and Sundararajan 2013).

The Agile method is a type of incremental software development with rapid and iterative cycles. Specifically, the entire application development process is divided into several (often many) small cycles. The development team performs every stage in the phases of design, development, testing, and deployment of the application in each cycle. At the end of each cycle, the team needs to deliver an agreed-to subset of the product, and the stakeholders can see a work in progress. One strength of the Agile model is that the stakeholders actively participate throughout the project. This way, the development team has opportunities to truly understand the stakeholders' vision with their reviews after each cycle. The Agile model also allows for changes in the next iteration, which provides an opportunity to refine and re-prioritize the overall product constantly. However, having constant interactions with stakeholders might result in complicated communication management. A lack of a final vision or making too often changes might disorganize the project. Also, there will be weak documentation since the interactions with the stakeholders

are mostly verbal due to the nature of the frequent communication.

2.2 SDLC Concerning (Geo)Spatial Data

In the design phase, the development team conducts the requirement analysis with the stakeholders, selects software tools (e.g., programming languages, software frameworks & architecture, software libraries, and system architectures), and determines the computing environment (e.g., desktop vs. cloud-based applications). In the case of GIS applications, critical decisions should consider three essential questions concerning the required (geo)spatial data and processes for the target application:

- What are the required types and sizes of (geo)spatial data (as the input and output of the system)?
- What are the required (geo)spatial processes?
- What is the desired response time for each (geo)spatial process?

These three questions are inter-related. The required (geo)spatial data types often determine the applicable (geo)spatial processes and the potential data sizes, which have a significant impact on the process response time. Answering these questions helps identify a suitable software architecture and computing environment.

In the development phase, the development team builds the applications according to the requirements outlined in the design phase. A common development strategy is to independently develop each software module using the predefined input, process, and output from the design phase to build multiple modules in parallel and test them separately in the next phase. For example, a preferred way of developing GIS applications is to separate the data preprocessing and preparation module from the analysis module. Concerning a GIS application for air quality prediction, the prediction module can predefine the input format and coordinate system and leave all data collection, cleaning, and transformation to the preprocessing module.

In the testing phase, the development team uses a variety of scenarios to test the built application. For example, the testing phase would include processing large amounts of data of different kinds on various operating systems or having users intentionally make mistakes when operating the application. For GIS applications, the test data should include (geo)spatial data from various coordinate systems and scales to ensure correct transformation and integration of (geo)spatial data from heterogeneous sources. The testing phase should show that the built application can handle typical and unexpected scenarios (e.g., errors in the input data).

In the deployment phase, the development team integrates the built application with the required software frameworks, libraries, and architectures in a production environment and provides application access to the stakeholders. The development team usually creates an installation package for the built application on the target operating system (e.g., Windows Installer for Microsoft Windows) and sometimes needs to release the installation package to an application marketplace (e.g., Apple App Store).

After the deployment of an application, the development team needs to ensure that the application continues to be functional. For example, for a cloud-based application, the server should be responsive without any downtime. Also, in each phase, the development

team needs to create application documentation to facilitate the development process, enable future maintenance, and provide the operation manual. Additional operation needs might include releasing the documentation to the public so that a wide range of users (like data scientists and geoscientists) can access and use the applications as well as providing technical support (see Yang et al. 2019).

3. Case Study: Archived Traffic Data Management System

The Archived Traffic Data Management System (ADMS) (Anastasiou et al. 2019) is a big transportation data warehouse developed by the University of Southern California's Integrated Media Systems Center (IMSC), with a partnership with Los Angeles Metropolitan Transportation Authority (LA Metro) and METRANS. ADMS fuses and analyzes very large-scale and high-resolution (both spatial and temporal) traffic sensor data from a variety of transportation authorities in Southern California, including the California Department of Transportation (Caltrans), Los Angeles Department of Transportation (LADOT), California Highway Patrol (CHP), and Long Beach Transit (LBT). This dataset includes both historical and real-time data. The real-time data have an update rate as high as every 30 seconds for freeway and arterial traffic sensors (14,500 loop-detectors) covering 4,300 miles, 2,000 bus and train automatic vehicle location (AVL). The real-time data also include incidents, such as accidents, traffic hazards, road closures reported (approximately 400 per day) by the Los Angeles Police Department (LAPD) and CHP, and ramp meters. ADMS has been continuously collecting and archiving datasets for the past nine years. ADMS, with 15GB annual growth, is the most massive traffic sensor data warehouse built so far in Southern California. This section explains the applications on ADMS and discusses the SDLC method used in the overall ADMS development process.

3.1 Applications on ADMS

ADMS is a multi-layer architecture running on the IMSC cloud platform (Figure 1). The multi-layer architecture includes the data collection layer, data management layer, offline & online analytics layer, learning & intelligence layer, service layer, and presentation layer. Together, the multi-layer architecture supports various types of applications that require processing (geo)spatial data. For example, IMSC has developed (geo)spatial data-driven approaches and applications for traffic forecasting (Li et al. 2018) and performance reliability estimates of public vehicles (Nguyen et al. 2018) on top of the AMDS infrastructure. Both applications reside in the learning & intelligence layer and rely on data cleaning and transformation modules in other layers to access the required input data and prepare the data in a specfiic format. ADMS's multi-layer architecture is flexible and easy to maintain since every layer has an individual predefined structured input and output format. Each layer contains modularized components that can be updated quickly when new technologies are available without rebuilding the entire infrastructure.

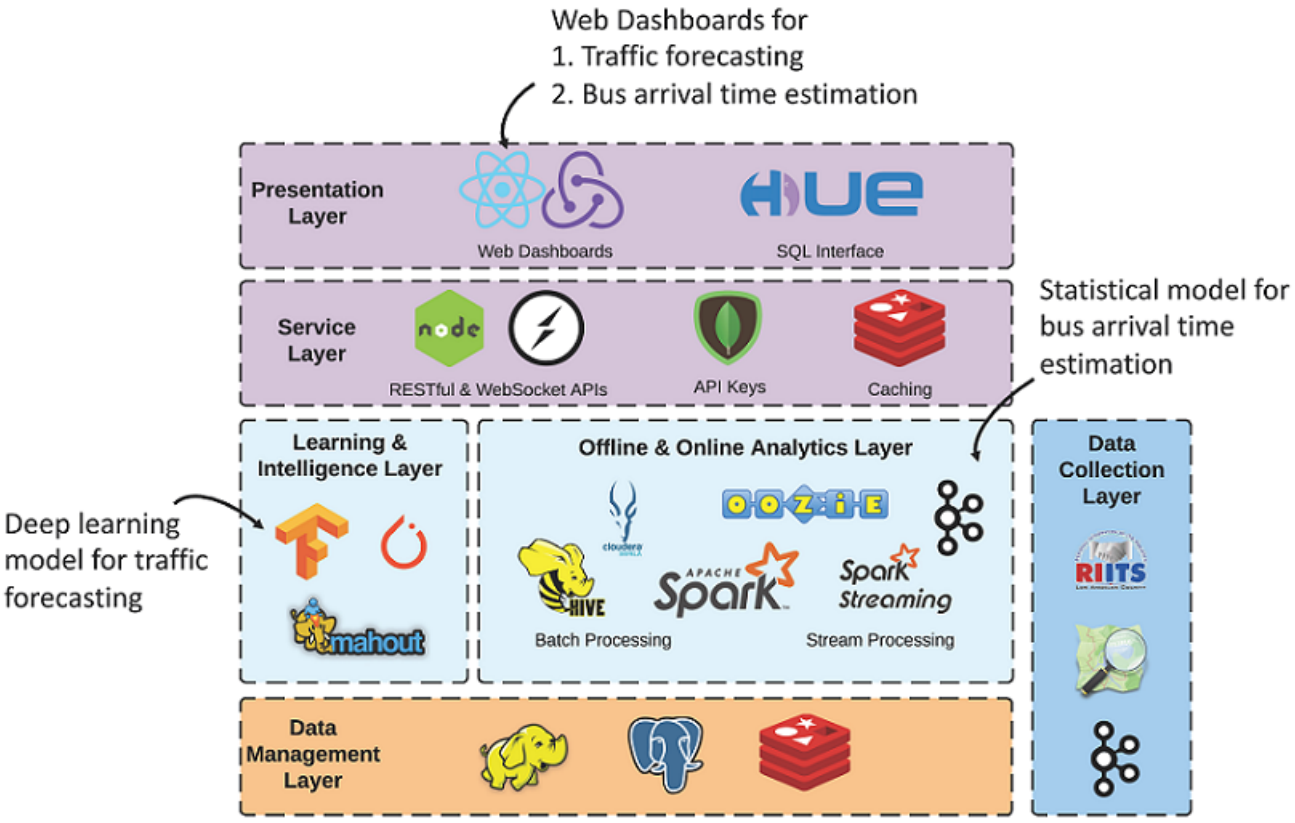

Figure 1. The multi-layer architecture in ADMS (Archived Traffic Data Management System). Source: authors.

The traffic forecasting application (Li et al. 2018) is a novel graph convolutional neural network (GCRNN) that models the complex temporal dependency and topological dependency of traffic flow for traffic forecasting. The application can forecast traffic at various spatial (e.g., individual regions, road segments, or sensors) and temporal (e.g., next 5 minutes and 30 minutes) resolutions. During the forecasting time, the application consumes real-time traffic updates from the message queue from the offline & online analytics layer, which ensures a continuous data feed for hourly forecasting. Then a module in the data management layer stores the forecasting results. An API in the service layer consumes the stored forecasting results and feeds the results to a Web GIS dashboard in the presentation layer. The stakeholders can access the Web GIS dashboard and click on street segments to visualize the traffic forecasting results.

The bus arrival time estimates application (Nguyen et al. 2018) uses large amounts of historical bus trajectory datasets stored in both HDFS and PostGIS in the data management layer. The application utilizes the offline parallel processing capability in the analytics layer to generate historical arrival time statistics for individual bus routes to estimate the delay for each bus stop at different times of the day. Like the traffic forecasting application, the estimation results are stored in the data management layer with an API in the service layer to provide public access from a Web GIS dashboard. In addition to the arrival time estimates, the Web GIS dashboard also displays real-time bus locations by consuming the bus location API in the service layer.

Both applications require access to traffic data offline and in real-time. The traffic

forecasting application requires a machine learning framework, while the bus performance estimation application relies on a simple data aggregation process on multiple-terabyte of historical data. In building these applications, the
development team builds and tests individual modules on separate layers independently using the predefined input, module process, and output, which enables quick prototyping and the detection of potential errors.

3.2 SDLC of ADMS

The SDLC of ADMS generally follows the Agile model in which the development process includes several iterations. In the first iteration, the development team builds each layer's initial functionality to ensure that the data pipeline from the data collection layer to the presentation layer is functional to support an end-to-end application. In the next iteration, the development team adds new functionalities to each layer to support sophisticated applications. In each iteration, the development team builds each layer sequentially. Here, we briefly discuss the ADMS SDLC strategy.

In the first iteration, the goal is to have a fully functional ADMS but handle only a few critical data feeds that can support bus arrival time estimates. The determined requirements include 1) collecting the LA Metro data feeds for real-time vehicle speed and volumes from thousands of traffic sensors and bus locations from hundreds of buses, and 2) storing the collected data according to their usage types. The recently collected data (e.g., months) require frequent spatial operations (e.g., spatial join). The near real-time collected data (e.g., minutes) require fast data access. The long-term historical data require data aggregation and filtering operations that can handle large datasets.

After the design phase, the team implements the data collection layer with a few critical modulized adaptors. The modulized adaptor implementation allows the system to easily accommodate new data feeds (e.g., weather information in neighborhoods) with additional adaptors in the next Agile iteration. The output of this data collection layer is the data streams from various data sources. Then the development team follows the requirements to implement the data management layer, including specialized data storage systems for accessing 1) recent data requiring robust spatial operations (PostGIS), 2) near-real-time data requiring fast data access using in-memory data store (redis), and 3) all historical data requiring distributed computing for data aggregation and filtering (Hadoop Distributed File System, HDFS). Next, the development team builds the basic framework for the service layer, learning & intelligence layer, and the presentation layer to accommodate end-to-end application deployment. The service layer includes several data access APIs for consuming data in ADMS (including results from the applications deployed in the learning & intelligence layer, e.g., bus arrival time estimation). The presentation layer contains Web dashboards and maps to visualize and disseminate ADMS data and output from the machine learning models.

The development team uses bus arrival time estimate to test the end-to-end data pipeline and Web visualization in the testing phase. In particular, the test phase verifies the implementation has fulfilled the requirements determined in the design phase. In the deployment phase, the development team opens the application to the general public and generates user manual and tutorial videos. In the maintenance phase, the development team hosts weekly office hours to help resolve user issues. The development team also installs software to monitor the status of ADMS and ensure ADMS has a near 100% uptime

to consume real-time data and generate near real-time results continuously.

In the second iteration, the development team repeats the same SDLC as in the first iteration with additional considerations to design, develop, test, deploy, and maintain the traffic forecasting application. For example, the second iteration requirement includes the support of an advanced machine learning framework in the learning & intelligence layer. Overall, the development of ADMS and benefits from using the AGILE model so that the stakeholders can review an end-to-end application after each iteration. During each iteration, the development team can add additional functionality (computing modules) in each ADMS architecture layer.

4. Summary

This article introduced various types of GIS applications and the considerations in the GIS development life cycle, from the perspectives of storing, accessing, and analyzing (geo)spatial data. The article also showcased two existing real-world applications to highlight critical considerations in each step of the GiSDLC.

References

Anastasiou, C. J. Lin, C. He, Y.-Y. Chiang, and C. Shahabi. (2019). Admsv2: A modern architecture for transportation data management and analysis. In ACM SIGSPATIAL International Workshop on Advances on Resilient and Intelligent Cities, 2019.

Brunsdon, C., Fotheringham, A.S., & Charlton, M.E. (1996). Geographically Weighted Regression: A Method for Exploring Spatial Nonstationarity. Geographical Analysis, 28(4), 281-298.

Clarke, K. C. (2011). Getting Started with Geographic Information Systems (5th Edition). Pearson Prentice Hall.

Cressie, N. (1990). The origins of kriging. Mathematical Geology, 22(3):239-252.

Edeki, C. (2015) Agile software development methodology. European Journal of Mathematics and Computer Science, 2(1).

ESRI. (1998). ESRI Shapefile Technical Description. An ESRI White Paper - July 1998.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2018). Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In International Conference on Learning Representations, 2018.

Lin, Y., Mago, N., Gao, Y. Li, Y., Chiang, Y.-Y., Shahabi, C., and Ambite, J. L. (2018). Exploiting spatiotemporal patterns for accurate air quality forecasting using deep learning. SIGSPATIAL '18: Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 359-368.

Mahalakshmi, M. and Sundararajan, M. (2013). Traditional SDLE vs Scrum Methodology: a

Comparative Study. International Journal of Emerging Technology and Advanced Engineering, 3(6):192-196.

McCormick, M. (2012). Waterfall vs. Agile Methodology. MPCS, Inc.

Nguyen, K., Yang, J., Lin, Y., Lin, J., Chiang, Y.-Y., and Shahabi, C. (2018). Los Angeles metro bus data analysis using gps trajectory and schedule data (demo paper). In Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 560- 563.

Yang, C., M. Yu, Y. Li, F. Hu, Y. Jiang, Q. Liu, D. Sha, M. Xu, and J. Gu. (2019). Big earth data analytics: A survey. Big Earth Data, 3(2):83-107.