Name: Donna Wang
Class: 2025
Advisors: Wyatt Lloyd, Nicolaas Kaashoek

Title: Improving maps & navigation application using serverless computing

Motivation and Goal.

High latencies in map and navigation apps are a major pain point for users and one of the primary drivers of complaints. A sudden spike in latency often leads to increased complaints on application-specific support forums, such as Google's, and social media platforms like Reddit, underscoring the importance of the issue. For instance, in a recent Reddit thread, multiple users reported experiencing lags of up to 10 seconds while using Google Maps while driving.[1] In response, another user suggested using offline maps as a potential solution, noting that turning off data resulted in faster performance. They believed this improvement was due to reducing the amount of data transmitted, suggesting a connection between latency and the volume of data sent between users and servers. The goal of my project is to improve the latency of map applications by leveraging serverless computing on the edge.

Problem Background and Related Work.

In the paper "Geospatial Serverless Computing: Architectures, Tools, and Future Directions" by Bebortta et al., the authors discuss various scalable serverless frameworks for managing geospatial big data and propose a framework within the context of a Cloud Geographic Information System (GIS) platform. This framework integrates serverless architecture to reduce timing constraints and enhance the performance of geospatial data processing, particularly for high-dimensional hyperspectral data. While the paper relates to maps and navigation, its focus is primarily on serverless computing in big data analytics, whereas my project aims to directly leverage serverless computing for real-time map applications.

In another paper titled "Real-time GPS Tracking Using Serverless Architecture and ARM Processor" by Sundar Anand et al., the authors build a GPS tracker for vehicles using a serverless architecture.[2] Their goal is to utilize cloud computing to improve the efficiency of data storage and analysis as opposed to more traditional methods without. In this project, user location data is stored in the cloud, as most stateful applications do. My project, however, seeks to utilize storage at the edge using Radical, which not only provides the benefit of low latency at the edge but also provides the guarantees offered by the datacenter. This approach allows stateful applications to leverage the benefits of serverless computing without requiring it to be stateless.

[1] https://www.reddit.com/r/GoogleMaps/comments/1cb1h7s/google_maps_acting_up_super_slow_lately/
[2] https://ieeexplore-ieee-org.ezproxy.princeton.edu/abstract/document/8711273

Approach.

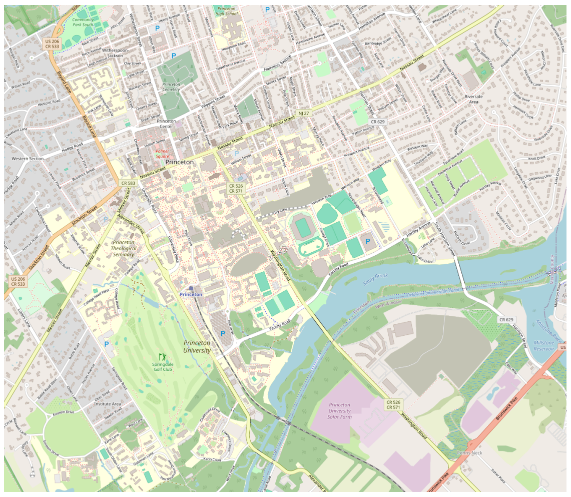
My project will implement a version of a real time maps and navigation application such as Google Maps on top of Radical, a serverless computing framework. Radical, which is a framework that enables applications to run at the edge without sacrificing storage guarantees.[3] Applications running at the edge have the benefit of being closer to the users and thus resulting in lower latencies. However, applications run at the edge lack many of the guarantees that are available at the data centers. Radical addresses this gap by leveraging the lower latencies available at the edge while maintaining the consistency guaranteed at the datacenter. My goal is to leverage the properties of Radical to either decrease the latency of routing algorithms

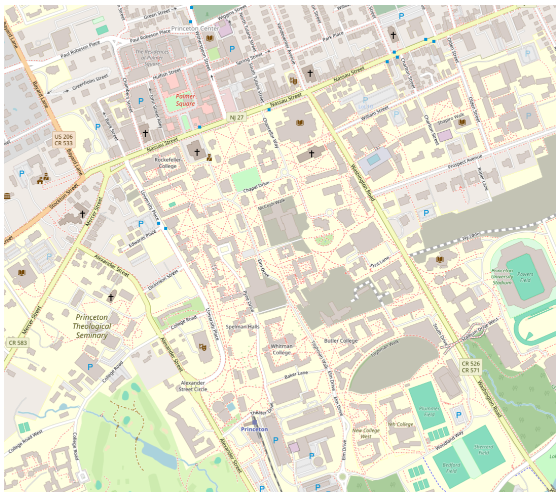
Plan.

Data set.

The data set to be used in this project will be acquired from Open Street Map. Specifically, I will be focusing on Princeton University's main campus. There are two sets of data I will be using: one which involves the majority of the campus and its surrounding area, and the other is a smaller map containing only the main campus (excluding Graduate buildings). The larger map is 7,299 KB in size and the smaller map is 1,596 KB in size.

The map files are in XML format. Each file contains the following data types: nodes, ways, and relations. Nodes represent a specific geolocation on earth with latitude and longitude. Ways contain a series of nodes that make up roads, highways, streets, etc. Relations define logical or geographical relationships between the different data objects.





[3] Radical paper draft

Princeton campus and surrounding areas (left) and Princeton main campus (right).

Data Processing

There are a couple things I need to achieve in regards to data processing:

1. Extract relevant information from OSM XML files.
2. Organize data in such a way that will a) allow me to construct a graphical representation of the geological data, and b) can be uploaded to a database that reduces query complexity.

To parse the OSM XML files, I used PyOsmium. I chose to keep and reuse the id fields from the OSM XML files as object identifiers. For nodes, I also extracted longitude and latitude information. Then, in order to decrease the complexity of function calls later on, I created adjacency lists for the 'node' objects. I did this by iterating over each 'way' object. Then, using the fact that the nodes contained in the way objects are ordered, I append the left and right nodes of each node in the way to its adjacency list. Currently, this setup is sufficient for constructing a graph network representation of the geological data. However, as the project progresses, it's likely that I will need additional information that is provided in the XML files, and this may change how I process and organize my information.

Database Setup

Radical requires that the data is stored in a consistent, durable, and high-capacity storage system. Dynamodb satisfies this requirement, and since it was also previously used to test Radical, I decided to start by uploading my data to Dynamodb. Since Dynamodb is non-relational, my data schema is very simple: the keys of my data are just the object id's that were extracted from the OSM XML files, and the values are the remaining information per object broken down into various attributes such as 'longitude', 'latitude', and 'adjacency_list.' After inserting my data, there are 6199 node objects, 927 way objects, and 19 relation objects.

At a later point in the project, I might also want to run my database using PostgreSql. There are two primary benefits of making this change: a) I can test the functionality of Radical on a relational database, and b) I can align the project more closely with OSM and other common practices in the field regarding map data. For OSM data, since it is designed with PostgreSQL in mind, using PostgreSQL would reduce the manual data processing required. Further, in practice, map data is also often stored in relational databases such as PostgreSQL, due to its integration with PostGIS, which provides geospatial capabilities. Therefore, implementing my project with this database could make it more valuable and compatible with common practices in the field.

Map Application Implementation

The implementation of the map application will involve basic map function calls to simulate real-world functionality. In order to stay consistent with Radical, the language used for these functions will be in Rust. A baseline function will take a source location and a series of destination points as input, then run Dijkstra's shortest path algorithm on the input nodes. This process will include querying the database to retrieve edges between nodes and the distances between them. The function will then output the shortest path between the requested locations, along with the total travel distance.

One additional feature I hope to add is rerouting users when they veer off the supplied path during navigation, or even simply detecting if users have gone off course. In this case, I would have to store the user location in the database as well. The addition of this data would require the use of a stateful application, and Radical's advantage is that it can provide the benefits of a serverless application to something that must be stateful. Another feature that can bring out the advantage of using Radical is integrating real time traffic data when determining the paths.

Radical Integration

To integrate my application with Radical, I would need to compile my code to WebAssembly. In order to compare the effectiveness of Radical, I think I would also need a serverless version of my application. This would mean writing a version of my application using AWS lambda.

Additional Features Summary

Here is a summary of a list of potential features that I can add to my project once I get the baseline working on radical:

1. Rerouting users when they veer off course
2. Account for live traffic data when deciding routes
3. Test Radical using PSQL
4. Creating a visualization tool to view output paths

Evaluation.

I plan to develop two versions of my map application: one following a traditional server-based approach and another built on top of a serverless approach using Radical. To support these versions, I will use three AWS data centers: one for the main data center where DynamoDB will reside, and two as edge locations situated closer to the client. To simulate a true edge environment, I will restrict the storage capacity at the edge locations. I will then run Dijkstra's shortest path algorithm across a series of test cases, including a base case, varying path complexities, and different lengths of input nodes. The latencies from both versions will be recorded and compared across these experiments.