Database Project Report

Tim Walsh
COSC 580
Georgetown University

May 2, 2018

Abstract

This report explains the details of system design, configuration, usage, and limitations of my project. The goals of the project are to demonstrate a nearest-neighbor (kNN) search for restaurants in the Yelp Dataset [1], and to demonstrate the two-phase commit (2PC) protocol to insert new restaurants into a distributed system with multiple nodes.

1 System Design

My system is built on top of several Ubuntu 16.04 LTS virtual machines (VM) running in VirtualBox, MySQL 14.14, and Python 3.6 with tkinter as the user interface (GUI) [5]. It leverages the Yelp Fusion API [4], the Google Geocoding API [7], and the Google Static Maps API [8].

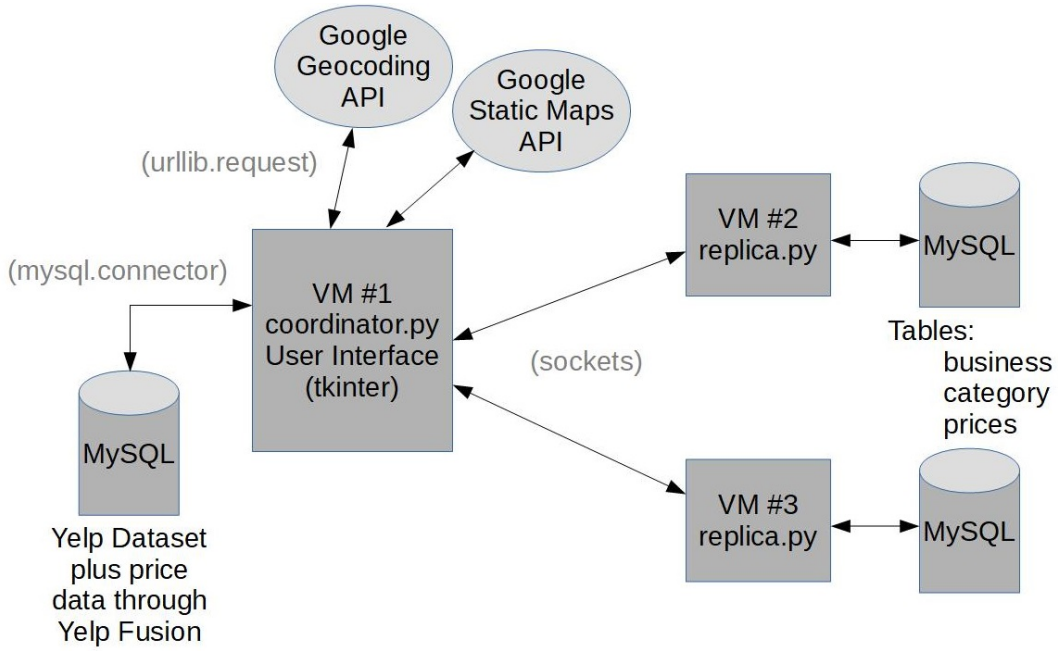

Figure 1: Conceptual diagram of the system

The primary VM has the full Yelp Dataset [1] in its database, augmented by the price data for all restaurants in Las Vegas, and runs the code to provide the GUI and act as the coordinator for 2PC. On start-up, it establishes a connection to the database, builds the GUI, and awaits user input.

The replica VMs have the Yelp Dataset schema in their database, plus the prices table, but do not need to contain the full Yelp Dataset for the sake of demonstration (in order to save disk space on the host machine). The replicas run code that is just a wrapper around the database to communicate with the coordinator during 2PC. On start-up, they establish connections to their databases, create UDP server sockets, and await instructions from the coordinator [10].

kNN Search When a user issues a search query, only the primary VM is involved. It performs a linear-time nearest-neighbor search over the restaurants offering the specified type of cuisine. I rank query results in ascending order of their linear combination of distances over the parameters of: geographic distance from a specified address, price from a specified price (from $ to $$$$), and rating from a specified rating (from 1 to 5 stars), with each parameter given a weight and normalized. I convert the specified address into latitude and longitude coordinates through the Google Geocoding API [7], and restaurant coordinates are the in the Yelp Dataset. I obtain price data for each restaurant in Las Vegas, as a one-time operation, through the Yelp Fusion API [4]. Star ratings are in the Yelp Dataset. While iterating through each row of the dataset, I keep the current top 9 results in a max-heap, and check each row against the current max. If the row's distance is less than the current max, I pop the max and push the new row onto the heap, so I only ever need to store 9 rows at one time in memory. After getting the final results, I generate a new map image [6] through the Google Static Maps API [8] to visually plot the specified address and the top results, as well as printing the list of results.

Distributed Insertion As I reviewed specifications of 2PC while working on this project, I learned that many design decisions are left to developers. 2PC is clearly defined in terms of what happens as long as none of the participating nodes fail (i.e. meaning "failure to communicate" here, not voting "no" during a transaction): either all nodes vote "yes" and the transaction commits, or some node votes "no" and the transaction aborts. Left undefined is what should happen if the coordinator or some replicas fail for a period of time during the transaction and recover, or not. If replicas fail, the coordinator can either keep the transaction open (blocking other transactions) until the protocol completes, abort the transaction after reaching a timeout, or tell remaining nodes to commit (assuming all voted "yes") after reaching a timeout and reconciling things offline with the failed node later when it recovers. Similarly, if the coordinator fails, replicas can either wait indefinitely for the coordinator to come back online, or one replica can step up into the role of coordinator to continue the transaction after reaching a timeout. These choices have significant consequences in terms of how and when nodes send messages and log actions, and require a varying degree of complexity. In my system, for the sake of simplicity, I chose to handle replica failures by keeping the transaction open until the replica recovers. I do not handle the case where the coordinator fails and recovers, nor the case where a replica fails permanently.

When a user attempts to insert a new restaurant, the coordinator sends a UDP packet containing three SQL statements that make up the transaction to each replica node. I chose to use UDP with the replica nodes acting as servers in order to simplify the recovery of replica node failures, as I found it quite complicated to re-establish broken TCP connections during a transaction. The coordinator re-sends the packet with statements to prepare every five seconds until it gets a vote from each replica, thereby handling the case where nodes fail prior to casting votes, as the recovered nodes will just get the statements again. The coordinator then sends the commit or abort command every five seconds until it gets an acknowledgment from each replica, thereby handling the case where replicas fail after casting votes but before receiving and executing the phase-two command, as the recovered nodes will just get the command again. Replicas log the prepared statements when they vote, log the coordinator's command upon execution of it, and then clear the log after sending the acknowledgment. During a recovery, if replicas find a live transaction in the log, they redo work as necessary to catch up with the ongoing 2PC process, and then rejoin at the appropriate step.

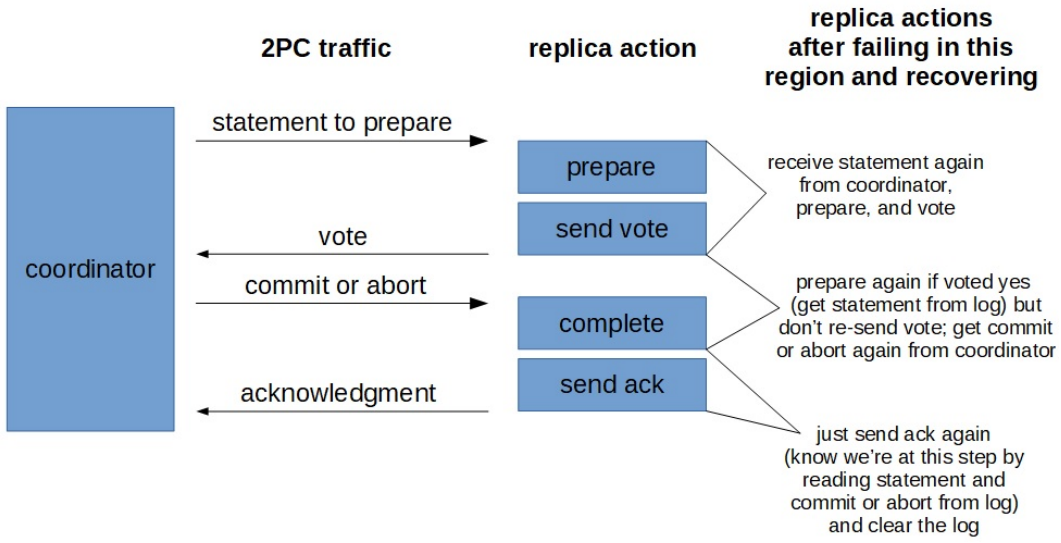

Figure 2: Failure points and recovery steps for replica nodes

2 Configuration Instructions

On all VMs, install Ubuntu 16.04 LTS, MySQL 14.14 (with "apt-get install mysql-server"), and Python 3.6. Run "apt-get install python3-pip", and then "pip3 install mysql-connector" [3].

Adjust VirtualBox settings so that each VM has two network interfaces: one with NAT (to communicate with the Internet), and one with Host-Only Adapter (to communicate between VMs) [9]. Run "ifconfig" on each replica VM to get their IP addresses.

On each VM, login to MySQL and create a new database named "yelp_db" [2].

On only the primary VM, download the Yelp Dataset [1] and load the dataset into "yelp_db" with the downloaded "yelp.sql" script. Next, on only the primary VM, run "python3 clean-data.py [MySQL root password]" (to remove some erroneous entries and extraneous tables in the Yelp Dataset) and "python3 get-prices.py [MySQL root password]" (to get price data through Yelp Fusion).

On the replica VMs, simply run "python3 build-schema.py [MySQL root password]".

3 Using the System

On the primary VM, run "python3 coordinator.py [MySQL root password] [replica #1 IP address] [replica #2 IP address]". On each replica VM, run "python3 replica.py [MySQL root password]".

To issue a search query, fill in the address box with a valid Las Vegas place or address (omit the city, state, and zip code), select cuisine, stars, and price from the drop-down menus, and adjust weights for each criteria as desired. Weights should sum to 1.0, but you can enter any positive values to obtain your desired ratio. Anything in the name box will be ignored during a search query. Press search and the GUI will refresh with your results.

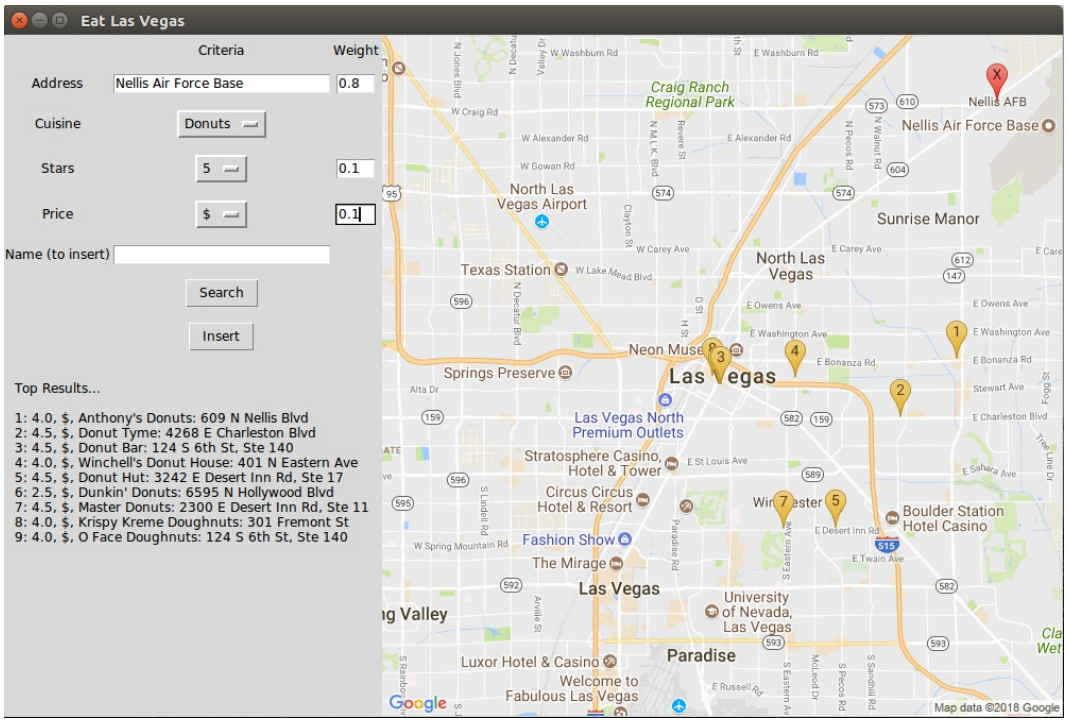

Figure 3: Screenshot of the user interface after a search query

To insert a new restaurant, fill in the address box with a valid Las Vegas place or address (omit the city, state, and zip code), select cuisine, stars, and price from the drop-down menus, and enter a name for the restaurant. Anything in the criteria weight boxes will be ignored during an insertion. Press insert and wait for a result. If all three nodes are able to commit the transaction, the GUI will display that the insertion was successful and the map will show the new restaurant's location. If any node is unable to commit the transaction, the GUI will say that the insertion failed. The most likely reason for a failure is that the database already has an entry for that restaurant name at that address. I generate a unique identifier by concatenating the name and address, and taking the first 22 characters of an MD5 hash digest, and the database will not accept two entries with the same identifier.

4 Limitations

In its current form, as simply a demonstration of database concepts, my project is extremely brittle in terms of both security and scalability. I have not made any attempt to sanitize inputs against SQL injection attacks, prevent abuse of the Yelp and Google APIs, or protect against sniffing/spoofing/modifying traffic between the coordinator and replicas. The system may accept and crash on certain inputs to the criteria and weights. The linear-time algorithm used for search is likely unacceptable on large enough datasets. In the presence of heavy network traffic, replicas may receive and try to act on partial or out-of-order data sent by the coordinator. My implementation of 2PC can withstand the failure and recovery of replica nodes, but not a failure of the coordinator. There are some critical points at which a failure of a replica will cause the entire system to hang.

The most interesting extensions to this work would be implementing recovery for a failed coordinator, involving replica nodes in search queries (returning the answer from the fastest node), and launching new threads for every insertion to allow handling of multiple insertions (possibly by multiple coordinators) in parallel.

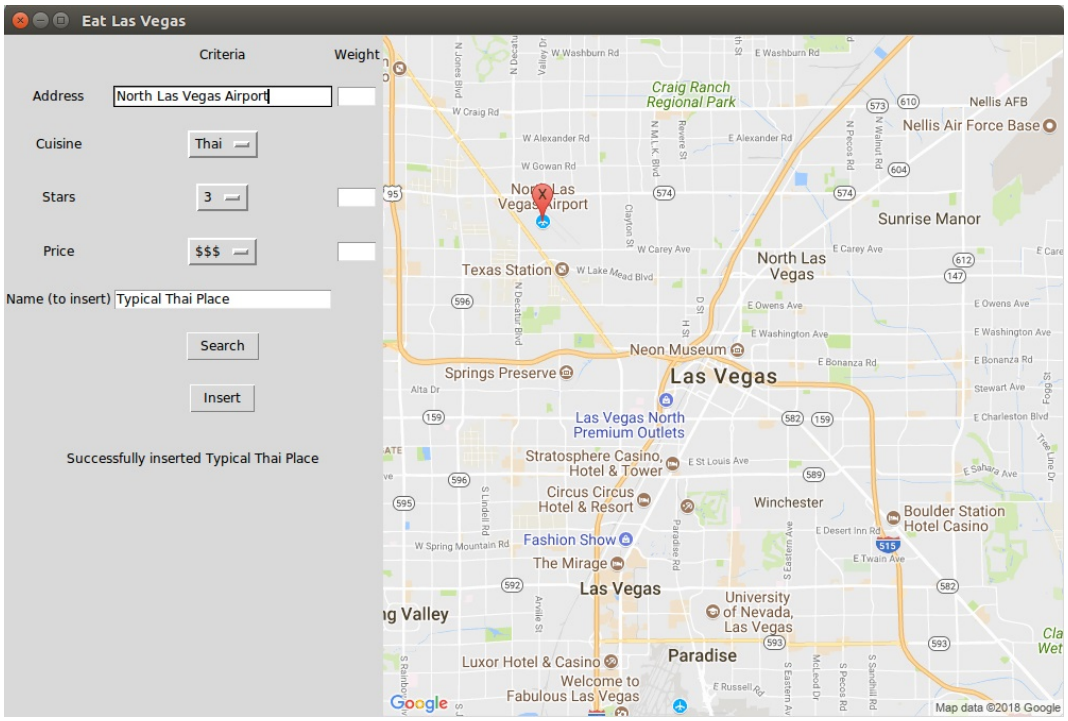

Figure 4: Screenshot of the user interface after inserting a new restaurant

5 Conclusion

The goals of this project were to demonstrate a nearest-neighbor search for restaurants in the Yelp Dataset, and to demonstrate the two-phase commit (2PC) protocol to insert new restaurants into a distributed system with multiple nodes. For the purpose of demonstrating a proof-of-concept, I succeeded in meeting those goals. This hands-on practice made me more familiar with tools like Linux, VirtualBox, MySQL, Python, and commercial APIs. More importantly, it gave me a much deeper appreciation for the details and nuances of 2PC and other consensus protocols for distributed systems.

References

[1] https://www.yelp.com/dataset/documentation/sql

[2] https://dev.mysql.com/doc/refman/5.7/en/tutorial.html

[3] https://dev.mysql.com/doc/connector-python/en/

[4] https://www.yelp.com/developers/documentation/v3/business

[5] https://www.python-course.eu/python_tkinter.php

[6] https://www.daniweb.com/programming/software-development/code/449036/display-an-image-from-the-web-tkinter

[7] https://developers.google.com/maps/documentation/geocoding/intro

[8] https://developers.google.com/maps/documentation/static-maps/intro

[9] https://www.thomas-krenn.com/en/wiki/Network_Configuration_in_VirtualBox

[10] https://pymotw.com/2/socket/udp.html