Read/write transaction-level routing in Odyssey (2020)

Project Description

Odyssey is an advanced multi-threaded PostgreSQL connection pooler and request router. Odyssey allows defining client routing rules by specifying client database, client user name and storage connection. In some HA scenarios, it is necessary to define different routing for read-only and read-write transactions. Odyssey can parse transaction opening statements and assign different routes depending on transaction types.

It would be cool to support many consistency models for reads on standby: 1. Redirecting read query on standby only if standby is lagging no more than X seconds 2. Redirecting read query on standby unless there was recent write (writer should see his writes) 3. Redirecting some read queries on syn or async standby

Skills needed

- C programming

Difficulty Level

Moderate.

Potential Mentors

- Andrey Borodin can mentor.

A list of deliverables A detailed description Outline your approach Outline your approximate schedule A document describing your background Please also include if you have other commitments during the period of GSoC. This includes jobs (part time or full time), exams, vacation, and anything else which might prevent you from working on your project.

Updated as of: March 30, 2020, 3:26 pm EST

Background on me:

At my previous coop (Electronic Arts), I was working on a telemetry system for games in (relatively) low bandwidth networks during launch time. My coop project was to leverage the telemetry system to build a highly available ETL application that

brokered telemetry messages to various teams and gave pertinent statistics on the messages. Besides the application itself, my contribution to the overall process in improving their broker is as follows:
1. I replaced their external Redis keystore with a erlang consistent hashing plugin (some context: we used RabbitMQ to queue the messages, RabbitMQ uses erlang internally, and RabbitMQ had a consistent hashing plugin but it wasn't customized to what we wanted). Instead of using an external Redis service to broker messages, I exploited RabbitMQ's builtin functionality to achieve the same outcome, while also ingesting more messages (an important improvement especially during game launch time).
2. I also changed the client to be idempotent (some more context: RabbitMQ has three levels of guarantees: this lowest level guarantees that >= 0 messages are sent, the second level guarantees that >= 1 message is sent, and the final level guarantees that exactly one message is sent). After I made my changes to the client, which used the highest level of the guarantees, we could downgrade to the second level and increase the number of messages being ingested by the system while not compromising on the data sent (through some other features of RabbitMQ).

Before EA, I was at Thales where I worked on building low latency camera drivers in C and C++. It involved designing and implementing a parallel camera driver in C++ resulting in 56 % lower latencies with multiple high resolution cameras.

At school, I'm part of Watonomous, a self driving car design team competing in the Autodrive competition. I have worked in several capacities over the past approx. three years (I pulled this from my resume):

- Optimized A* path exploration by 70% based on estimated velocity profiles of surrounding objects in C++
- Implemented fast lookup to find lane ids in HD Maps based on gps coordinate using rtree and convex hulls in python; also employed memory mapping for large files and thread pools
- Implemented real time dynamic routing for global map, consequently reducing the reevaluation time for the cost map by 23% in python
- Used OpenVino to load CNN onto FPGA; used OpenCL and Quartus to make custom activation functions in OpenCL/C++
- Implemented sketch sampling module to efficiently determine Euclidean distances between filtered sparse LIDAR data points with OpenCL/C++ resulting in a 40% increase in noise rejection for landmark detection in SLAM techniques
- Implemented graph signal processing noise rejection to improve feature quality by 35 % in 3d point clouds in C++
- Featured in Documentary - Shifting Gears: Autonomous Vehicles

My systems programming and concurrency class was in C:
https://github.com/AnkilP/ece252/tree/lab5

A list of deliverables:

1. Create utility functions
2. Create routing rules for:
    a. Redirecting read query on standby only if standby is lagging no more than X seconds
    b. Redirecting read query on standby unless there was recent write (writer should see his writes)
    c. Redirecting some read queries on syn or async standby
3. Documentation & Testing

A detailed description:

Docs on standby replicas
https://docs.oracle.com/cd/A84870_01/doc/server.816/a76995/standbyc.htm

Same issue but from 2019
https://github.com/yandex/odyssey/issues/47

Docs on async/sync replicas
https://severalnines.com/resources/tutorials/mysql-replication-high-availability-tutorial

Timeline:

It is assumed that there is a code review at least after every week.

Pre GSOC

Create toy problems with each of the three deliverables and determine what possible requirements and constraints I will have in successfully completing this project. This time will mostly be spent getting acquainted with the Odyssey code base and identifying the best questions to ask during the community bonding period.

Community Bonding

Getting acquainted with the group's preferred git etiquette. I will also need to determine the most relevant requirements from the shareholders. These discussions will include determining the most useful metrics to measure success, possible scenarios to test against, points of contact for various components and any other caveats.

Week 1

Understand the relevant parts of the Odyssey code base and use the toy problems to flesh out a system design/class diagrams and submit for review.

Week 2

After the system design is approved, I'll start creating tests and skeleton code.

Weeks 3-4

Create utility functions:

1. Create test cases specific to redirecting read queries to standby provided standby is lagging no more than X seconds
2. Create function to periodically query standbys to compute replication lag
3. Create test cases specific to redirecting read queries after writes
4. Create function/state to track writes (this will require research that will happen in both the pre-GSoC stage and the community-bonding stage)
5. Create test cases specific to redirecting read queries (semi-)synchronously or asynchronously to standby replicas

6. Still researching this but MariaDB uses GTID to help provide consistency for semi-synchronous cases - create GTID function or leverage PGSql's existing feature.

Weeks 5–6

This would be the mid term review period. Create routing rules for all three deliverables.

1. Route to standby in pool if lag < X
2. Route to standby if no recent writes
3. Route to standby either asynchronously or semi-synchronously
    a. Handle confirmation differently depending on case
    b. Effectively load balancing on read-slaves

Week 7

Provided all unit tests have passed, it's time to benchmark the implementation and optimize if necessary. I'll do regression testing as well to ensure the new features are compatible with the old ones.

Week 8–10

Perform Load Testing, Spike Testing, Stress Testing, Endurance Testing, Volume Testing, Scalability Testing - this feature would be primarily useful for scaling PostGreSQL and as a result, it will be important to test these scenarios.

Week 11–12

Take feedback from the community and iterate on the designs. Create documentation and submit for approval.

Week 13

Extra week in the event of an emergency or otherwise.

Other commitments:

I'll be working full-time but I don't see it being an issue - I will be able to commit at least 40 hours over the week: 4-5 hours/weekday and the rest on the weekend. It will start in May and end in August. I'm confident that I will have acclimated to my full-time job by the start of GSOC. The hours are also flexible.

Questions for mentors:

For read-after-write functionality, are we referring to eventual consistency?