GSoC 2021 PostgreSQL Project Proposal

Develop Performance Farm Benchmarks and Website

1. Basic Information

• Name: YoungHwan Joo

• Website (CV & Portfolio & Links): https://rulyox.com

• Email: rulyox@gmail.com

• Location: Seoul, South Korea (UTC+09:00)

2. About me

I am YoungHwan Joo, a 20-year-old student majoring Computer Science in Hanyang University, Seoul, South Korea. I finished 4 semesters and I took a semester off. So, I am not attending university classes in 2021. I am mostly interested in full-stack web development and databases.

I am currently working at Bio-Medical Informatics Lab, Seoul National University Hospital as an application developer and a data scientist. I worked at SNUH from December 2019. I worked full-time in vacations and part-time during the semester. One of the biggest projects that I developed is called Drug Dashboard. This is a web application that visualizes statistics about drugs that are used. I used the OMOP CDM Schema on PostgreSQL and AWS Redshift. I developed this full-stack including databases, back-end, front-end, and deployment.

Also, I am participating in a project called LobbyView which is from Massachusetts Institute of Technology. I worked on data querying using SQL, front-end development, and API development. I have been working at LobbyView since July 2020 and this project is not a full-time job but more of a sub project.

I am familiar with technological stacks such as Front-end using React, Vue, Back-end using Node.js Express, Python Flask, Django, Java Spring, Relational Databases, and Infrastructure Systems such as Docker.

Furthermore, I love learning new technology and solving problems. I can quickly learn and combine software skills to solve a complex problem.

3. Why I am interested in PostgreSQL

I have always been passionate about open-source and I use a lot of open-source software, frameworks, and libraries. However, I didn't have a chance to contribute to an actual open-source project. This is why I am applying to GSoC 2021.

Because my biggest interests are web development and databases, and because I am using PostgreSQL a lot, I thought that I would be most passionate if I could contribute to PostgreSQL's projects.

While I was working in the organizations above and doing my personal projects, I learned some complex SQL skills such as Window Functions, CUBE, and ways to optimize Common Table Expressions. I am also a big fan of JSONB. Furthermore, I have used automatic API generation tools with PostgreSQL such as PostGraphile and PostgREST in my projects.

4. Project Abstract

PostgreSQL is a big and active open-source project. This means that the code changes frequently and there are a lot of branches and commits. When big changes are made, performance has to be tested.

Performance Farm is a program that can be used to collect and visualize the benchmark results while the source code changes.

The actual tests can be run on Linux and maxOS machines while the users can easily see the results using a website.

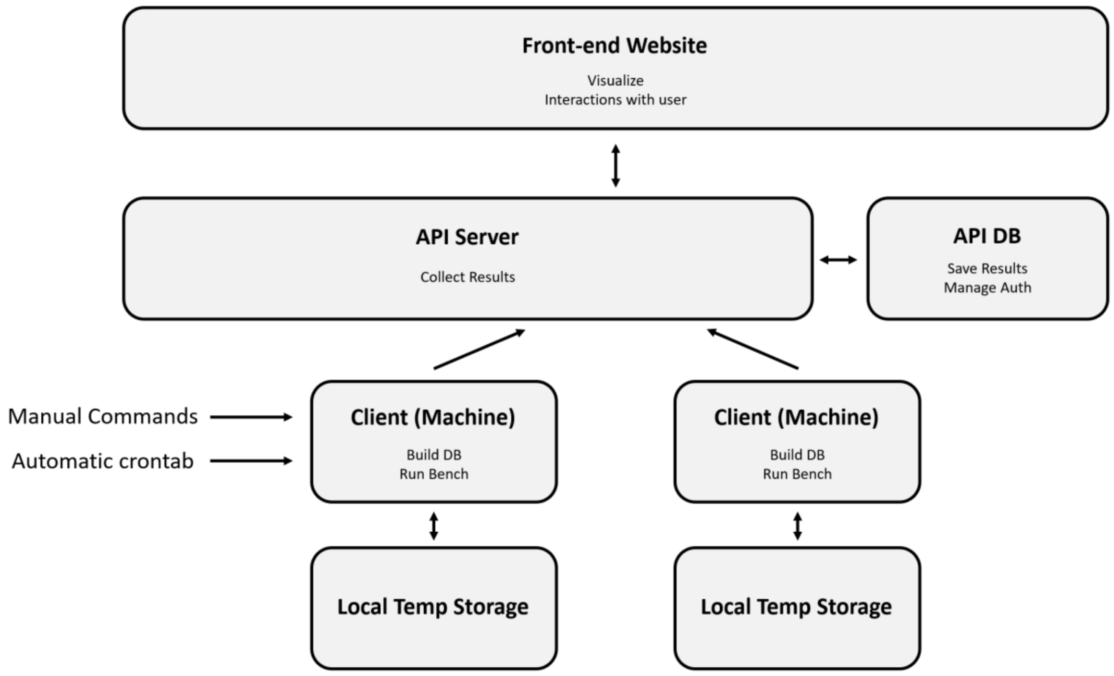
5. Current System

The Performance Farm consists of 3 parts.

• **Client**: Clones source code from a specific repository. Create a temporary PostgreSQL cluster. Run *pgbench*. Collect results. Upload results to the API.

• **API**: Uses Django. Receive benchmark results from the Client. Provides REST API. Uses DB for storing past results and user information.

• **Front-end Website**: Built with Vue. Visualizes benchmark results.



Current System Diagram

6. Features to be implemented

Below are the features that were in the GSoC ideas list and I added some details.

• Collect system metadata

When doing a benchmark, collecting information about the system is important. In the current version of the Client, some basic data from the DB, operating system, kernel, CPU, and data from *sysctl* is collected.

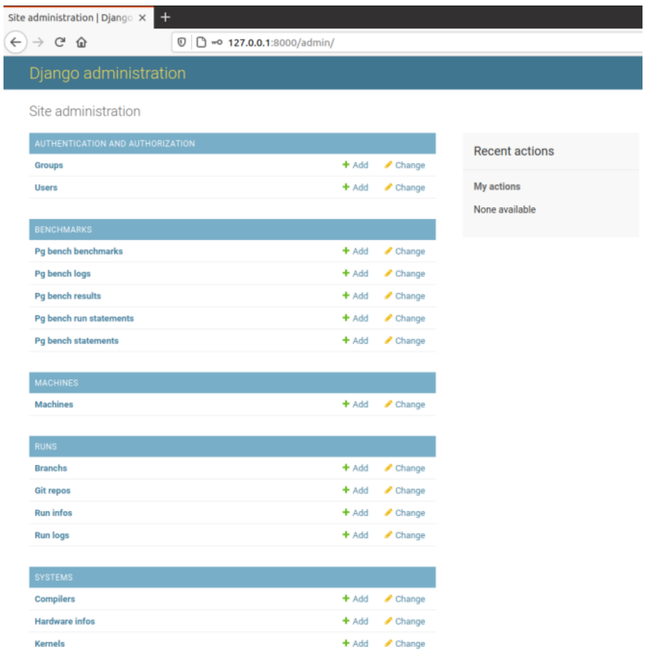
Using *collectd*, we can collect CPU and memory utilization during the benchmark. I think that data such as highest values or the time period it held the highest utilization rate could be useful. The package should be installed in the system.

With the *pg_stat_statement* extension, we can get some internal data of the DB such as number of cache hits of shared, local, and temp blocks. This can be easily enabled by updating *shared_preload_libraries* and executing *CREATE EXTENSION*.

Because the collected data are stored in the API's database, the database table about the system should also be refactored. (Which is currently at *rest_api/systems/models.py*.)

• Admin panel inside website

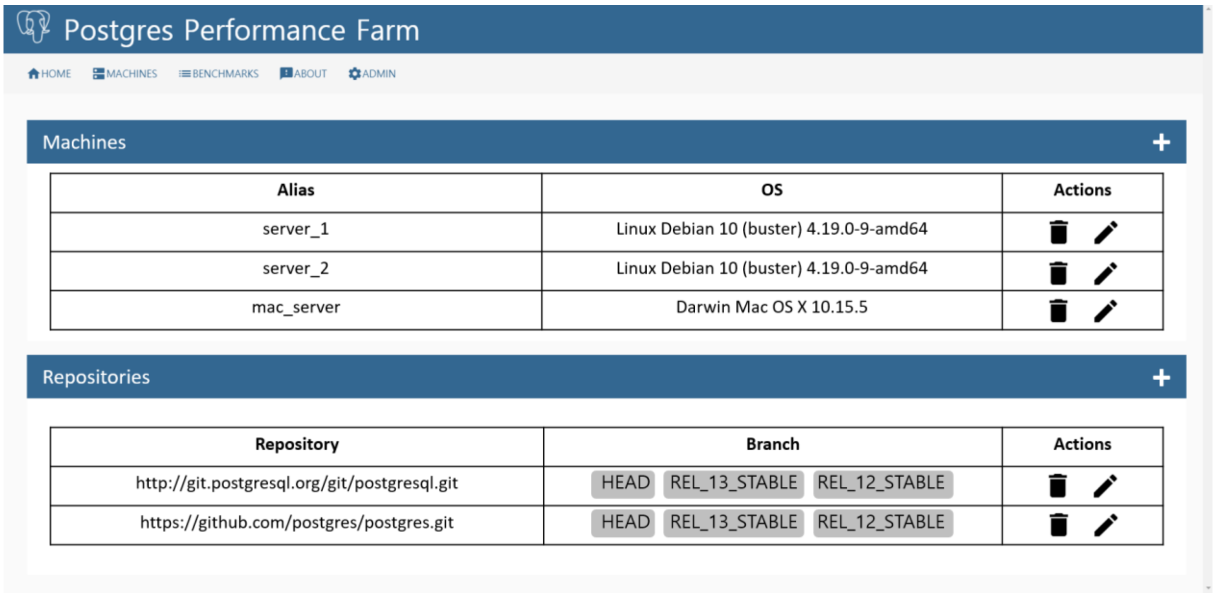
Currently, the only way to change the API's configurations is through the Django Administration. However, this should not be open to the public.



Django Administration Page

Adding an admin panel to the Front-end Website would be the appropriate way.

In the admin panel, there should be menus to manage the connected machines and to manage supported repositories.



Machines

Alias	OS	Actions
server_1	Linux Debian 10 (buster) 4.19.0-9-amd64	
server_2	Linux Debian 10 (buster) 4.19.0-9-amd64	
mac_server	Darwin Mac OS X 10.15.5	

Repositories

Repository	Branch	Actions
http://git.postgresql.org/git/postgresql.git	HEAD REL_13_STABLE REL_12_STABLE	
https://github.com/postgres/postgres.git	HEAD REL_13_STABLE REL_12_STABLE	

Administration Panel Mockup

This is the mockup example of the admin panel.

• **GUI improvements**

User Interface and User Experience is always an important point to consider.

The users of Performance Farm should be able to easily see the benchmark results, find especially fast or slow results, and compare results between specific time points or git commits.

First, improvements for the “trend” page.

Currently, Plotly is being used to visualize a Latency/TPS chart for each machine. However, these charts are broken in some situations. I hope to use D3.js which is a highly flexible visualization library and because this library has only few constraints, it will work well in most situations.

Also, I think adding a menu to change the X axis which is currently only commit to support both commit and time. Additionally, a way to select the starting point and end point of the X axis is needed because if there is a lot of data, it will be hard to see everything in a single chart screen.

Second, improvements for the “benchmarks” page.

Currently, the benchmarks page simply lists the configurations. This is hard to see if there are a lot of configurations. I think the configurations have to be sorted in the order by the number of machines that use it.

Also, there should be a filter to select specific configurations. For example, a user might want to search only the configurations that’s scale is 2 and that did a read-write test.

Finally, improvement about the whole website.

The first time I opened the Performance Farm website, I was confused because there were many different pages. Especially, run, trend, and result pages were confusing.

I think there should be some kind of navigation or page hierarchy that shows what page I am currently at. Also, a simple description for each page would be nice.

• Dependency removal

Programs that are in the PostgreSQL project have to be maintained stably. Also, contributors have to be familiar with the frameworks that are used. This is the reason why dependency removal is necessary.

The Front-end Website is using a lot of npm packages. Though npm packages are easy to use, they are heavy and they do not have a reliable repository.

Vue itself is also a dependency. Conversion to vanilla JavaScript will also be an improvement. However, if Vuetify is removed, we should think about alternative design systems or ways to deliver a clean and seamless UI.

The most important features of Vue that is used currently are reuse of components, routing using Vue Router, and state management using Vuex. These are the most important features that have to be developed when migrating to vanilla JavaScript.

If we are not using Vue Router, we don't have to make a SPA website but just make multiple pages. Also, a state manager can be made by having functions that update and subscribe the data.

7. Additional ideas

These are the ideas that I thought would be a nice feature for Performance Farm. The priority is the same as the order.

• Containerizing using Docker

I have cloned the Performance Farm source code and tested it. However, I had some dependency issues and some processes were uncomfortable. Because Performance Farm builds the whole DBMS, there were a lot of requirements needed.

If Performance Farm gets containerized, not only setting up everything on a new machine but also processes such as updating Performance Farm to a new version will be much easier.

There should be containers for each Client, API, and Front-end Website.

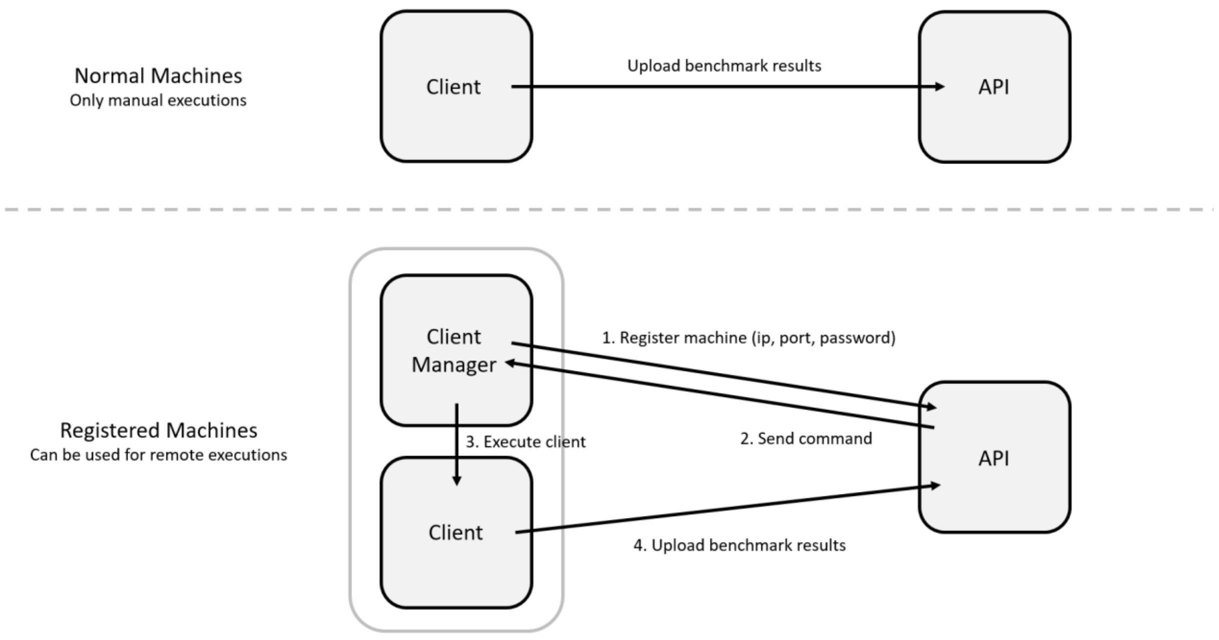
• Benchmark execution through API or Website using a Client Manager

Currently to manually start a benchmark, a user has to access the Client directly.

I think permitting the administrator or privileged users to start a benchmark at a specific machine would be a useful feature.

However, this could lead to security problems. For example, the admin of the API could damage the Client by sending too many execution commands. To solve this, I thought of a module called Client Manager.

A Client can be used in 2 ways. The first way is a normal machine which is the current way. The second way is a registered machine which uses the Client Manager. Only machines that are willing to be used by remote executions have to use the second way.



Normal Machines and Registered Machines

Client Manager is a module that acts as a simple HTTP server. First, it registers itself to the API. The API can send commands as a HTTP request to the registered machine. If the Client Manager receives a request, it executes the Client.

There should be some security measures such as checking authentication of the request. Also, the Client Manager should prevent overflowing the Client's workload. This can be done by rejecting requests from the API if the Client process is currently running or a lot of requests came in a certain time period.

• Support custom repositories

There are organizations and companies that build their own fork version of PostgreSQL. I thought that these organizations might want to run Performance Farm specifically for their forks to test their version's performance.

The current version of Performance Farm only displays results of official repositories. I hope to add options that enables to choose certain repositories to be displayed.

An example of this feature is in the admin panel mockup above.

8. Schedule

• Application Period ~ April 21 (3 weeks)

Before GSoC announcement.

Test Performance Farm using different environments and settings.

Use and understand unfamiliar PostgreSQL tools such as *pgbench* and *pg_ctl*.

Analyze existing code.

• April 21 ~ May 18 (4 weeks)

Design concrete improvement methods.

Make mockups for new web elements and the admin page.

Understand completely about the API's database.

• May 18

GSoC announcement.

• May 18 ~ May 26 (1 week)

Get to know the team better.

Community bonding is an important step because many people participate in open-source projects.

Make the overall frame for the new vanilla JavaScript website.

• May 26 ~ June 16 (3 weeks)

Coding officially starts.

Upgrade the Client and the API.

Add metadata collection to the Client.

Add authentication features to the API.

• June 16 ~ June 23 (1 week)

Create essential features such as components and state managers using vanilla JavaScript.

- **June 23 ~ July 13 (3 weeks)**

Migrate to vanilla JavaScript.

Use new design systems.

Remake charts using D3.js.

- **July 13 ~ July 17**

First evaluation between mentors and students.

- **July 13 ~ July 28 (2 weeks)**

Upgrade the website.

Add the admin panel.

Improve GUI.

- **July 28 ~ August 17 (3 weeks)**

Implement Client Manager and the remote execution API.

Test and fix bugs.

Containerize using Docker. (This won't take long, but should be done in the last step because it depends on the used dependencies.)

- **August 17 ~ August 24**

Submit code and final evaluation.

- **The future**

I hope to contribute to Performance Farm and other software in the PostgreSQL ecosystem even after GSoC 2021.