GSoC 2021 PostgreSQL Project Proposal

pgagroal: Metrics and monitoring

1. Basic Information

Name: Junduo Dong
Email: andj4cn@gmail.com
GitHub: An-DJ
Location: Jincheng, China(UTC+08:00)

2. About me

I'm Junduo Dong, a 22-year-old studying at China University of Geosciences. My major is Software Engineering, especially focusing on spatial databases and spatial distributed computing.

I once worked as a R&D intern in the Infrastructure Group of the Data Technology Department of Xiaomi in 2019. I am mainly responsible for the development of internal micro-service framework and the maintenance of infrastructure such as traffic monitoring. So I have some experience with monitoring system design and cloud-native monitoring tools such as Grafana and Prometheus.

I am familiar with Back-end stack and programming languages such as C/C++, Java and Golang. I have a lot of experience in the development of back-end services and the use of database middleware, such as Hikari, ShardingSphere, etc.

In addition, I am also an open source enthusiast. I love open source projects related to distributed systems and distributed databases and I am currently a code contributor for TiDB, a cloud native distributed database.

3. About pgagroal

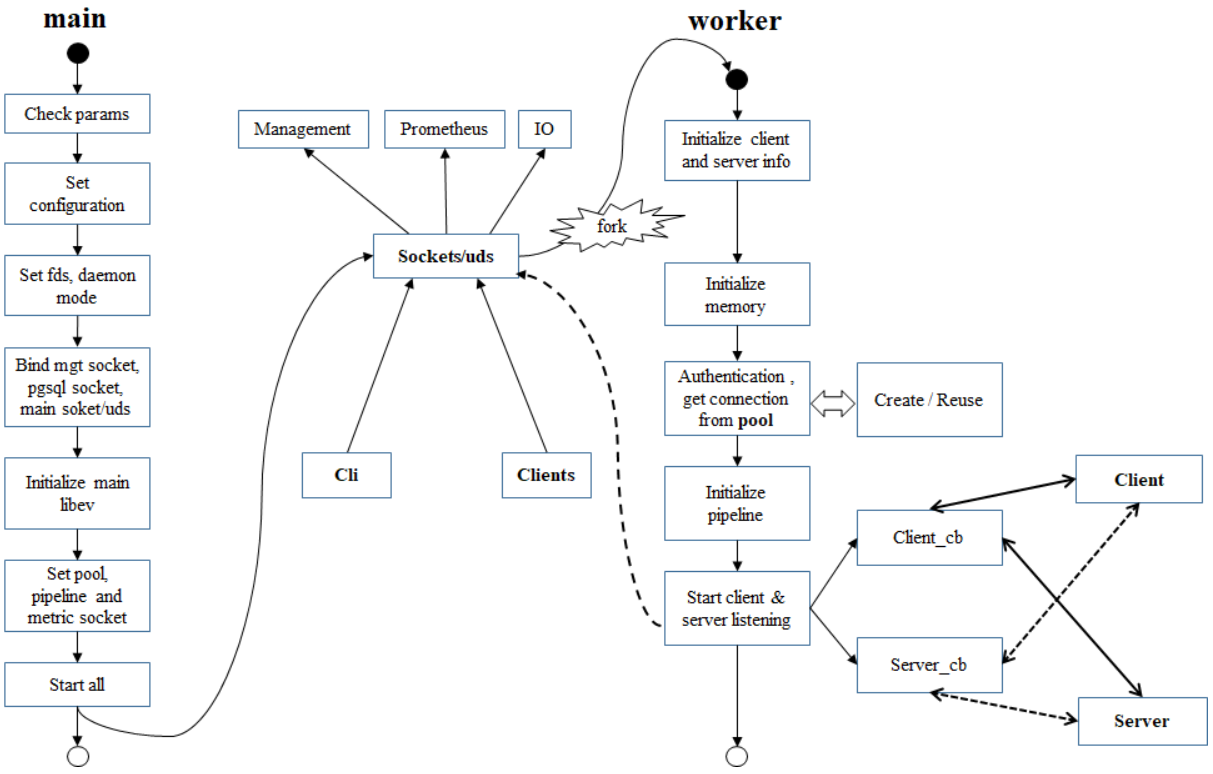
From my perspective, pgagroal is a protocol-native PostgreSQL connection pool which has higher performance than others. Pgagroal provides an out-of-the-box connection pooling mechanism to reduce the performance overhead caused by frequent database connection creation and destruction. Pgagroal provides authentication and limitation mechanisms to manage database access. The failover solution in pgagroal supports database failure recovery. Pgagroal's

administration and cli tools allow you to gracefully shutdown connection pool, remove idle connections and other management operations locally or remotely, which makes managing it extremely convenient. Meanwhile, pgagroal supports Transport Layer Security (TLS) V1.2 + and PostgreSQL v3 protocol.

Pgagroal uses the process model “fork” which forks a child process to handle one connection to PostgreSQL. The child process can be terminated or crashed when fatal errors occur or connection closed so that the main process of pgagroal cannot be influenced. Library libev is used in pgagroal to handle network requests caused by clients, database servers or internal modules.

Relatively speaking, pgagroal shows better performance by pgbench test [https://agroal.github.io/pgagroal/performance.html]. More importantly, pgagroal can be better integrated with cloud native ecology such as Prometheus and Grafana. Monitoring metrics can assist DBAs to judge the current state of the system. A good monitor dashboard can help DBAs troubleshoot performance bottlenecks or other exceptions in pgagroal. Prometheus support in pgagroal can make it easier to collect metrics and visualize them in the Grafana dashboards.

The main architecture of pgagroal is shown below.



4. About GSoC

Project "Metrics and monitoring" aims to collect more effective metrics which can stand for current status of pgagroal's pool, connections, clients, servers and internal state. Finally, the metric data can be illustrated by one or more Grafana dashboards to show clearly what DBA desires to know.

The monitoring points of the current system have been roughly complete, which offers a mature framework for adding more metrics and exposed by Prometheus. We can add more appropriate monitoring points and make practical and attractive Grafana dashboards.

5. Metrics

Pgagroal now includes metrics covering a number of areas such as the pool itself, limits of user-database-application configured in "pgagroal_hba.conf", connection state counter and so on. The current metrics can be roughly divided into 3 modules: "Pool", "Limit" and "Connection".

Based on my experience of monitoring systems and other implementations of connection pools, I think the monitoring metrics of pgagroal can cover the 6 areas (modules) below:

- **Pool** metrics which cover the pool state, result counter, authentication and so on. Metrics in pgagroal now include pool state, authentication counter and session distribution, but no normal query and transaction counter which can characterize the ability of current interactions.
- **Limit** metrics which cover all user-database-application-limit tuples. Metrics in pgagroal now has totally included all limitations such as max_connections and active_connections.
- **Server** metrics which cover all PostgreSQL instances' state, distribution of databases and so on. Metrics in pgagroal now just include server state and error counter, but no distribution of current connection and query amounts in different databases. The distribution metrics can show the hot or cold databases and give recommendations for DBAs to split or merge databases.
- **Client** metrics which cover clients' state, wait time and count before being allocated connections by pool and so on. There are no metrics about clients such as wait time and in use amounts which can show if the limit rule or resource configuration should be adjusted by the administrator.
- **Connection** metrics which cover all connections' state, result counter and so on. Metrics in pgagroal now include enough counters to show history and current connection state such as success return, error termination and timeout return etc.

- **Internal** metrics which cover resources which are in use such as network traffic, memory and socket. There are no metrics referring to internal resource information which can characterize the physical resource occupancy of pgagroal.

Combined with the above analysis, the more complete monitoring modules and details of pgagroal are as follows ('E' for 'Exist', 'N' for 'Not exist'). This GSoC project will focus on adding the following 'N' metrics according to their priorities.

Module	**Detail**	**Metric**	**Type**	**Status**
Pool	State	pgagroal_state	gauge	E
	Pipeline mode	pgagroal_pipeline_mode	gauge	N
	Session	pgagroal_session_time_seconds	histogram	E
	Authentication	pgagroal_auth_user_success	counter	E
		pgagroal_auth_user_bad_password	counter	E
		pgagroal_auth_user_error	counter	E
	Query	pgagroal_query_count	counter	N
		pgagroal_query_time	gauge	N
	Transaction	pgagroal_tx_count	counter	N
		pgagroal_tx_time	gauge	N
	Client wait time	pgagroal_wait_time	gauge	N
Limit	Limit	pgagroal_limit	gauge	E
Server	State	pgagroal_server_error	counter	E
		pgagroal_failed_servers	gauge	E
	Database connections	pgagroal_db_connections_bucket	histogram	N
		pgagroal_db_connections	gauge	N
	Database queries	pgagroal_db_queries_bucket	histogram	N
		pgagroal_db_queries	gauge	N
Client	Wait client	pgagroal_client_wait	gauge	N
		pgagroal_client_wait_bucket	histogram	N
	Inuse client	pgagroal_client_inuse	gauge	N
Connection	Connection result counter	pgagroal_connection_error	counter	E
		pgagroal_connection_kill	counter	E
		pgagroal_connection_remove	counter	E
		pgagroal_connection_timeout	counter	E
		pgagroal_connection_return	counter	E
		pgagroal_connection_invalid	counter	E
		pgagroal_connection_get	counter	E

		pgagroal_connection_idletimeout	counter	E
		pgagroal_connection_flush	counter	E
		pgagroal_connection_success	counter	E
	Connection state	pgagroal_connection	gauge	E
		pgagroal_active_connections	gauge	E
		pgagroal_total_connections	gauge	E
		pgagroal_max_connections	counter	E
Internal	Memory	pgagroal_mem_alloc	gauge	N
	Socket	pgagroal_self_sockets	gauge	N
		pgagroal_conn_client_sockets	gauge	N
		pgagroal_conn_server_sockets	gauge	N
	Network Traffic	pgagroal_sent	gauge	N
		pgagroal_receive	gauge	N

The 'N' metrics above have the following meanings.

Module	**Metric**	**Note**
Pool	pgagroal_pipeline_mode	pipeline mode(perf, session, tx)
	pgagroal_query_count	counter of total query
	pgagroal_query_time	gauge of last query time
	pgagroal_tx_count	counter of total transaction
	pgagroal_tx_time	gauge of last transaction time
	pgagroal_wait_time	gauge of last wait time
Server	pgagroal_db_connections_bucket	histogram of history connections count(per database)
	pgagroal_db_connections	gauge of current connections(per database)
	pgagroal_db_queries_bucket	histogram of history connections count(per database)
	pgagroal_db_queries	gauge of current connections(per database)
Client	pgagroal_client_wait	gauge of current wait client(per database)
	pgagroal_client_wait_bucket	histogram of history wait client(per database)
	pgagroal_client_inuse	gauge of inuse client
Internal	pgagroal_mem_alloc	gauge of memory allocation
	pgagroal_self_sockets	gauge of socket (main process, prometheus process, ...)
	pgagroal_conn_client_sockets	gauge of client socket
	pgagroal_conn_server_sockets	gauge of server socket
	pgagroal_sent	sent (to clients) bytes per second
	pgagroal_receive	receive (from clients) bytes per second

I noticed that the general steps for adding a new metric are below:

- Initialize the corresponding metric variable
- Add metric function to "prometheus.h" and implement it in "prometheus.c" by atomic function in C
- Call the function at the appropriate place to increase or decrease the metric
- Append the new metric to "data" variable in "metrics_page" function to show it on metric page

6. Grafana dashboard

After enough metrics have been added to Prometheus exporter, I will make a practical and attractive Grafana dashboard. The dashboard is mainly divided into 6 parts: "Pool", "Limit", "Server", "Client", "Connection" and "Internal". Each part consists of a suitable metric panel which corresponds to a pgagroal metric. For example, the state panel "pgagroal state" corresponds to the metric "pgagroal_state", and the bar chart "pgagroal_session_time_seconds" corresponds to the metric "pgagroal_session_time_seconds".

A panel is shown below that is configured based on the currently available metrics:

7. Schedule

April 14 - May 18	Read and familiarize myself with the code. Especially the detail of worker and message part.

May 18 - June 6	Learn more about pgagroal developer mode and try to link current Prometheus metric to Grafana. * Learn more about pgagroal community * Learn more Grafana new version features * Try to configure the relevant metric panel
June 7 - June 27 (3 weeks)	Implement internal metrics and configure its metric panel. * Implement internal metrics such as memory, socket and network metrics * Configure relevant internal monitoring Grafana panel
June 28 - July 11(2 weeks)	Implement client metrics and configure its metric panel. * Implement client metrics such as wait and in use client counter metrics * Configure relevant client monitoring Grafana panel
July 12 - July 17	First evaluation phase. * Submit my evaluations of mentors
July 18 - August 1 (2 weeks)	Implement server metrics and configure its metric panel. * Implement server metrics such as db connection and query metrics * Configure relevant server monitoring Grafana panel
August 2 - August 15 (2 weeks)	Implement pool metrics and configure its metric panel. * Implement pool metrics such as pool mode, query and transaction metrics * Configure relevant pool monitoring Grafana panel
August 15 - August 24	Final submission phase. * Submit all of code * Submit project summaries * Submit final evaluations of mentors