Pruebas y Resultados

Las pruebas se realizaron utilizando Gatling. Para la ejecución de las mismas se descargó una **demo** en la página oficial. Esta demo maneja los paquetes utilizando maven. Además, el dataset que se utilizó es de películas y sus ratings.

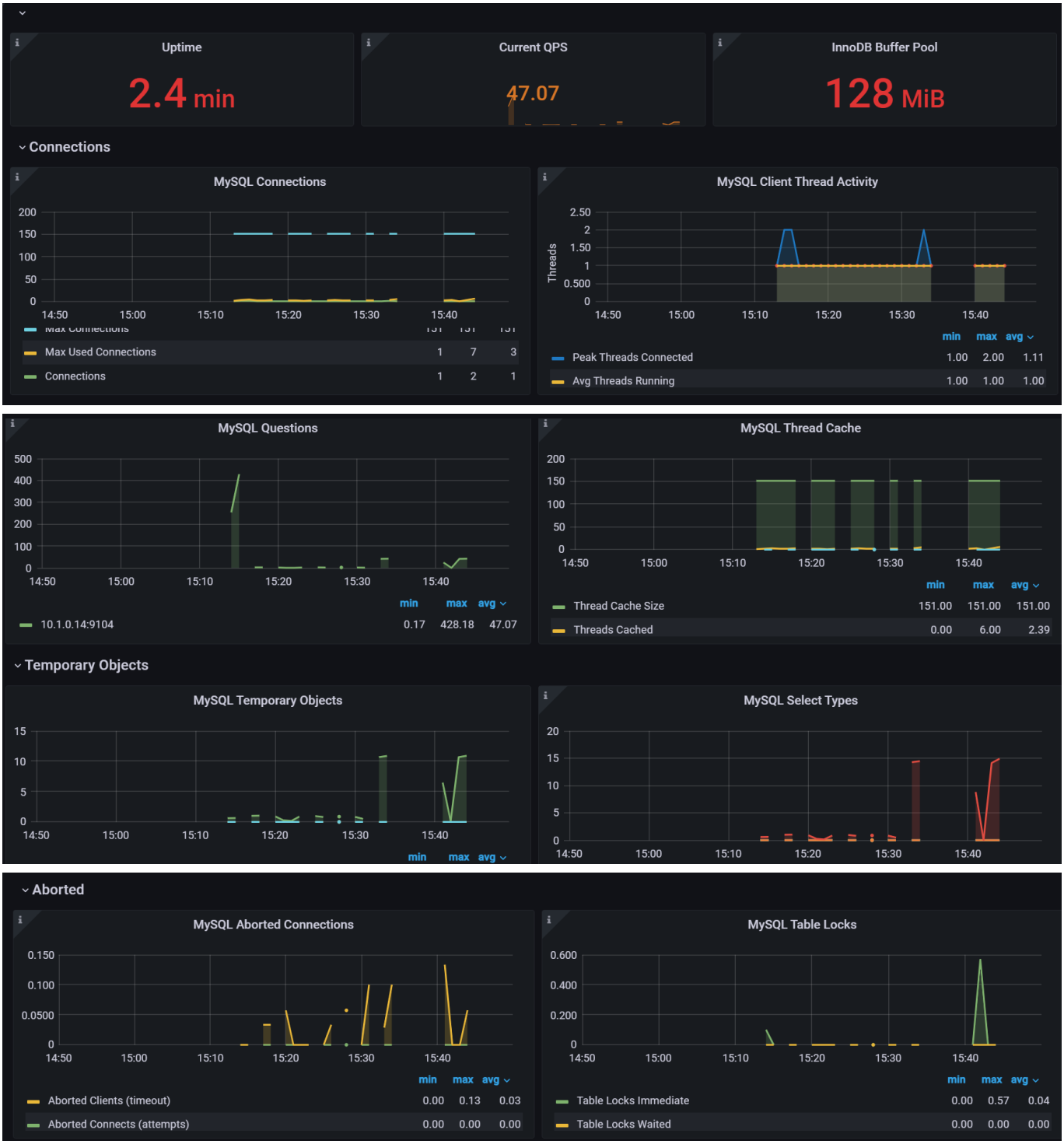
MariaDB

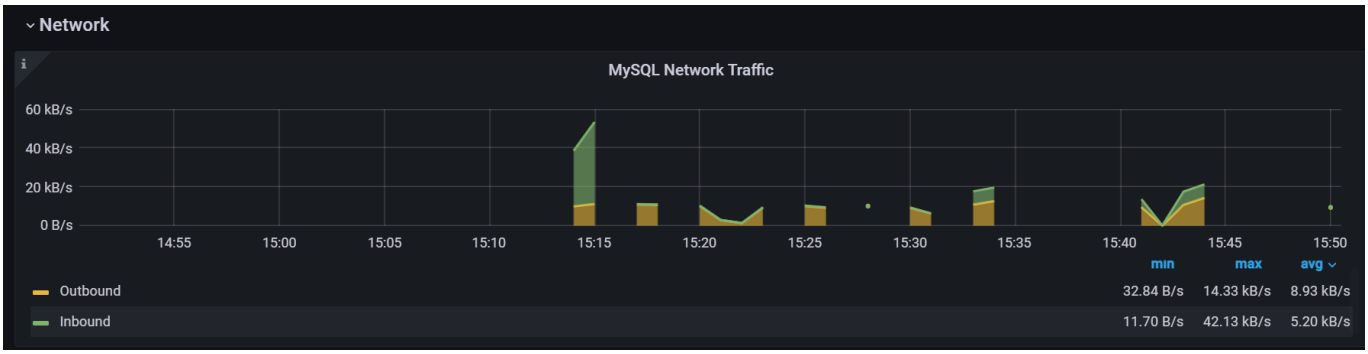
Primera prueba:

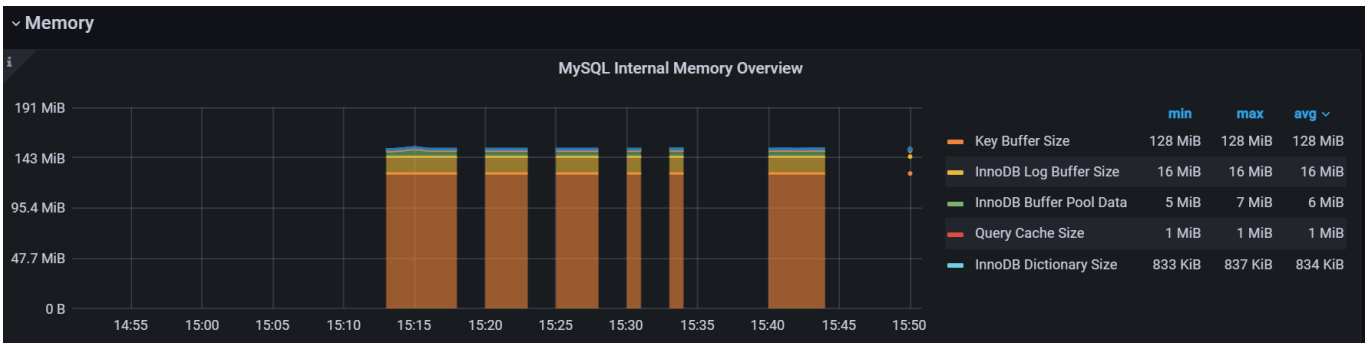
```
public class MariaDBLoadTest extends Simulation {

    // Definición de los diferentes endpoints
    private final String[] endpoints = {
            "/mariadb/movies/rating",
            "/mariadb/movies/best",
            "/mariadb/movies/trend",
            "/mariadb/movies/inconsistent"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random MariaDB Endpoints Load Test")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:51674")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Primera prueba:

Empieza en 15:15 Los cortes que se ven en los gráficos es porque el pod de MariaDB estaba inestable.







Segunda Prueba:

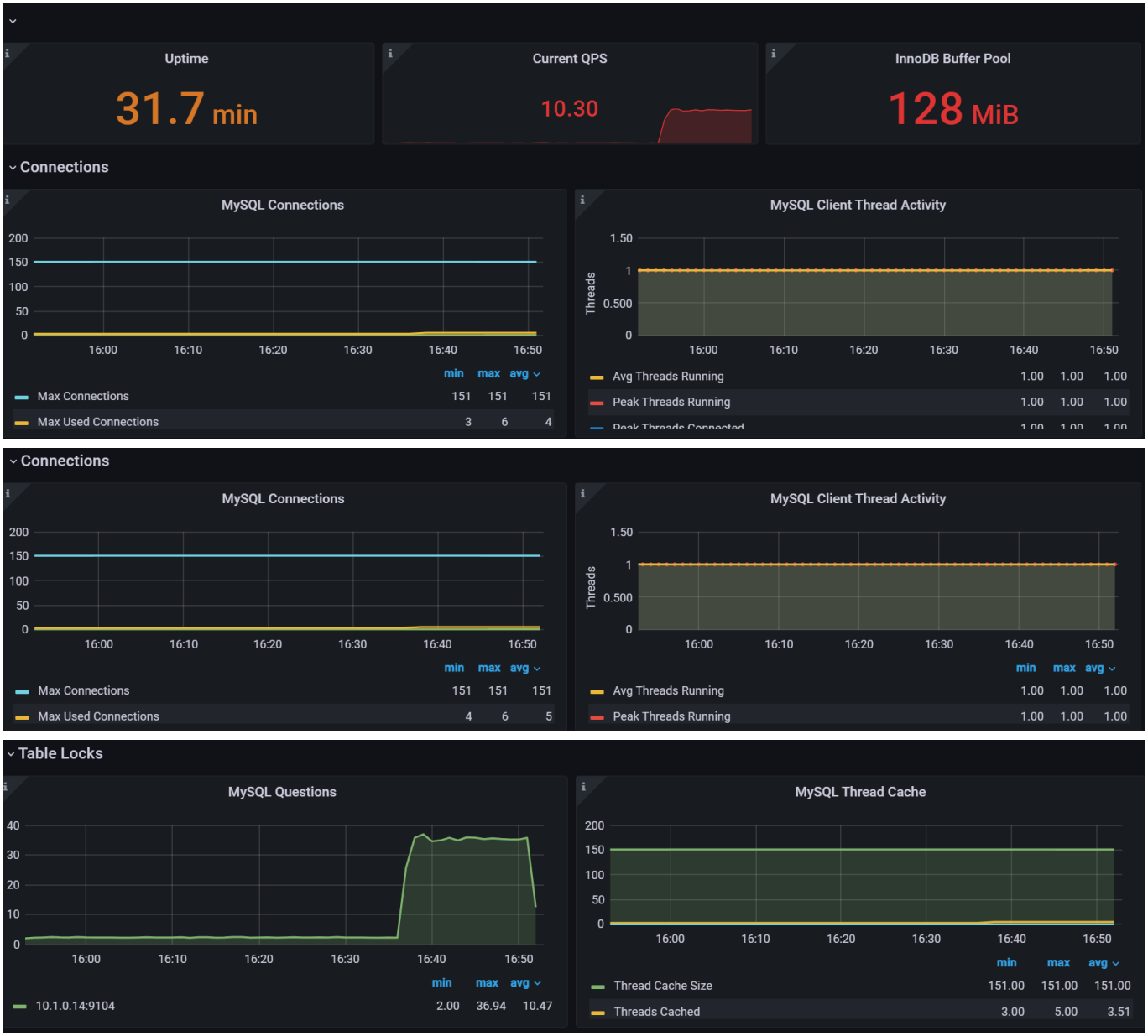
Esta prueba es con redis

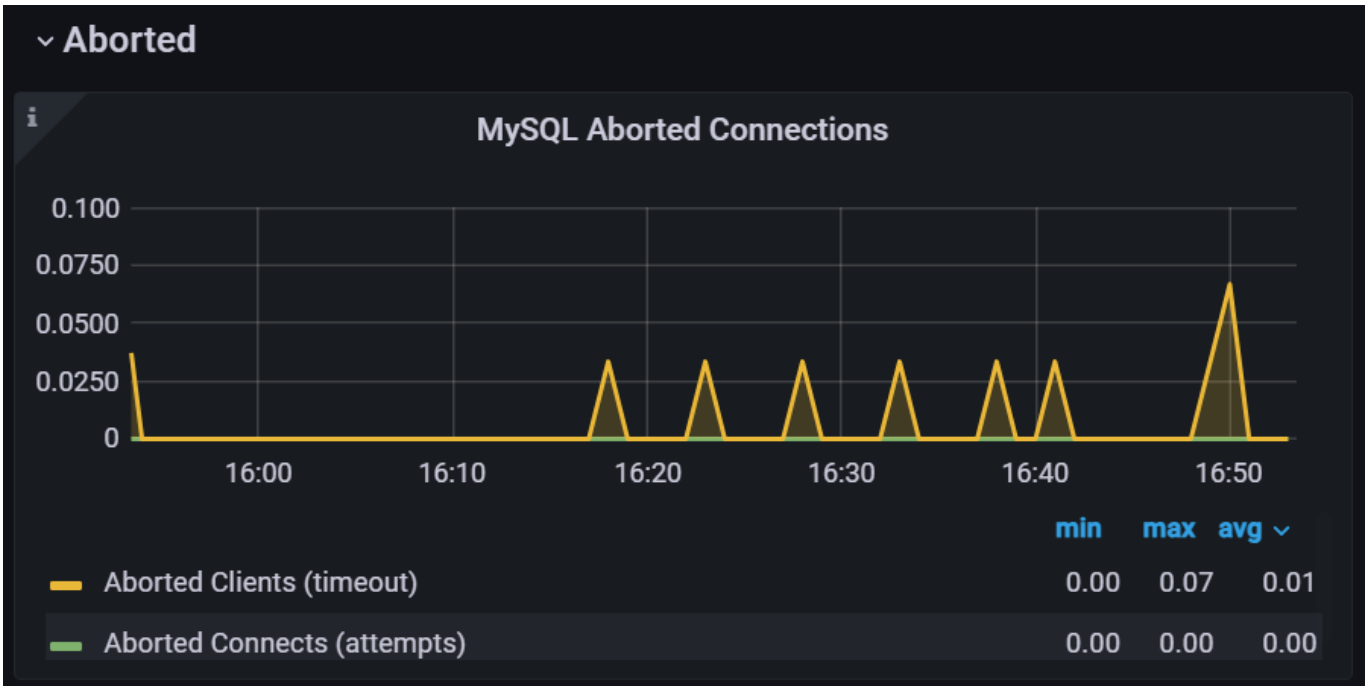
```
public class MariaDBLoadTestWithRedis extends Simulation {

    // Definición de los diferentes endpoints, incluyendo el uso de Redis
    private final String[] endpoints = {
            "/mariadb/movies/rating?cache=redis",
            "/mariadb/movies/best?cache=redis",
            "/mariadb/movies/trend?cache=redis",
            "/mariadb/movies/inconsistent?cache=redis"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random MariaDB Endpoints Load Test with Redis
Cache")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
```

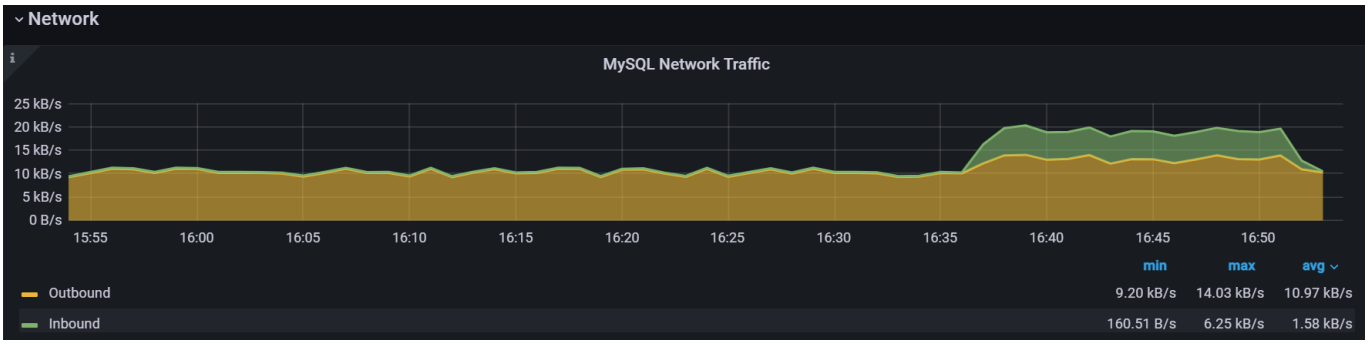
```
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

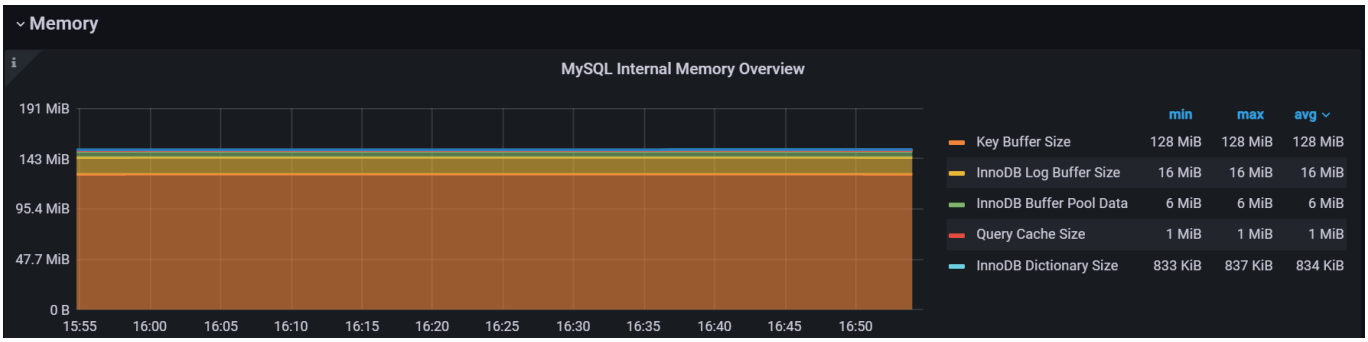
Resultados de la Segunda prueba:

Empieza en 16:35

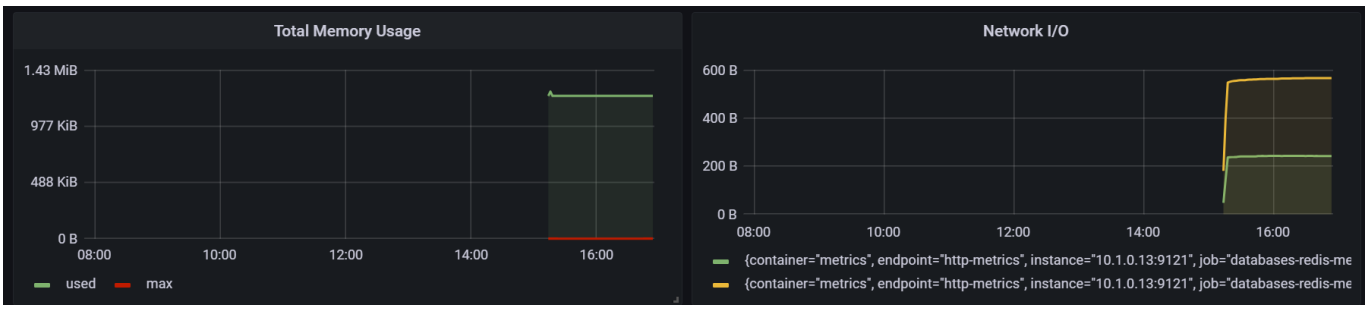



Aborted
MySQL Aborted Connections
0.100
0.0750
0.0500
0.0250
0
16:00
16:10
16:20
16:30
16:40
16:50
min max avg
Aborted Clients (timeout) 0.00 0.07 0.01
Aborted Connects (attempts) 0.00 0.00 0.00


Network
MySQL Network Traffic
25 kB/s
20 kB/s
15 kB/s
10 kB/s
5 kB/s
0 B/s
15:55
16:00
16:05
16:10
16:15
16:20
16:25
16:30
16:35
16:40
16:45
16:50
min max avg
Outbound 9.20 kB/s 14.03 kB/s 10.97 kB/s
Inbound 160.51 B/s 6.25 kB/s 1.58 kB/s


Memory
MySQL Internal Memory Overview
191 MiB
143 MiB
95.4 MiB
47.7 MiB
0 B
15:55
16:00
16:05
16:10
16:15
16:20
16:25
16:30
16:35
16:40
16:45
16:50
min max avg
Key Buffer Size 128 MiB 128 MiB 128 MiB
InnoDB Log Buffer Size 16 MiB 16 MiB 16 MiB
InnoDB Buffer Pool Data 6 MiB 6 MiB 6 MiB
Query Cache Size 1 MiB 1 MiB 1 MiB
InnoDB Dictionary Size 833 KiB 837 KiB 834 KiB


Uptime
2 hour
Clients
1
Memory Usage
∞%
Commands Executed / sec
6.60
6.40
6.20
6.00
5.80
08:00
10:00
12:00
14:00
16:00
Hits / Misses per Sec
1
0.750
0.500
0.250
0
08:00
10:00
12:00
14:00
16:00


Total Memory Usage
1.43 MiB
977 KiB
488 KiB
0 B
08:00
10:00
12:00
14:00
16:00
used
max
Network I/O
600 B
400 B
200 B
0 B
08:00
10:00
12:00
14:00
16:00
{container="metrics", endpoint="http-metrics", instance="10.1.0.13:9121", job="databases-redis-me
{container="metrics", endpoint="http-metrics", instance="10.1.0.13:9121", job="databases-redis-me

Para esta prueba se incluyeron los gráficos de redis. Se puede ver que la cantidad de queries disminuyo con respecto a la primera prueba.

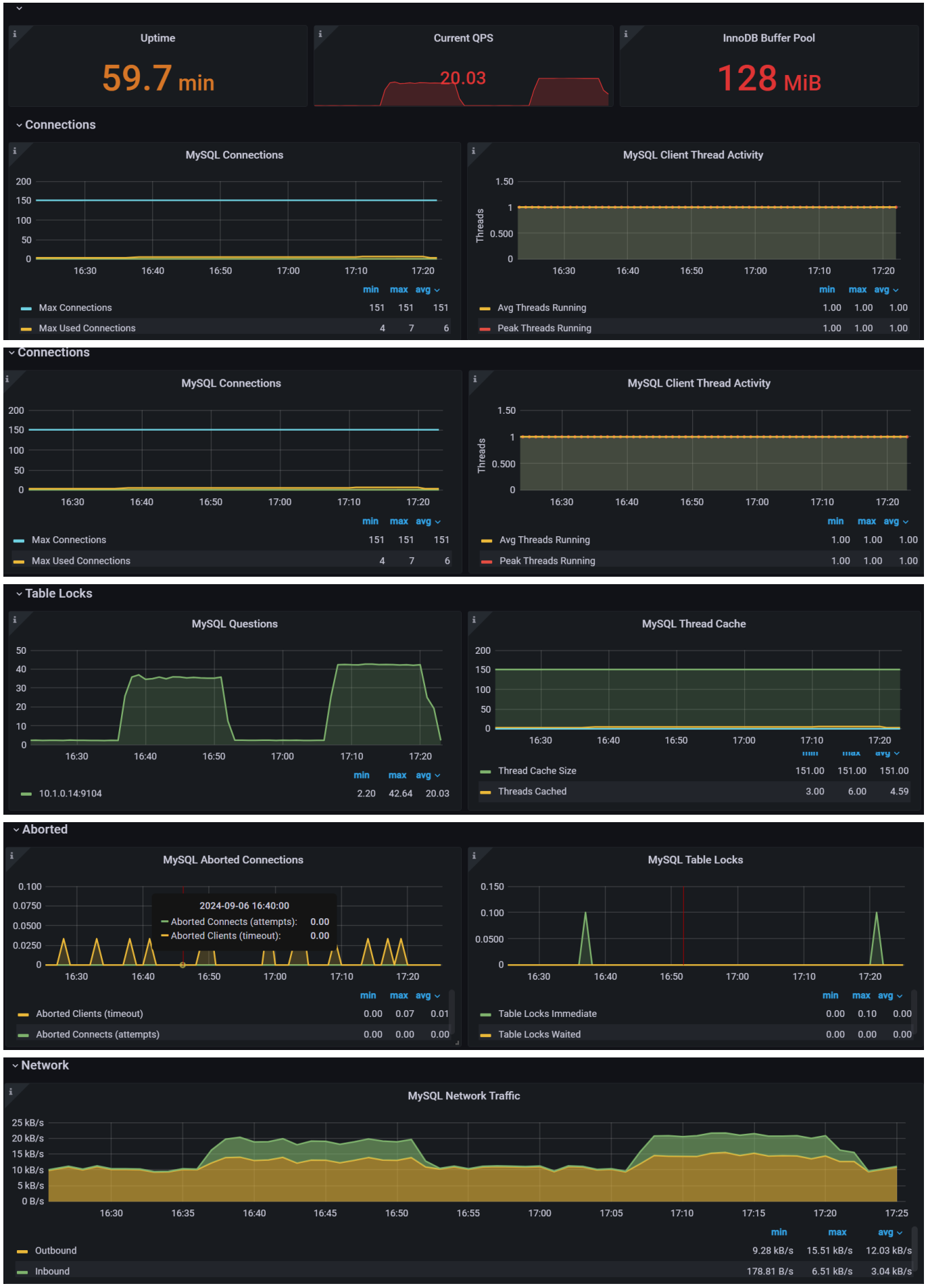
Tercera Prueba:

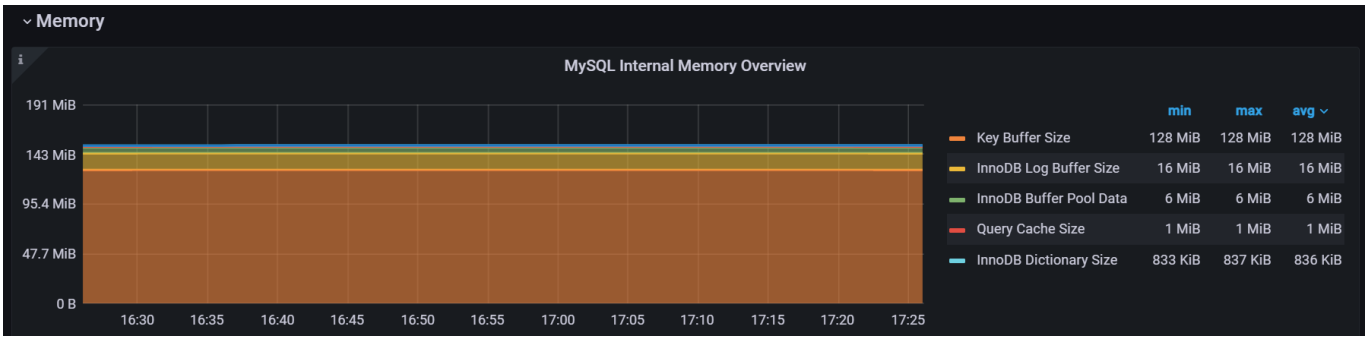
Esta prueba es con memcached

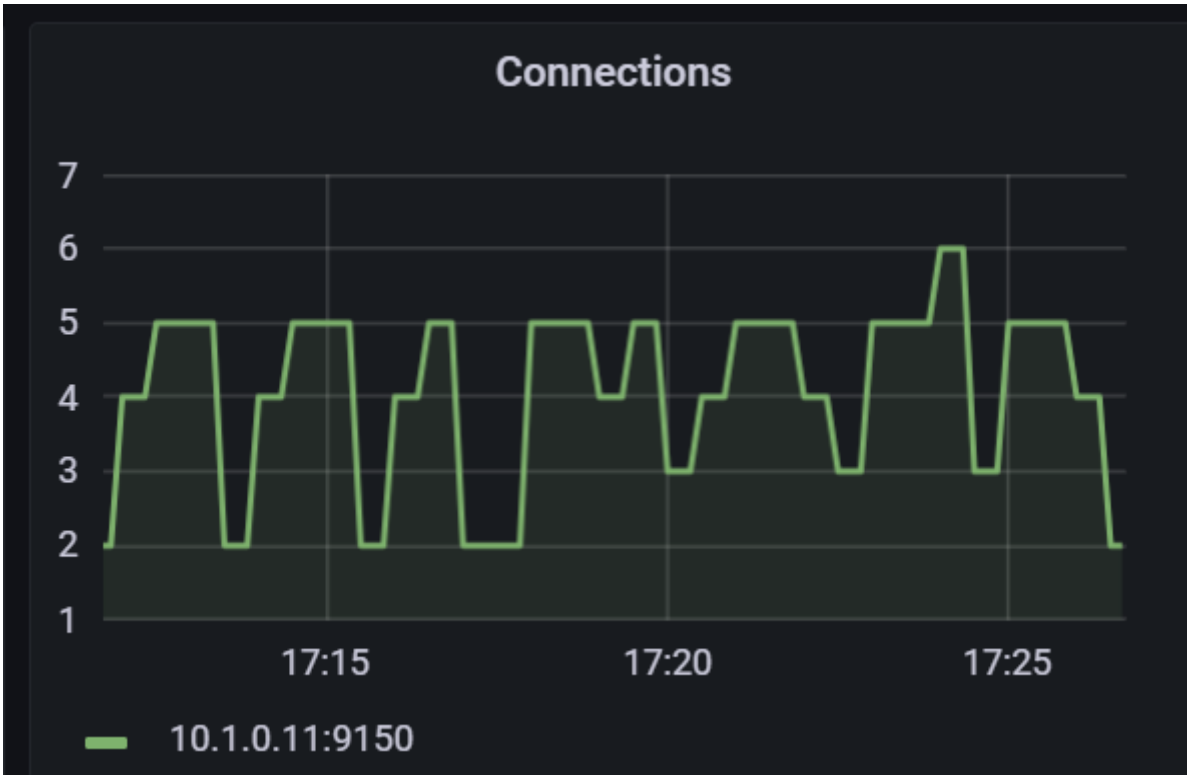
```
public class MariaDBLoadTestWithMemcached extends Simulation {

    // Definición de los diferentes endpoints, incluyendo solo el uso de Memcached
    private final String[] endpoints = {
            "/mariadb/movies/rating?cache=memcached",
            "/mariadb/movies/best?cache=memcached",
            "/mariadb/movies/trend?cache=memcached",
            "/mariadb/movies/inconsistent?cache=memcached"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random MariaDB Endpoints Load Test with
Memcached Cache")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Tercera prueba:

Empieza en 17:05







Para esta prueba se incluyeron los gráficos de memcached. Se puede ver que la cantidad de queries disminuyo con respecto a la primera prueba.

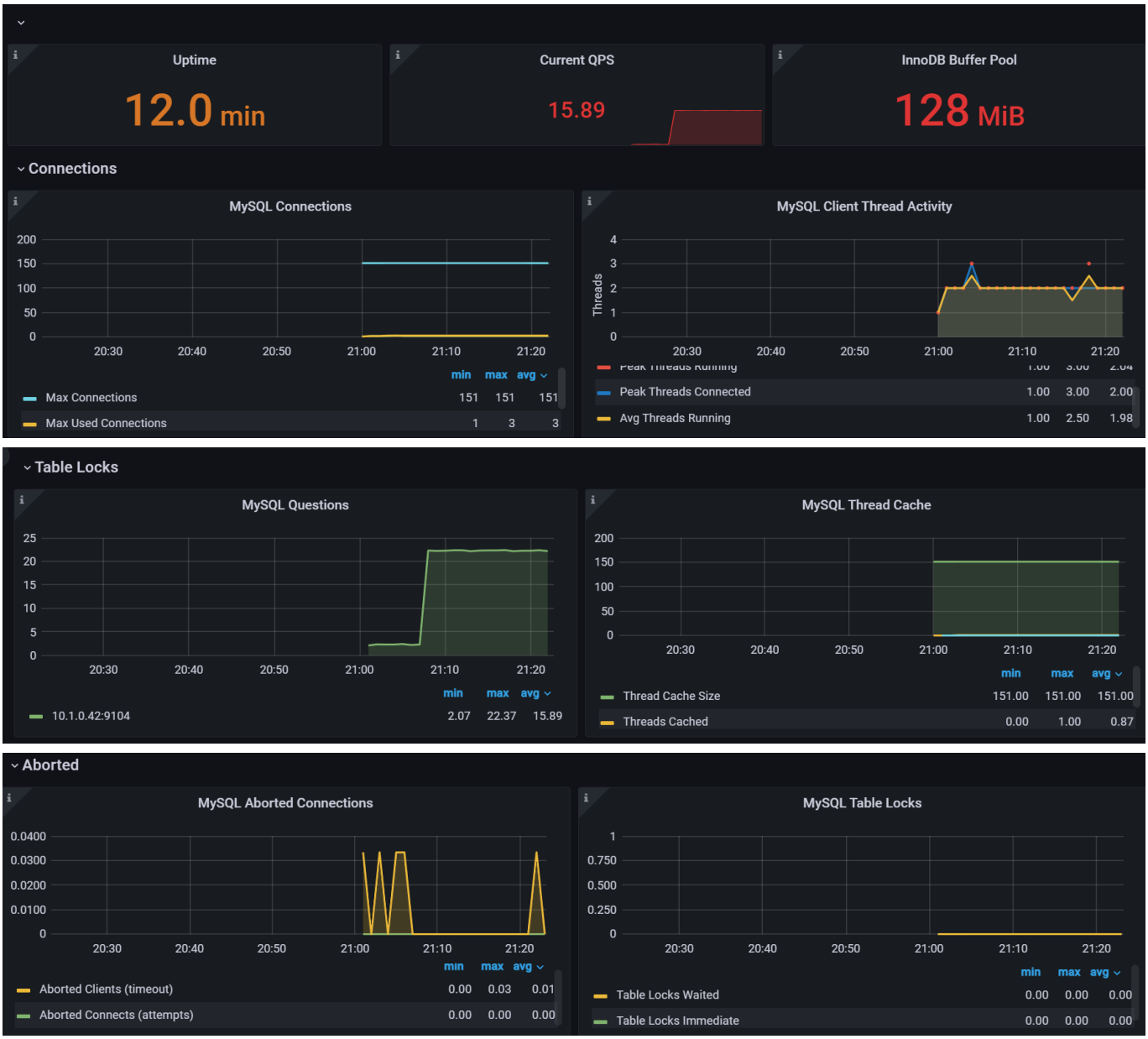
Cuarta Prueba:

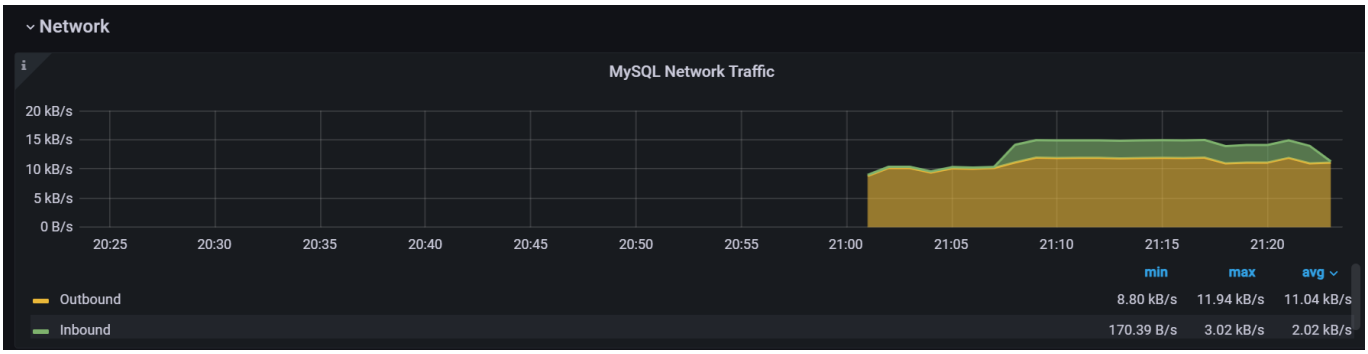
```
public class MariaDBLoadTest2 extends Simulation {

    // Definición de los diferentes endpoints
    private final String[] endpoints = {
            "/mariadb/movies/rating",
            "/mariadb/movies/best",
            "/mariadb/movies/trend",
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random MariaDB Endpoints Load Test")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );
```

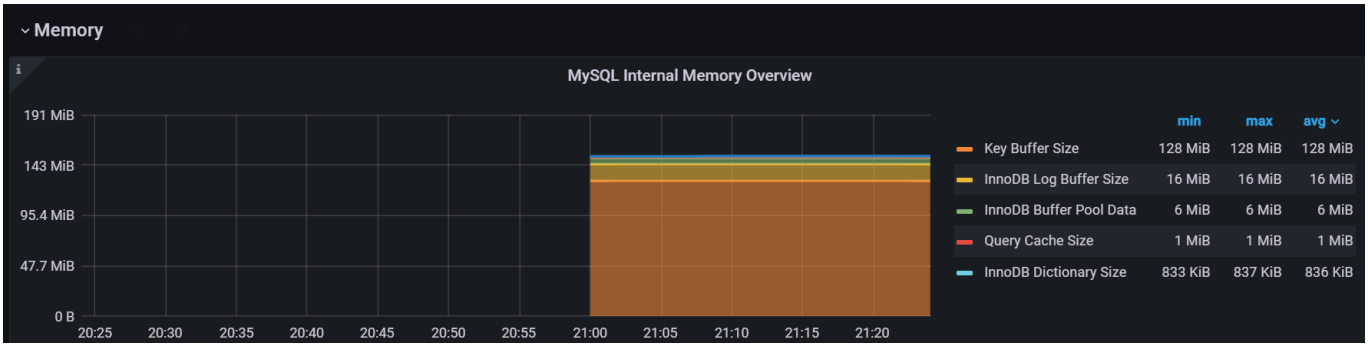
```
    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(5).during(Duration.ofSeconds(900)) //
5 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Cuarta prueba:

Empieza en 21:06







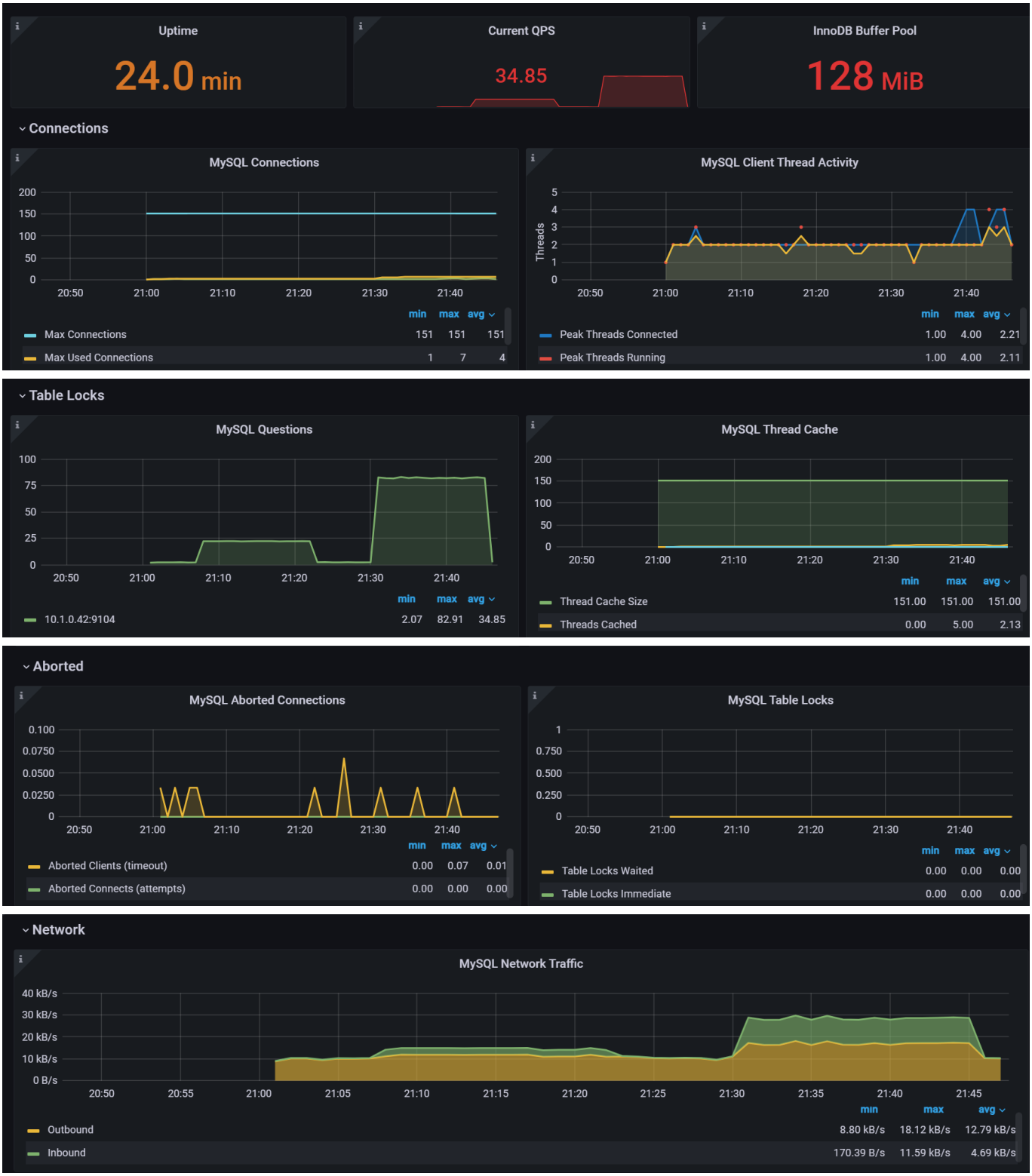
Esta prueba se realiza haciendo la conexión con tres endpoints y con la mitad de usuarios por segundo(5)

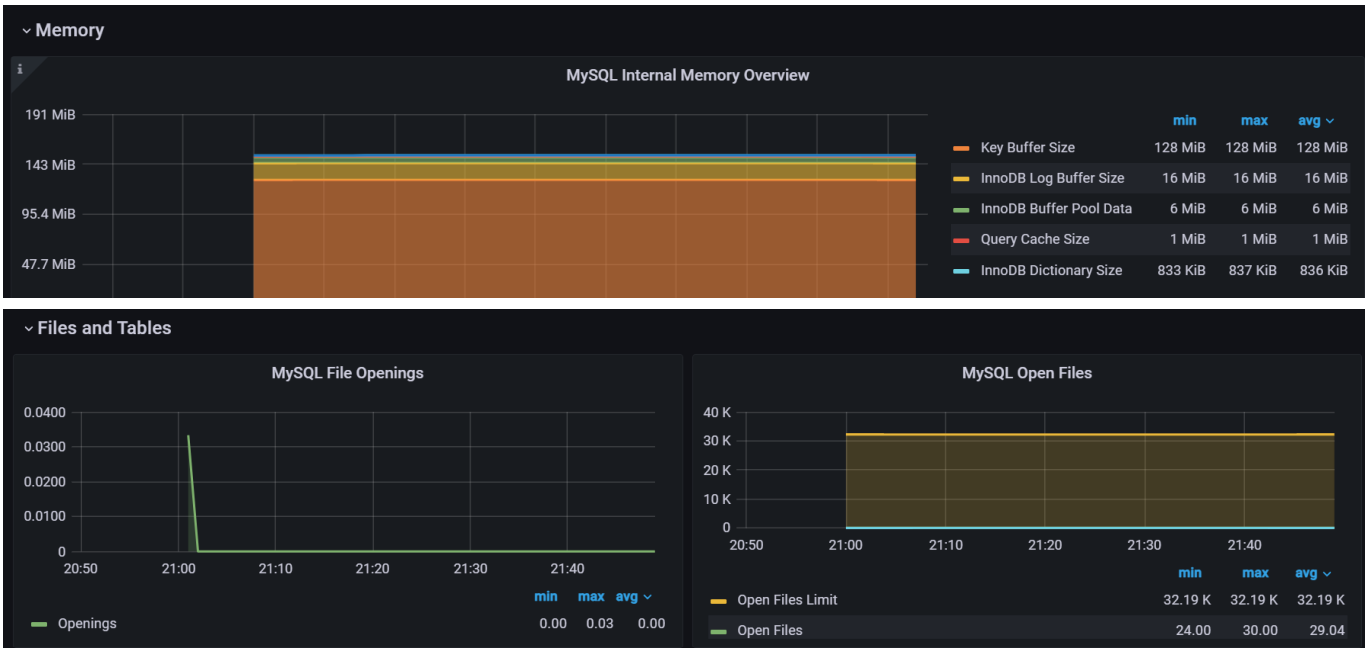
Quinta prueba:

```
public class MariaDBLoadTest3 extends Simulation {

    // Se define el endpoint específico
    private final String endpoint = "/mariadb/movies/rating";

    // Escenario con solicitudes al primer endpoint
    ScenarioBuilder scn = scenario("MariaDB Endpoint Load Test for Rating")
            .exec(http("Rating Endpoint Request")
                    .get(endpoint) // Llama siempre al primer endpoint
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(20).during(Duration.ofSeconds(900))
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Quinta prueba:

Empieza en 21:30





Esta prueba se realiza haciendo la conexión con solo un endpoint específico y simulando un total de 20 usuarios concurrentes.

PostgreSQL

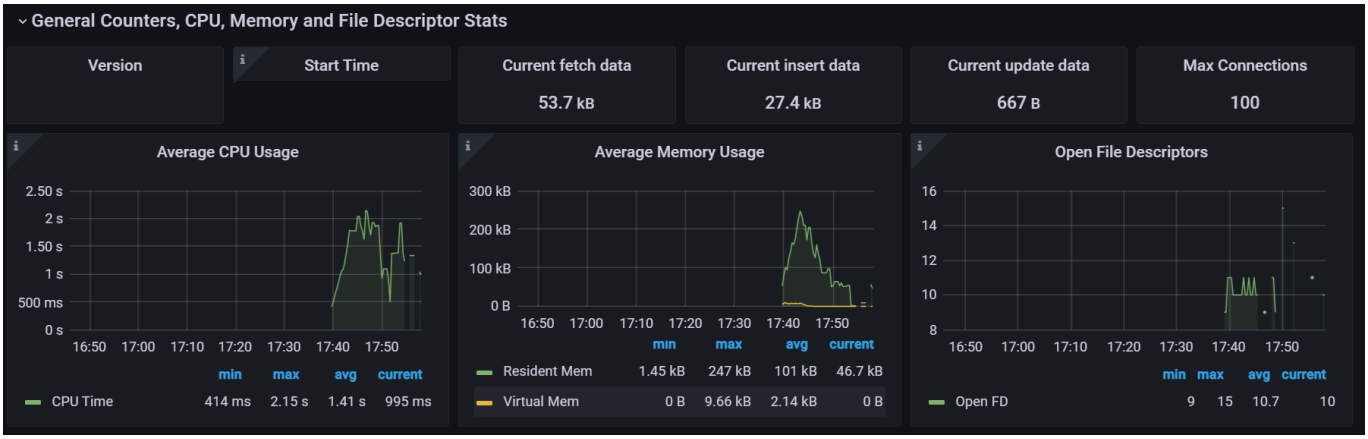
Primera Prueba:

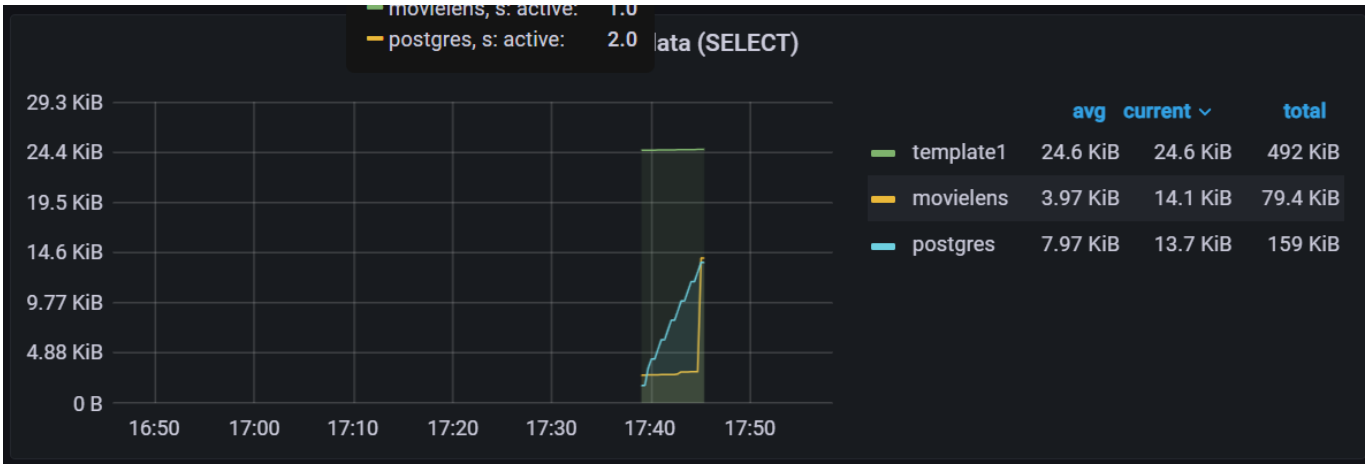
```
public class postgresqlLoadTest extends Simulation {

    // Definición de los diferentes endpoints
    private final String[] endpoints = {
            "/postgresql/movies/rating",
            "/postgresql/movies/best",
            "/postgresql/movies/trend",
            "/postgresql/movies/inconsistent"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random postgresql Endpoints Load Test")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:51674")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
```

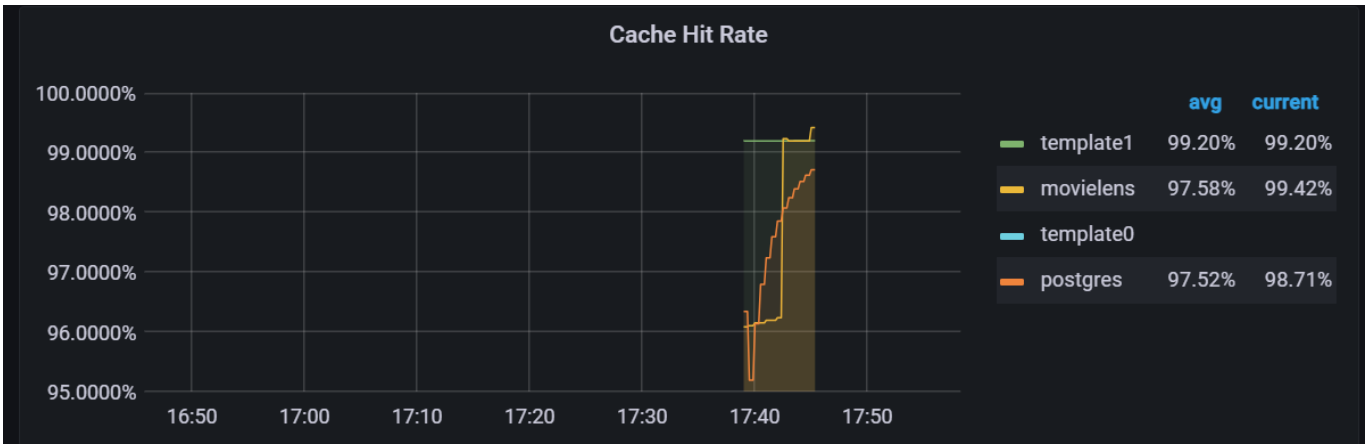
```
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
  10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Primera prueba:

Empieza en 17:45







Segunda Prueba:

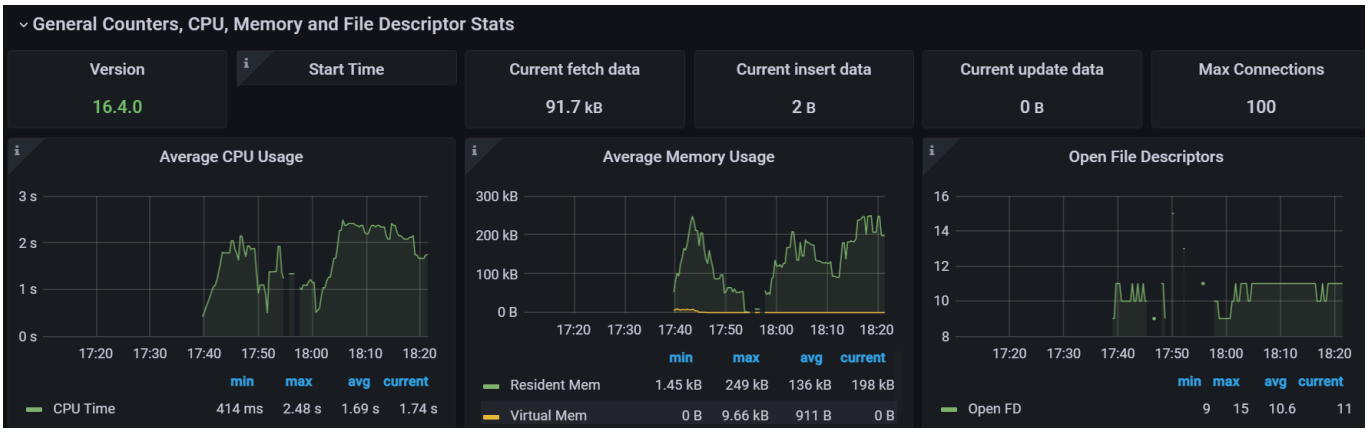
Esta prueba es con redis

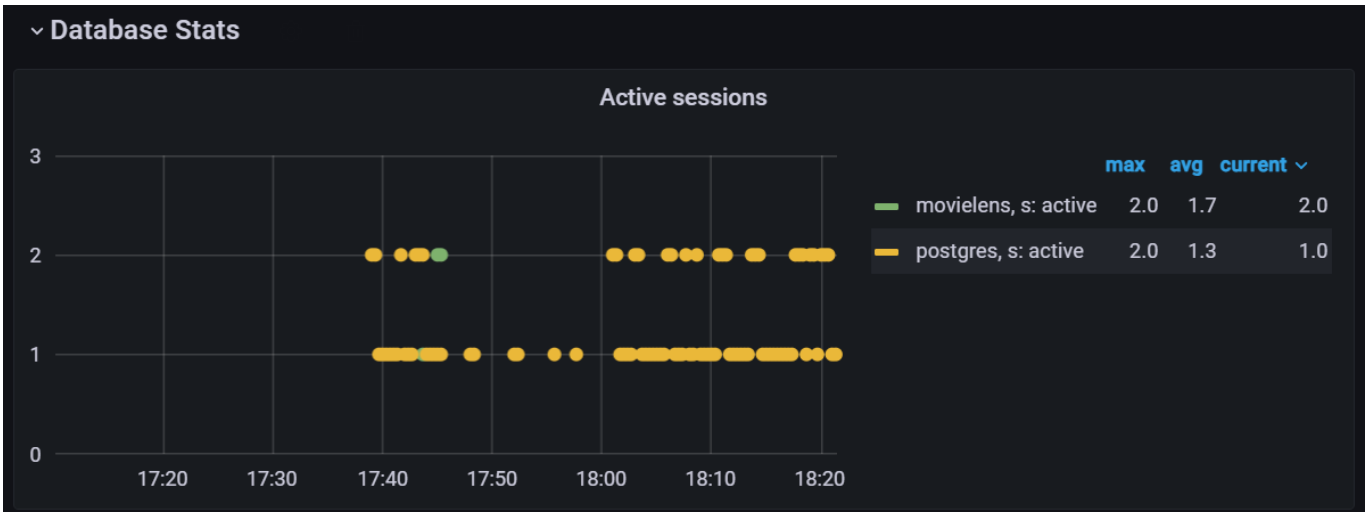
```
public class postgresqlLoadTestWithRedis extends Simulation {

    // Definición de los diferentes endpoints, incluyendo el uso de Redis
    private final String[] endpoints = {
            "/postgresql/movies/rating?cache=redis",
            "/postgresql/movies/best?cache=redis",
            "/postgresql/movies/trend?cache=redis",
            "/postgresql/movies/inconsistent?cache=redis",
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random postgresql Endpoints Load Test with
Redis Cache")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
```

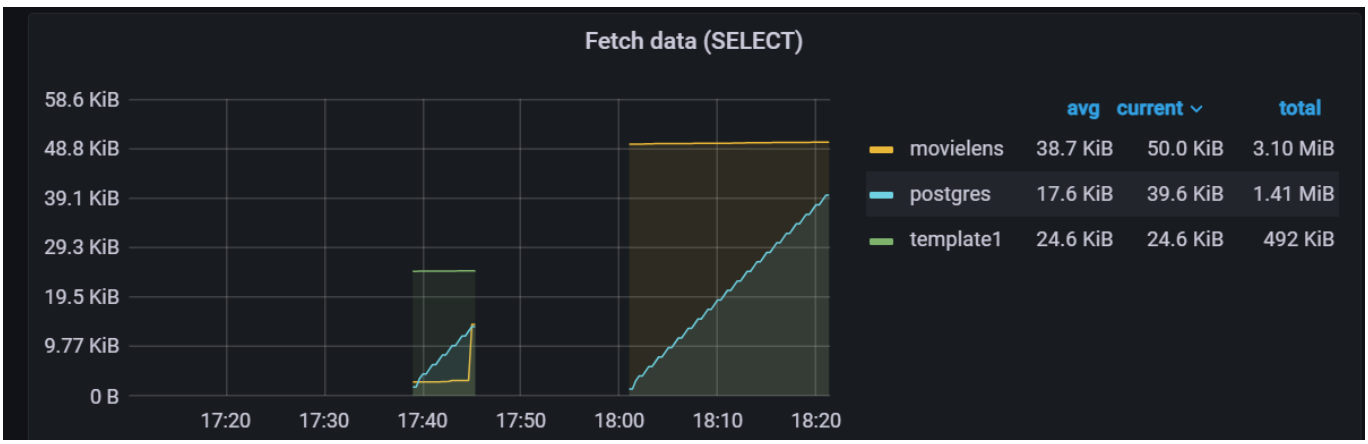
```
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Segunda prueba:

Empieza en 18:05







Se incluyen los gráficos de redis

Tercera Prueba:

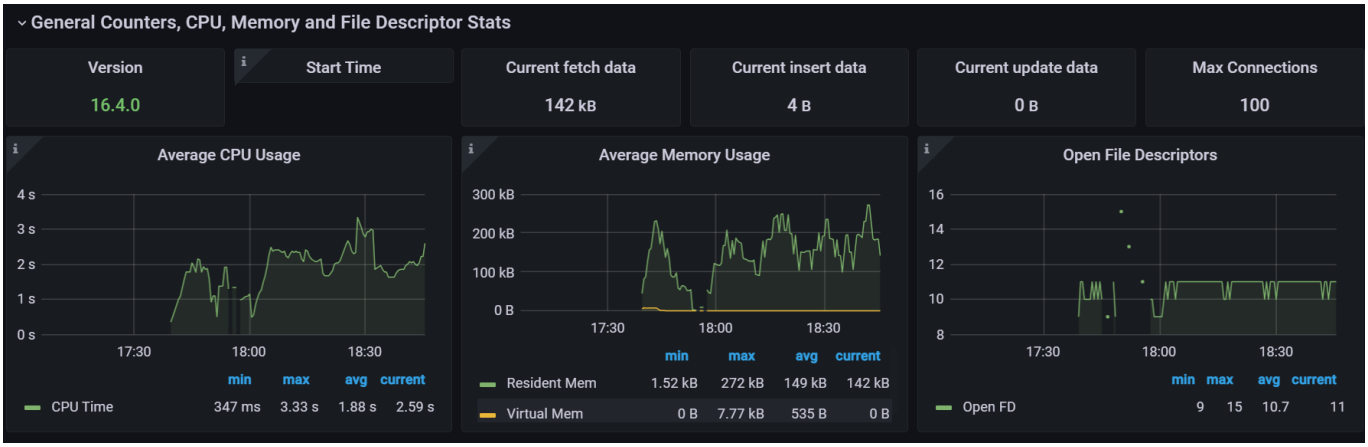
Esta prueba es con memcached

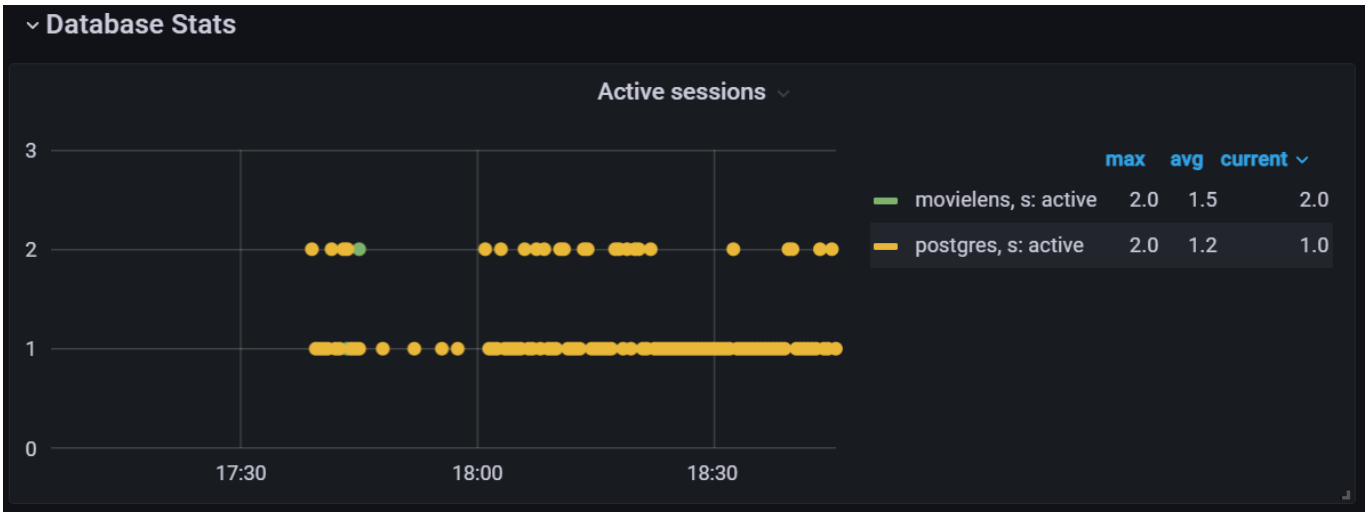
```
public class postgresqlLoadTestWithMemcached extends Simulation {

    // Definición de los diferentes endpoints, incluyendo solo el uso de Memcached
    private final String[] endpoints = {
            "/postgresql/movies/rating?cache=memcached",
            "/postgresql/movies/best?cache=memcached",
            "/postgresql/movies/trend?cache=memcached",
            "/postgresql/movies/inconsistent?cache=memcached"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random postgresql Endpoints Load Test with
Memcached Cache")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
```

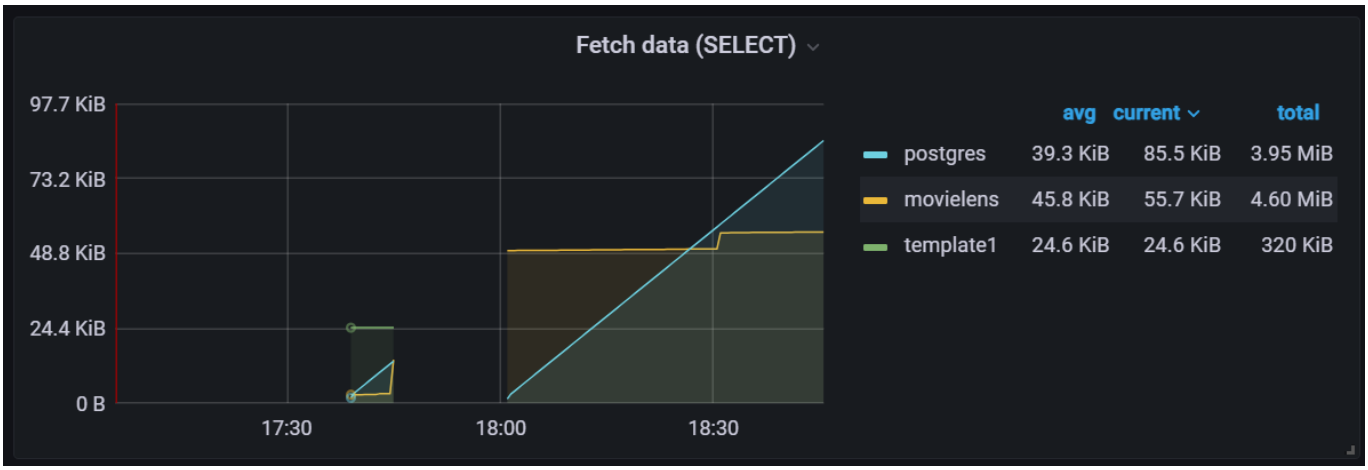
```
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

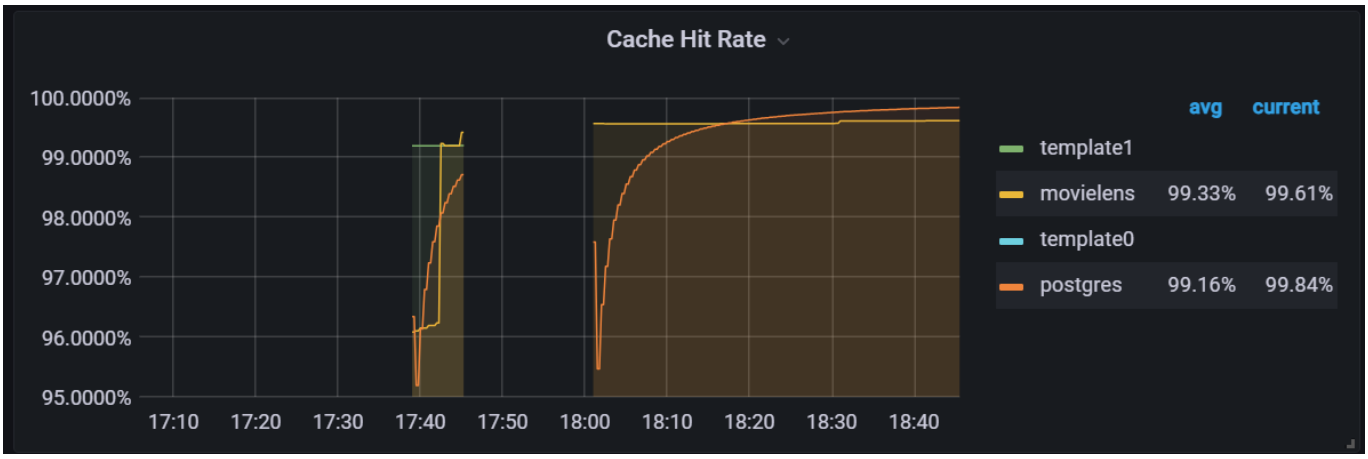
Resultados de la Tercera prueba:

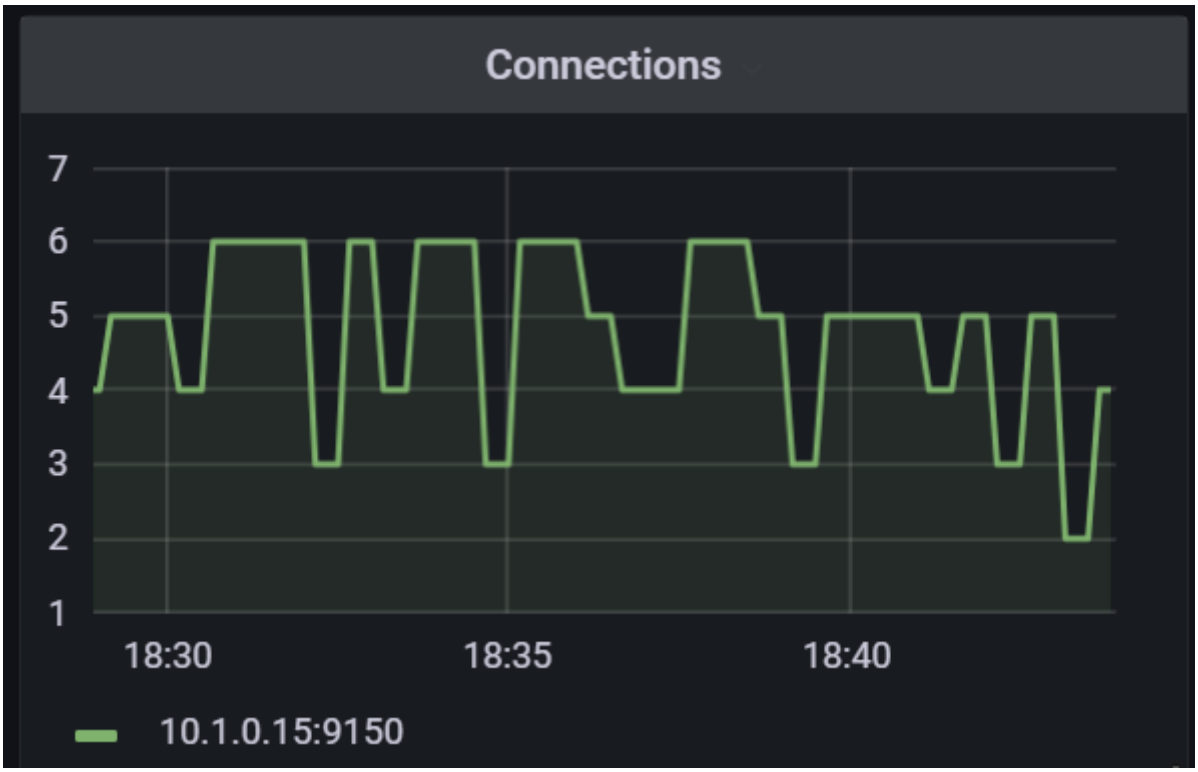
Empieza en 18:29

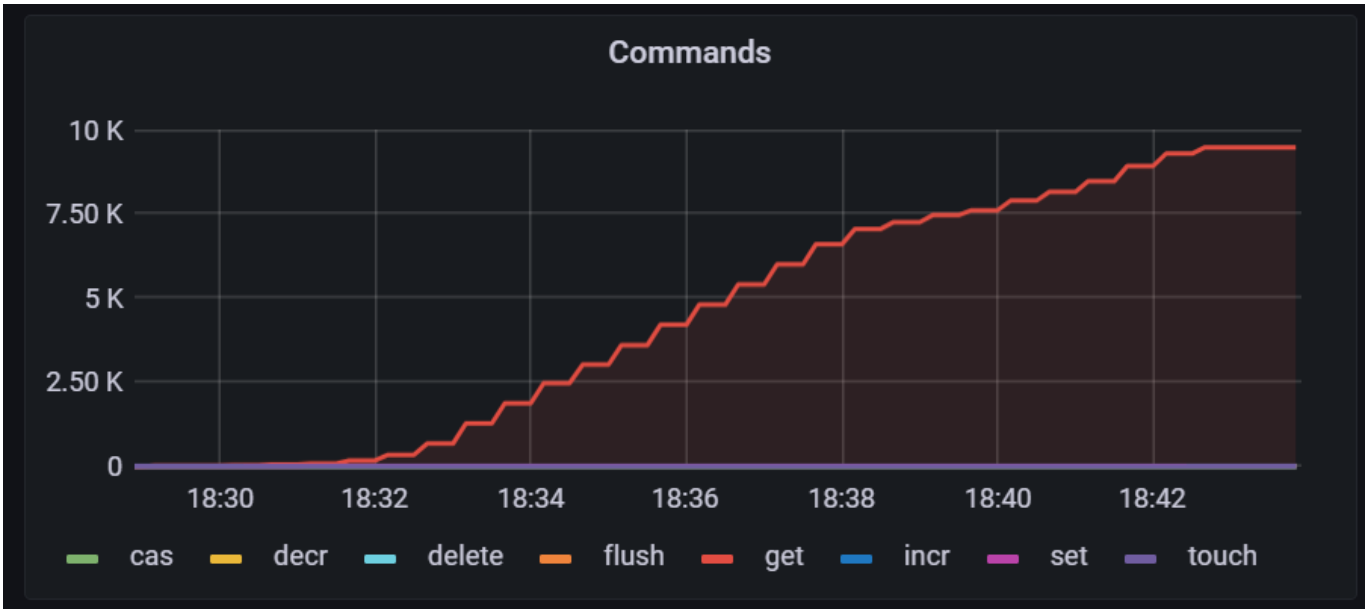





Fetch data (SELECT)
97.7 KiB
73.2 KiB
48.8 KiB
24.4 KiB
0 B
17:30
18:00
18:30
avg current total
postgres 39.3 KiB 85.5 KiB 3.95 MiB
movielens 45.8 KiB 55.7 KiB 4.60 MiB
template1 24.6 KiB 24.6 KiB 320 KiB


Cache Hit Rate
100.0000%
99.0000%
98.0000%
97.0000%
96.0000%
95.0000%
17:10
17:20
17:30
17:40
17:50
18:00
18:10
18:20
18:30
18:40
avg current
template1
movielens 99.33% 99.61%
template0
postgres 99.16% 99.84%


Connections
7
6
5
4
3
2
1
18:30
18:35
18:40
10.1.0.15:9150



Se incluyen los gráficos de redis

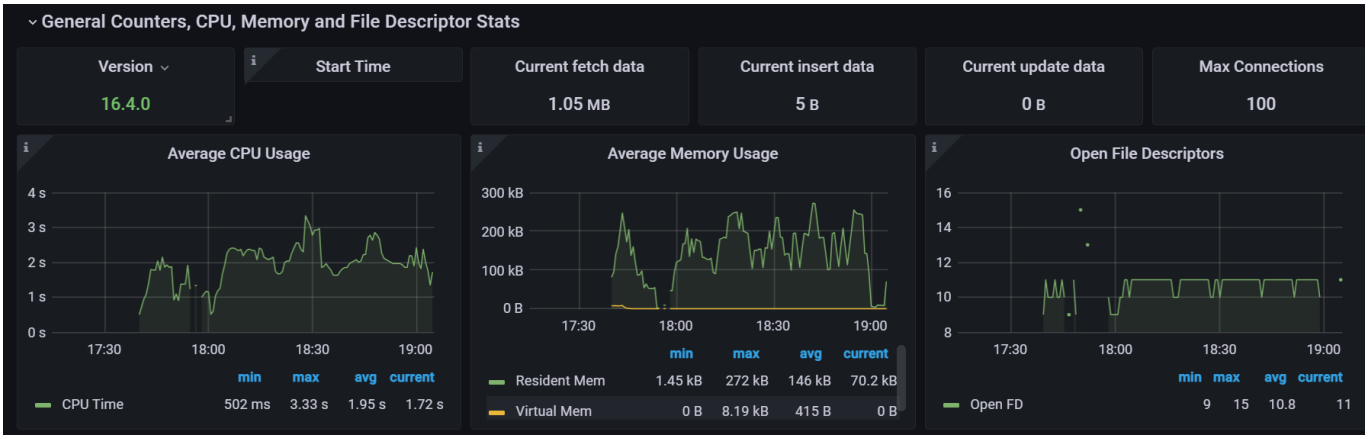
Cuarta Prueba:

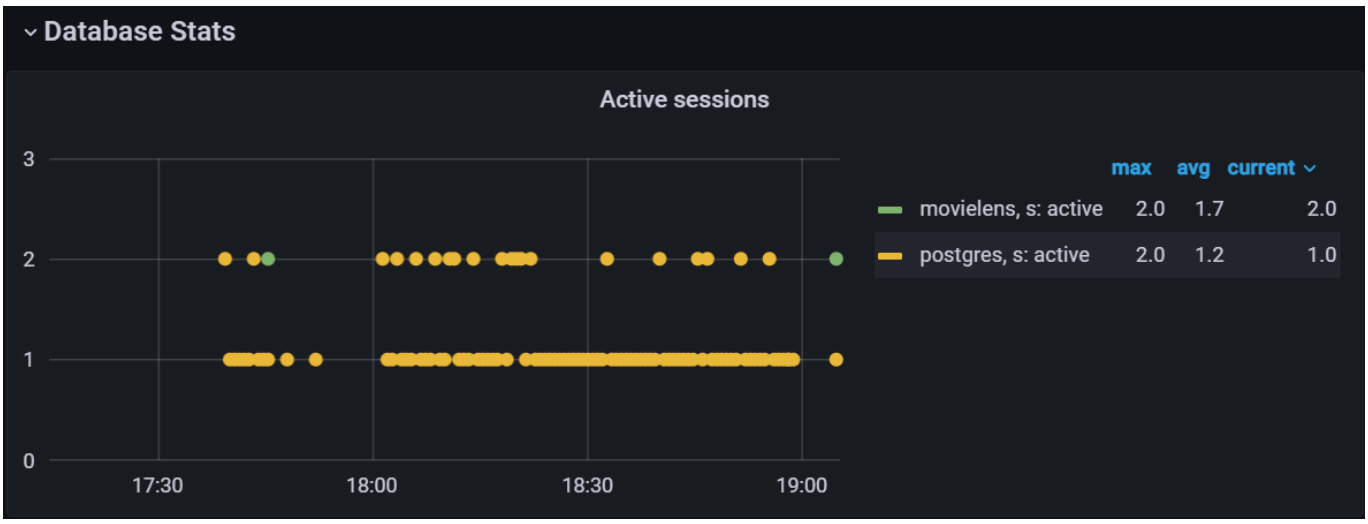
```
public class postgresqlLoadTest2 extends Simulation {

    // Definición de los diferentes endpoints
    private final String[] endpoints = {
            "/postgresql/movies/rating",
            "/postgresql/movies/best",
            "/postgresql/movies/trend",
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random postgresql Endpoints Load Test")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(5).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
```

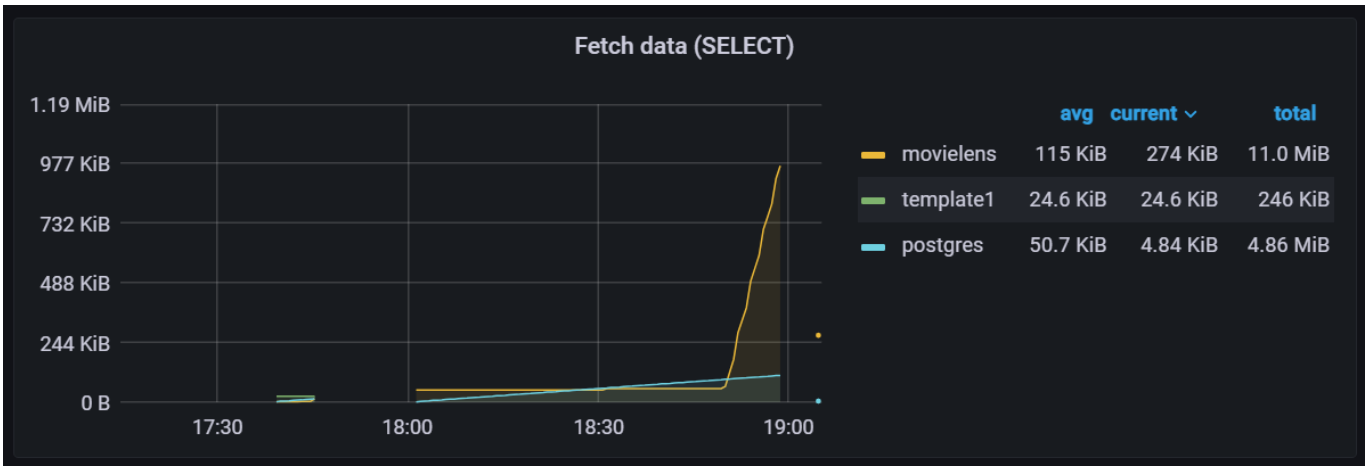
```
        }
    }
```

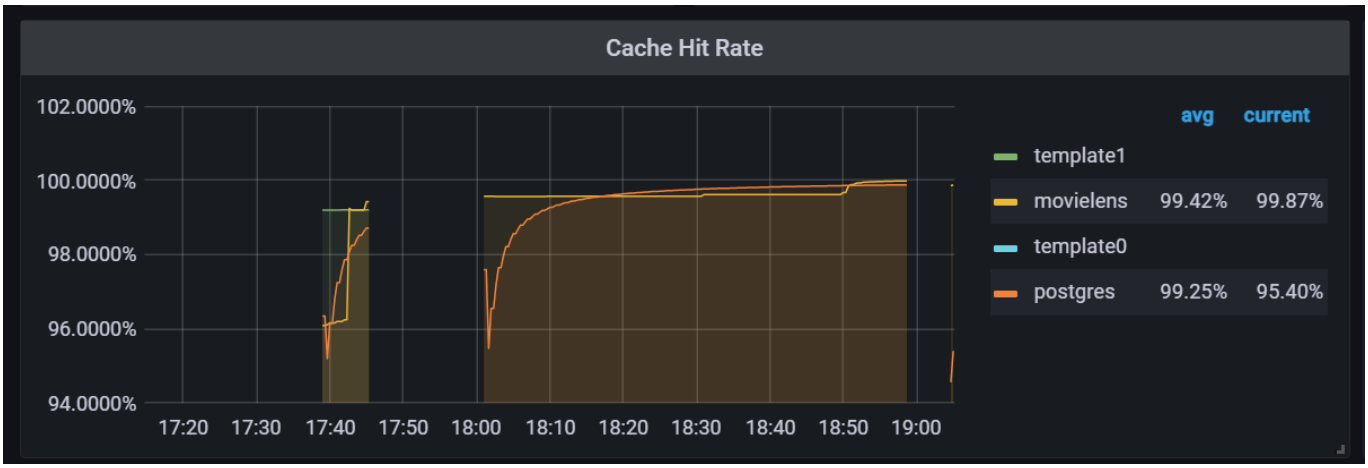
Resultados de la Cuarta prueba:

Empieza en 18:50









Esta prueba se realiza haciendo la conexión con tres endpoints y con la mitad de usuarios por segundo(5)

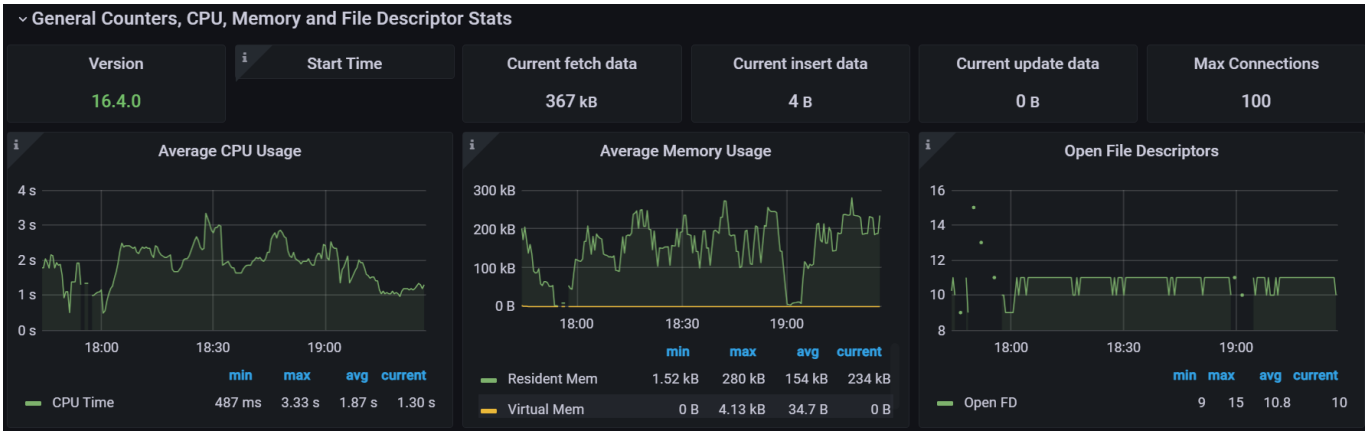
Quinta prueba:

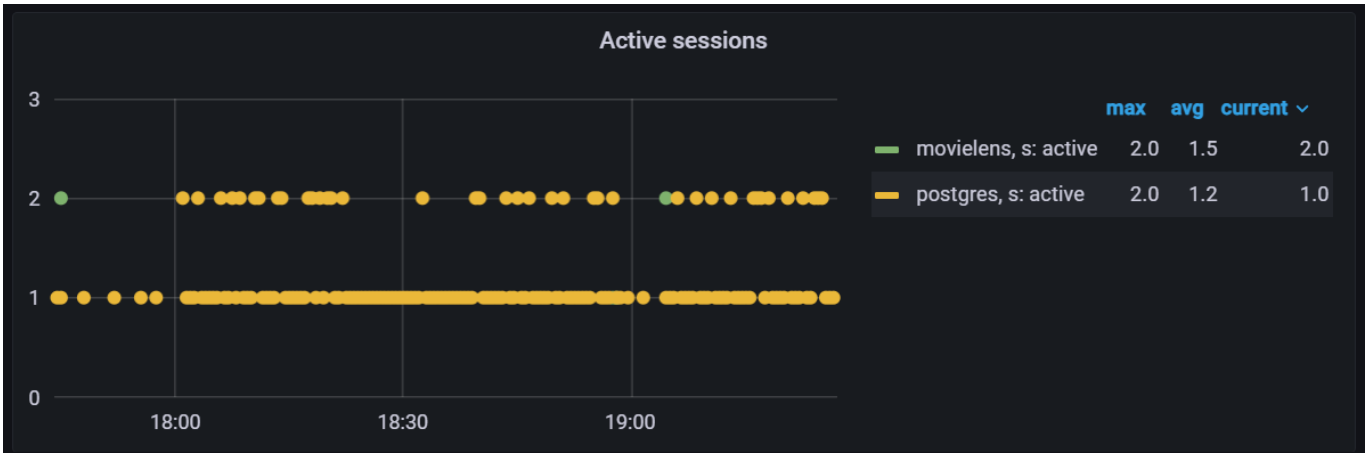
```
public class postgresqlLoadTest3 extends Simulation {

    // Se define el endpoint específico
    private final String endpoint = "/postgresql/movies/rating";

    // Escenario con solicitudes al primer endpoint
    ScenarioBuilder scn = scenario("postgresql Endpoint Load Test for Rating")
            .exec(http("Rating Endpoint Request")
                    .get(endpoint) // Llama siempre al primer endpoint
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(20).during(Duration.ofSeconds(900))
```

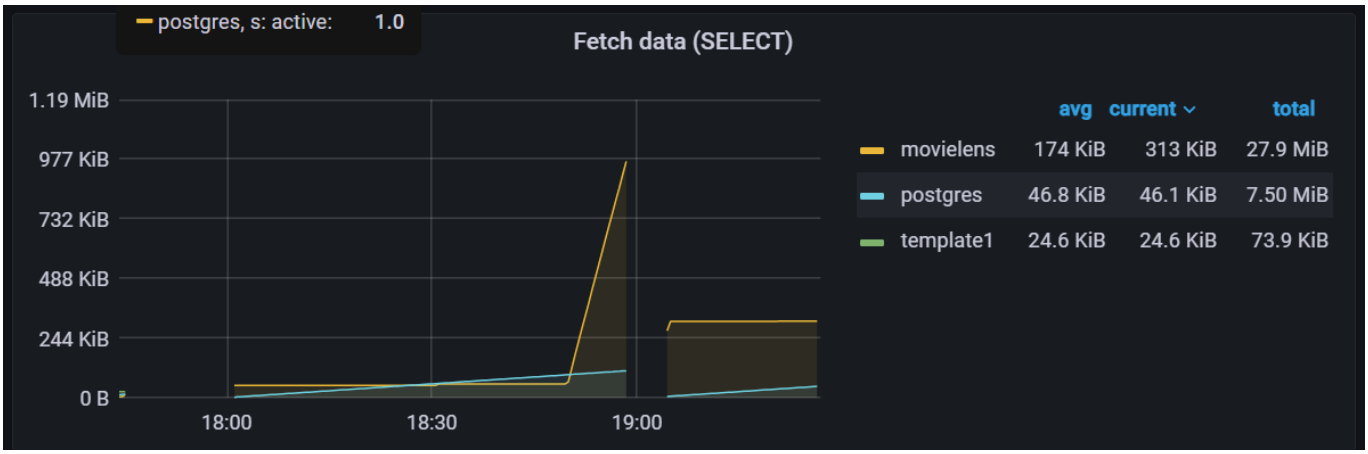
```
                ).protocols(httpProtocol)
        );
    }
}
```

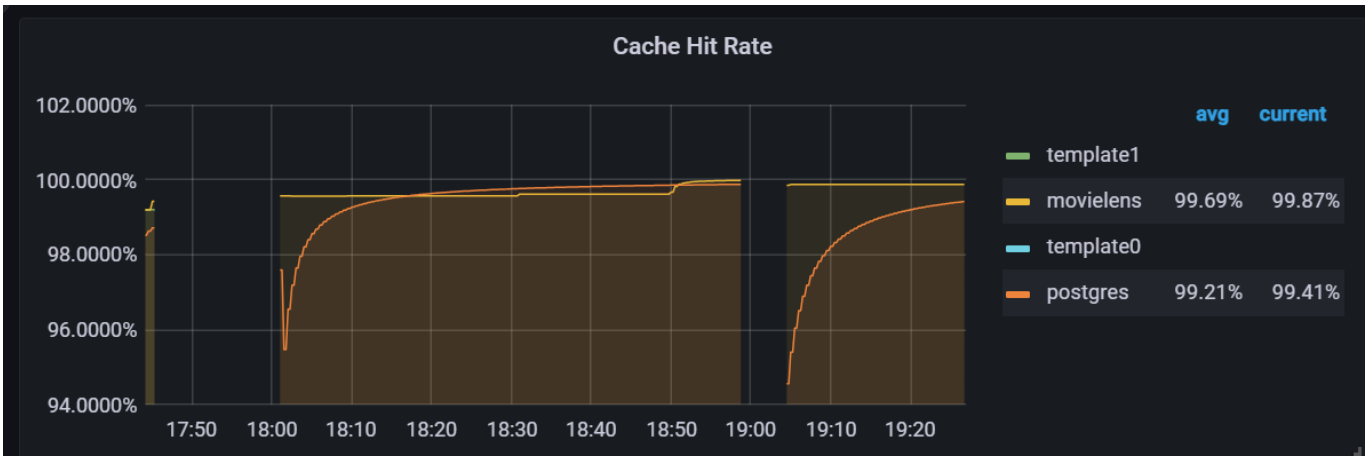
Resultados de la Primera prueba:

Empieza en 19:10









Esta prueba se realiza haciendo la conexión con solo un endpoint específico y simulando un total de 20 usuarios concurrentes.

Elastic Search

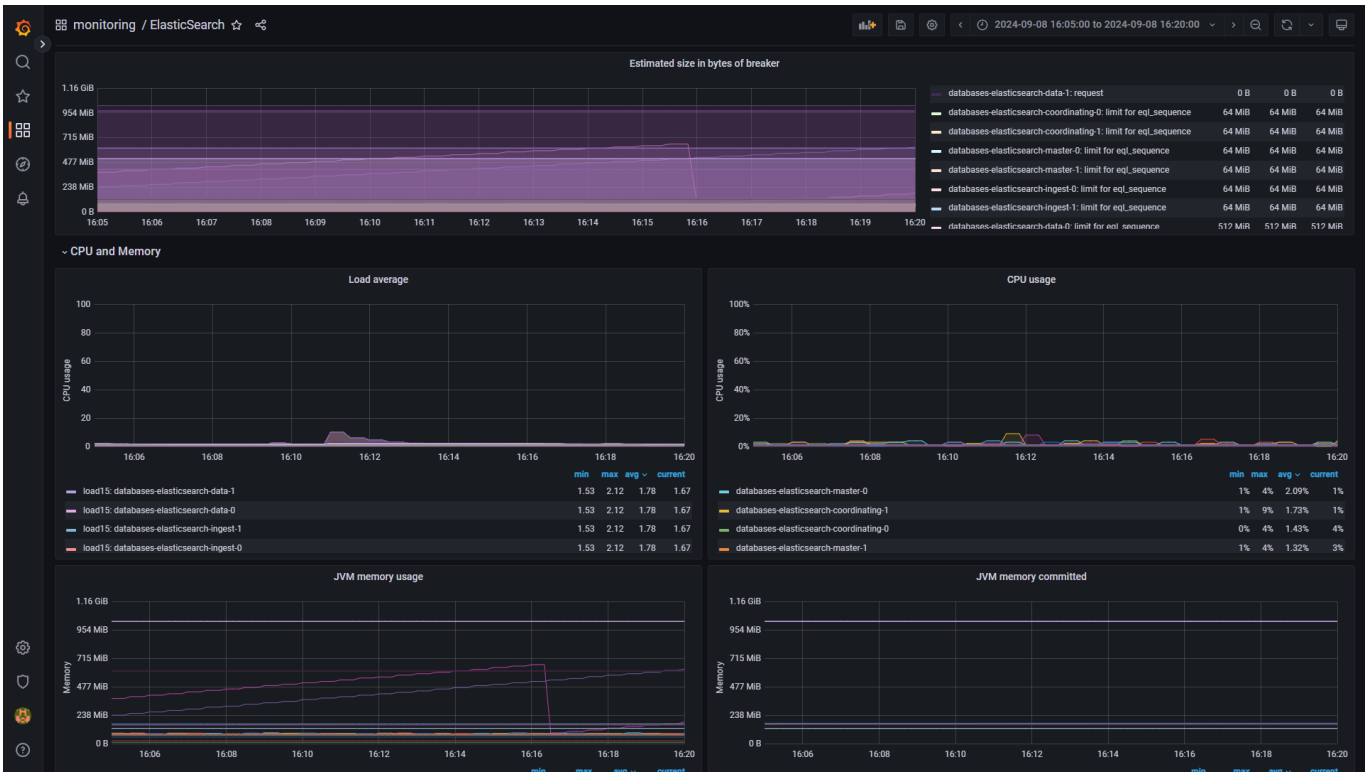
Primera Prueba:

```
public class elasticsearchLoadTest extends Simulation {

    // Definición de los diferentes endpoints
    private final String[] endpoints = {
            "/elasticsearch/movies/rating",
            "/elasticsearch/movies/best",
            "/elasticsearch/movies/trend",
            "/elasticsearch/movies/inconsistent"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random elasticsearch Endpoints Load Test")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
```

```
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:51674")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Primera prueba:

Empieza en 16:05



Segunda Prueba:

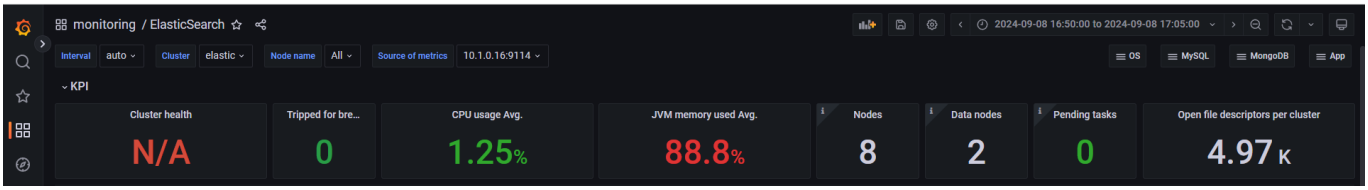
Esta prueba es con redis

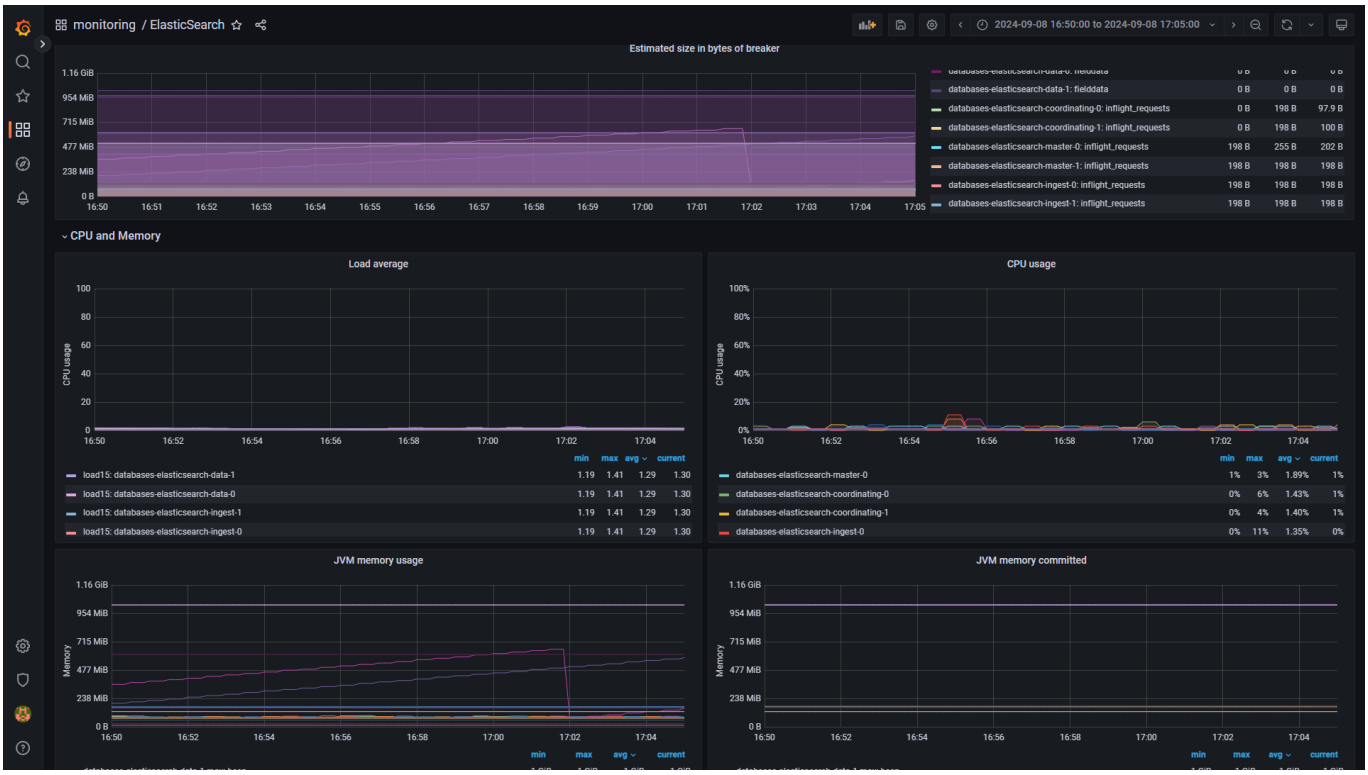
```
public class elasticsearchLoadTestWithRedis extends Simulation {
```

```
    // Definición de los diferentes endpoints, incluyendo el uso de Redis
    private final String[] endpoints = {
            "/elasticsearch/movies/rating?cache=redis",
            "/elasticsearch/movies/best?cache=redis",
            "/elasticsearch/movies/trend?cache=redis",
            "/elasticsearch/movies/inconsistent?cache=redis",
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random elasticsearch Endpoints Load Test with
Redis Cache")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Segunda prueba:

Empieza en 16:50





Tercera Prueba:

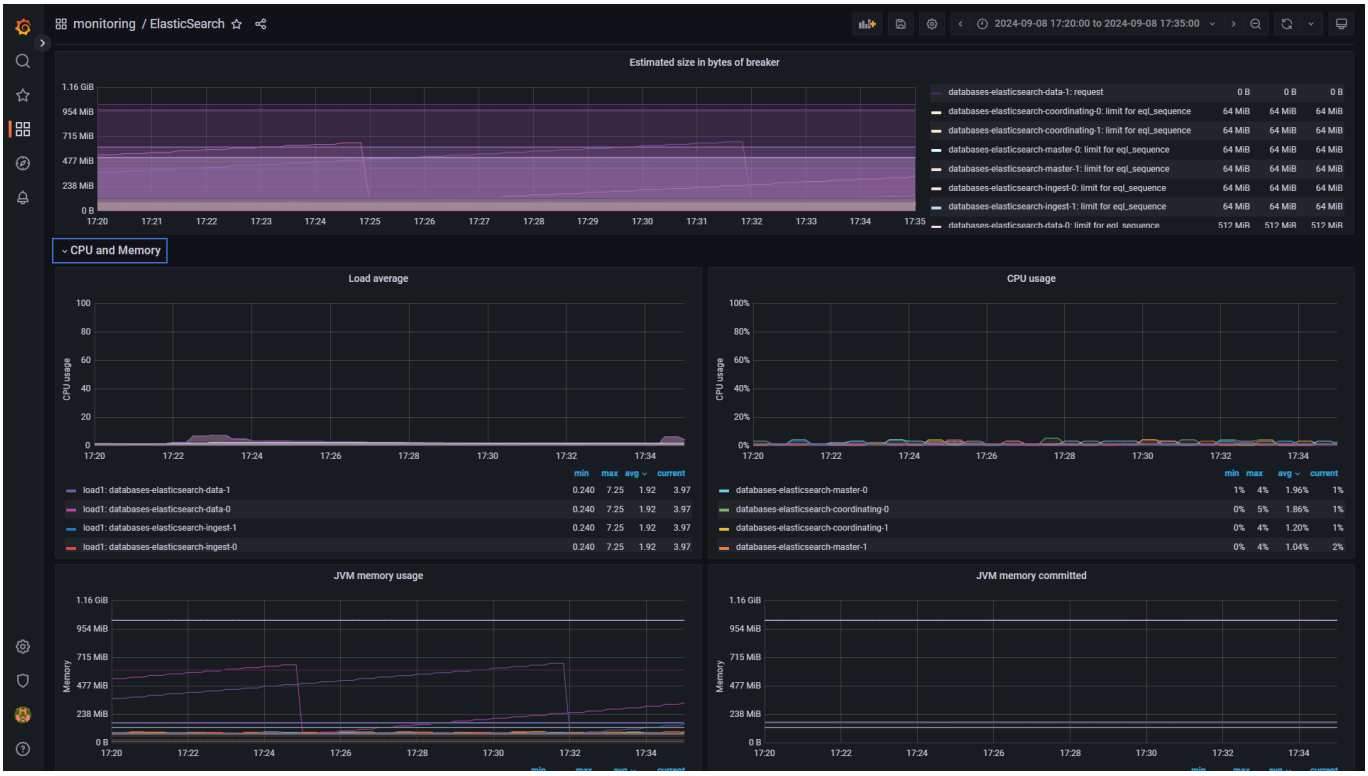
Esta prueba es con memcached

```
public class elasticsearchLoadTestWithMemcached extends Simulation {

    // Definición de los diferentes endpoints, incluyendo solo el uso de Memcached
    private final String[] endpoints = {
            "/elasticsearch/movies/rating?cache=memcached",
            "/elasticsearch/movies/best?cache=memcached",
            "/elasticsearch/movies/trend?cache=memcached",
            "/elasticsearch/movies/inconsistent?cache=memcached"
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random elasticsearch Endpoints Load Test with
Memcached Cache")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
```

```
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(10).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Tercera prueba:

Empieza en 17:20



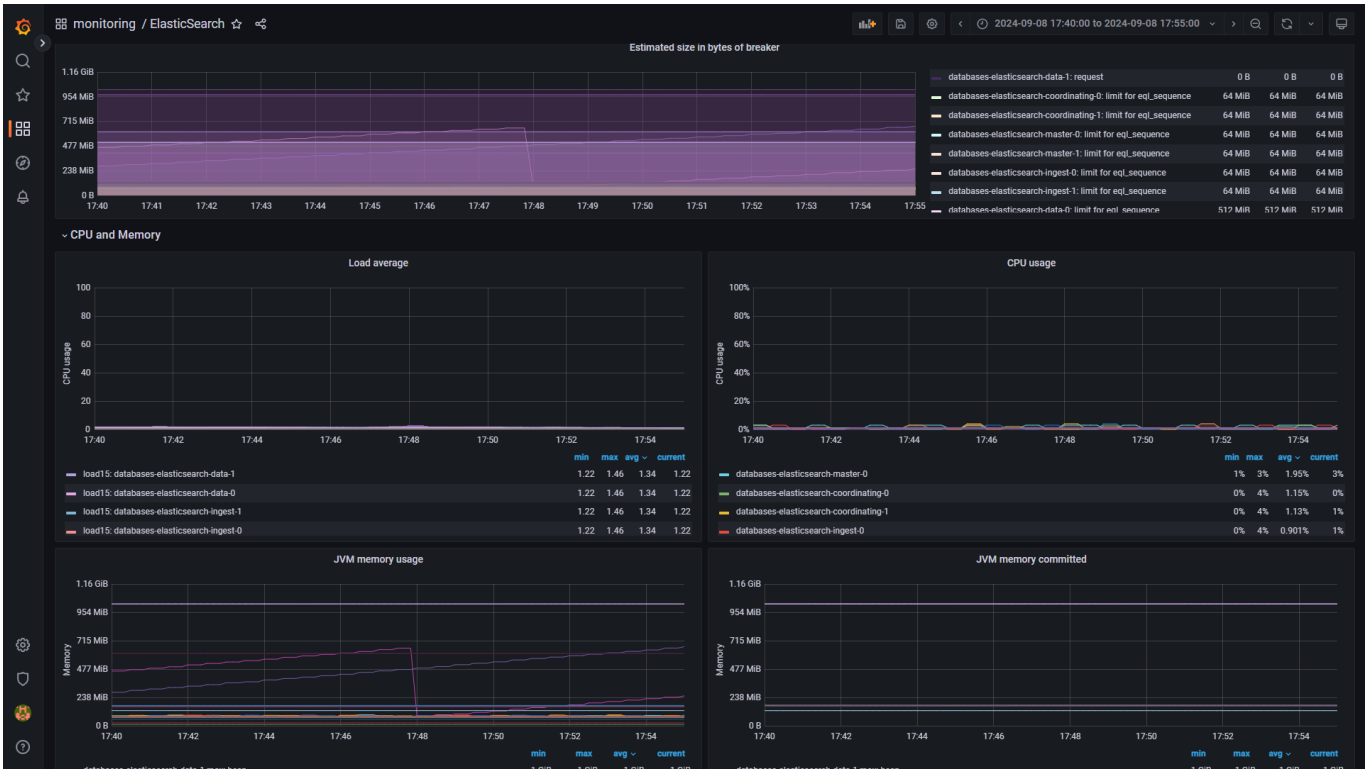
Cuarta Prueba:

```
public class elasticsearchLoadTest2 extends Simulation {

    // Definición de los diferentes endpoints
    private final String[] endpoints = {
            "/elasticsearch/movies/rating",
            "/elasticsearch/movies/best",
            "/elasticsearch/movies/trend",
    };

    // Escenario con solicitudes aleatorias a los endpoints
    ScenarioBuilder scn = scenario("Random elasticsearch Endpoints Load Test")
            .exec(http("Random Endpoint Request")
                    .get(session ->
endpoints[ThreadLocalRandom.current().nextInt(endpoints.length)])
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");
```

```
        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(5).during(Duration.ofSeconds(900)) //
10 usuarios por segundo durante 15 minutos
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Cuarta prueba:

Empieza en 17:40



Quinta prueba:

```
public class MariaDBLoadTest3 extends Simulation {

    // Se define el endpoint específico
    private final String endpoint = "/elasticsearch/movies/rating";

    // Escenario con solicitudes al primer endpoint
    ScenarioBuilder scn = scenario("elasticsearch Endpoint Load Test for Rating")
            .exec(http("Rating Endpoint Request")
                    .get(endpoint) // Llama siempre al primer endpoint
```

```
                    .check(status().is(200))
            );

    {
        // Configuración de HTTP
        HttpProtocolBuilder httpProtocol = http
                .baseUrl("http://localhost:49268")
                .acceptHeader("application/json");

        // Simulación de usuarios concurrentes
        setUp(
                scn.injectOpen(
                        constantUsersPerSec(20).during(Duration.ofSeconds(900))
                ).protocols(httpProtocol)
        );
    }
}
```

Resultados de la Quinta prueba:

Empieza en 18:25