Отчёт о нагрузочном тестировании приложения

Сергей Герасимов

2023-02-09

Содержание

1 Подготовка

Подготовка аналогична прошлому отчёту:

- Сгенерируем 1 миллион пользователей по API (через `/user/register` ручку)
- Создадим дамп базы, загрузим его на сервер
- Добавим отдельный `docker-compose.yml`, формирующий тестовое окружение
- *Новое* Добавим поддержку кластера Percona в сервис
- *Новое* Включим GTID, row-based репликацию в полу-синхронном режиме
- *Новое* Переведём запросы на чтение пользователей на реплику №1
- *Новое* Добавим сервис фиче-флагов, позволяющий включать и отключать репликацию
- При запуске тестового окружения, если дамп ещё не развёрнут, скачаем и развернём его

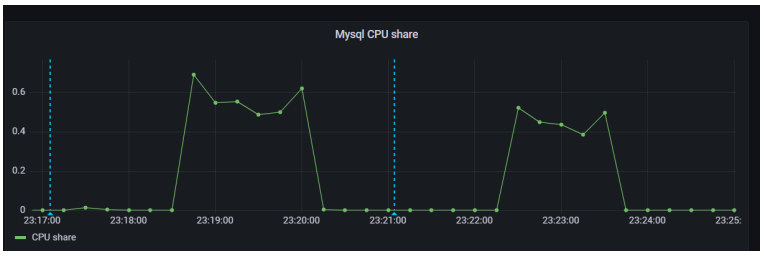
Для замера *throughput* и *latency* использованы **Prometheus** и **Grafana**. Для замера метрик приложения использован стандартный дашборд Grafana - **Jvm Micrometer**. Настройки дашбордов графаны, а также скрипт для генерации тестовых данных указаны в репозитории в `src/test/resources/`

2 Проверка нагрузки на мастер

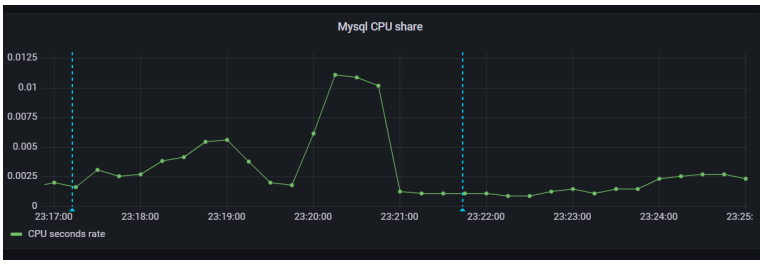
Replica off	Max CPU total second rate	0.01
Replica off	Max CPU share	0.7
Replica on	Max CPU total second rate	0.002
Replica on	Max CPU share	0.52

2.1 Графики

Max CPU total second rate (рост нагрузки на процессор)



Max CPU share (доля загрузки ядер)



Слева на графиках отображён тест с выключенным чтением с реплики, справа - с включённым.

Профиль нагрузки во всех случаях: 256 потоков, отправка поочереёдно пакетов из 10000, 20000, 50000 запросов. Видим, что в пике CPU на мастере при нагрузке всё равно высок (хотя и меньше на 20 п.п.), однако скорость прироста времени CPU на порядок ниже.

3 Тест пропадания мастера

- Создадим искусственную таблицу, которая инкрементирует и отдаёт клиенту счётчик при вставке (для упрощения эксперимента) и контроллер для работы с ней
- Создадим тест (`SyntheticLoadTest.kt`) на таблицу, который будет с интервалом в 200мс обращаться к контроллеру выше
- Запустим тест
- Убиваем мастер на ноде по docker kill
- Переключаем мастер на реплику-2
- Проверяем статус

3.1 Проверка результатов

Переключаем мастер на реплику-2.

```
CHANGE MASTER TO MASTER_HOST = 'percona-replica-2'
```

Получим данные о статусе. Убеждаемся, что статус кластера Primary и размер кластера 2.

```
SHOW STATUS LIKE 'wsrep_cluster%%'
+--------------------------+--------------------------------------+
|Variable_name             |Value                                 |
+--------------------------+--------------------------------------+
|wsrep_cluster_weight      |2                                     |
|wsrep_cluster_capabilities|                                      |
|wsrep_cluster_conf_id     |3                                     |
|wsrep_cluster_size        |2                                     |
|wsrep_cluster_state_uuid  |6708d7e8-a725-11ed-9f6f-ae60c774814e|
|wsrep_cluster_status      |Primary                               |
+--------------------------+--------------------------------------+
```

Получим данные по последним записанным числам:

```
-- replica 1
SELECT MAX(id) FROM numbers;
433
-- replica 2
SELECT MAX(id) FROM numbers;
433
```

Разницы нет. Однако, если мы повторим тестирование, уменьшив задержку между запросами с 2000мс до 5мс, и отправим большее их количество, репликация не справится:

```
-- master (после восстановления)
SELECT MAX(id) FROM numbers;
2194
-- replica 1
SELECT MAX(id) FROM numbers;
433
-- replica 2
SELECT MAX(id) FROM numbers;
433
```

4 Выводы

Видим, что репликация позволяет успешно реагировать на аварийные проблемы и помогать с распределением запросов на чтение, однако, со следующими ограничениями: - кластер должен быть корректно развёрнут - нагрузка должна соответствовать масштабу кластера (при единственном мастере и большом числе запросов наличие кластера не поможет никак) - в случае экстренной остановки мастера под высокой нагрузкой потери данных всё же возможны