## Сценарии

**Дата теста** 16.03.2024

1. Проверить производительность системы, когда все запросы идут на мастер и когда часть запросов перенесена на слейв.
2. Проверить отказоустойчивость системы при падении мастера при наличии 2 синхронных реплик.

## Последовательность действий.

1. Настраиваем асинхронную репликацию.
   1. В compose для мастера добавляем volumes db-data где будет находится кластер реплики (db-data:/var/lib/postgresql/database\_02).
   2. Добавляем в compose тег name для формирования имени сети для приложения.
   3. Запускаем мастер docker compose up
   4. Заходим на мастер docker exec -it social\_db\_postgres\_01 bash
   5. Открываем postgres .conf: vi /var/lib/postgresql/data/postgresql.conf
   6. В файле конфигурации меняем:
      1. ssl = off
      2. wal\_level = replica
      3. max\_wal\_senders = 4
   7. Открываем psql:
      1. su – postgres
      2. psql
   8. Создаем пользователя для репликации
      1. create role replicator with login replication password 'password';
   9. Добавляем запись в pg\_hba.conf с subnet, где subnet адрес сети, созданной из compose.
      1. Находим адрес сети: docker network inspect otus\_social\_default | grep Subnet
      2. Добавляем запись:

host replication replicator \_\_SUBNET\_\_ md5

* 1. Перезапускаем мастер: docker restart social\_db\_postgres\_01
  2. Заходим на мастер и создаем бекап для реплики
     1. docker exec -it social\_db\_postgres\_01 bash
     2. pg\_basebackup -h social\_db\_postgres\_01 -D /var/lib/postgresql/database\_02/ -U replicator -v -P --wal-method=stream
     3. Переходим в копию кластера и указываем, что это реплика: touch /var/lib/postgresql/database\_02/standby.signal
     4. Меняем postgresql.conf у реплики: primary\_conninfo = 'host=social\_db\_postgres\_01 port=5432 user=replicator password=password application\_name=social\_db\_postgres\_02'
  3. Запускаем новый экземпляр postgres. docker run -v "otus\_social\_db\_data:/var/lib/postgresql/data" -e POSTGRES\_PASSWORD=password -p "5444:5432" --network=otus\_social\_default --name=social\_db\_postgres\_02

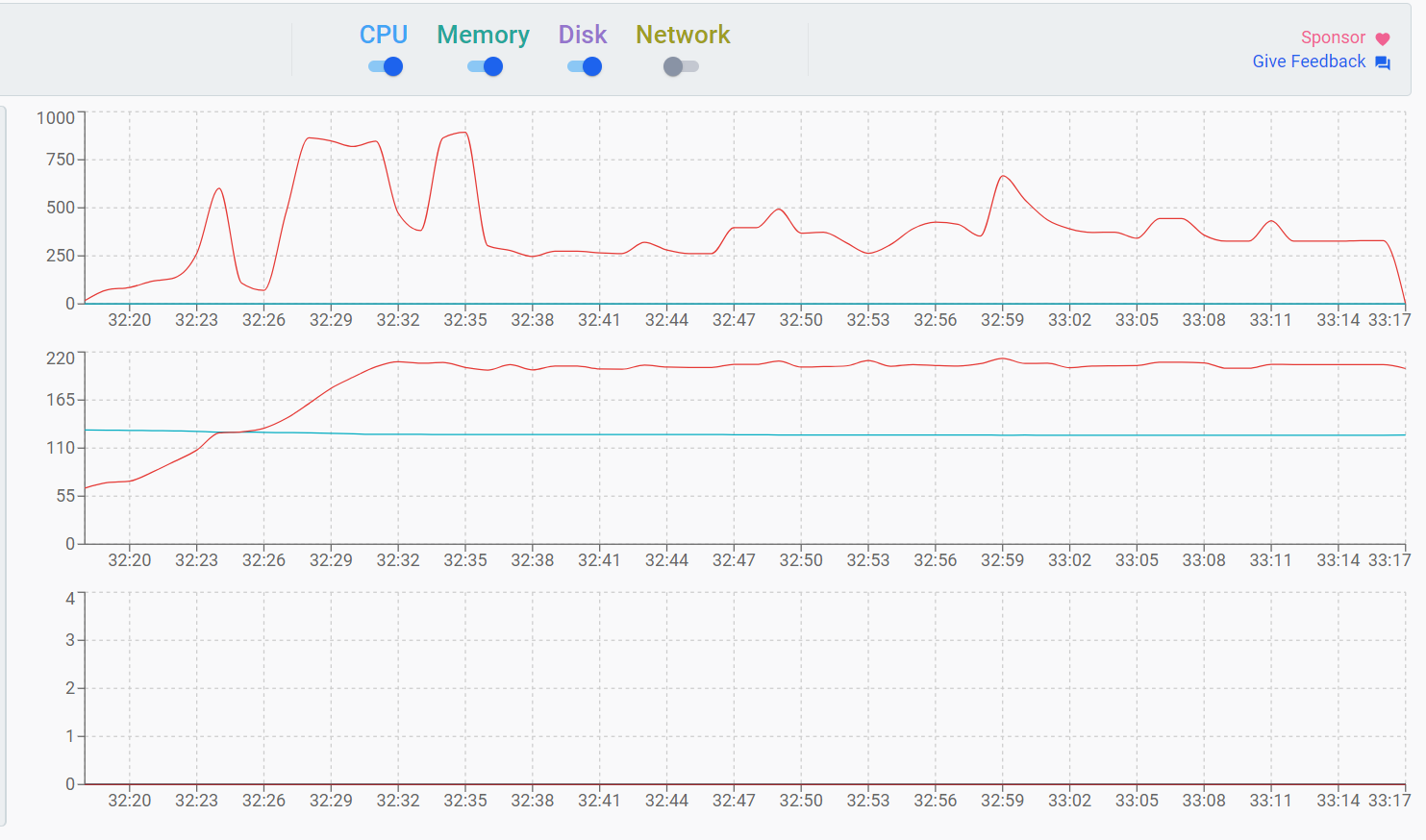
1. Замеряем нагрузку на мастере до перевода запросов на слейв.
   1. Запускаем нагрузочный тест (JMeter). Выполняем 1000 запросов на метод search в течение 1 минуты (load\_test.jmx). Данные представлены в разделе Результаты
2. Настроить два слейва и один мастер (Описание сокращенно, без точных команд).
   1. Добавил в мастер еще два volume, через docker commit.
   2. На всякий случай добавил архивацию wal
   3. Сделал две копии (по аналогии с асинхронной репликацией)
   4. Настроил копии как реплики (по аналогии с асинхронной репликацией), добавив файл standby.signal и прописав настройки для мастера.
   5. Сменил асинхронную репликацию на синхронную для мастера. В файле postgres.conf установил synchronous\_commit = on и synchronous\_standby\_names = FIRST 2 (social\_db\_postgres\_03, social\_db\_postgres\_04)
   6. Проверил, что новые реплики работаю в синхронном режиме.
   7. Запустил программу с флагом генерации тестовых данных.
   8. Завершил работы мастера командой container kill.
   9. Запромоутил social\_db\_postgres\_04. Pg\_promote()
   10. Для реплики social\_db\_postgres\_03 изменил настройку чтобы мастер стал social\_db\_postgres\_04.
   11. Проверил количество записей в базе social\_db\_postgres\_01: 1005119
   12. Проверил количество записей в базах social\_db\_postgres\_03 и social\_db\_postgres\_04: 1004863

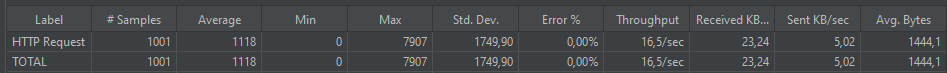
## Основные выводы

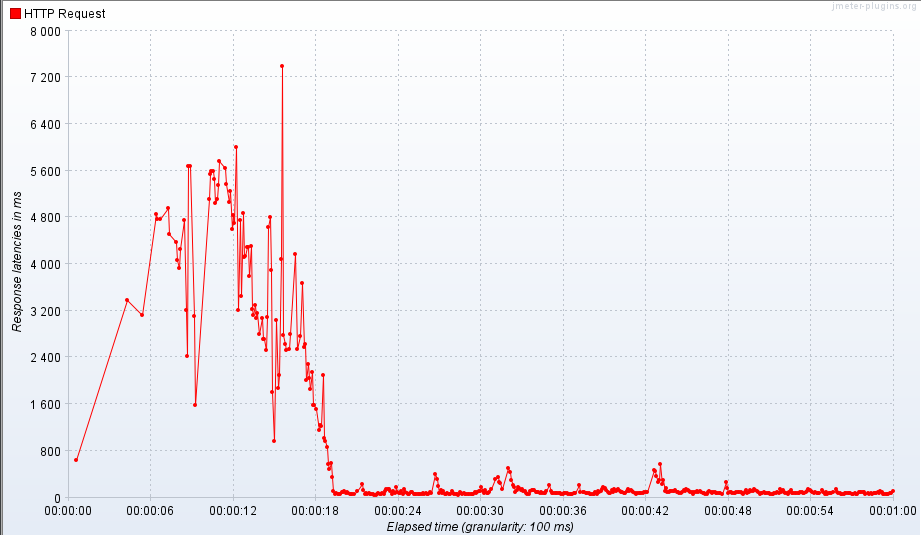
* При переносе запросов на слейв нагрузка на мастер снизилась, latency уменьшилось.
* Синхронная репликация не предотвращает возможную потерю данных. При экстренном завершении мастера были потеряны более 200 транзакций.

## Результаты

* Данные когда все запросы идут на мастер.







* *Данные, когда часть запросов идет на слейв.*

