Отчет по выполнению домашнего задания по теме «Репликация. Практическое применение»

Четвериков Владимир

Настройка асинхронной репликации

Создаем сеть docker для postgres: **docker network create pgnet**
Определяем подсеть созданной сети: **docker network inspect pgnet.** В моем случае это "Subnet": "**172.19.0.0/16**"

Запускаем мастер **docker run -dit -v "pgmaster:/var/lib/postgresql/data" -e POSTGRES_PASSWORD=pass -p "5432:5432" --restart=unless-stopped --network=pgnet --name=pgmaster postgres**

Для docker desktop windows файлы, используемые ниже, располагаются в папке **\\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes\pgmaster_data**
Раскомментируем и меняем параметры postgresql.conf в volume :
ssl = off
wal_level = replica
max_wal_senders = 4

Подключаемся к мастеру в docker: **docker exec -it pgmaster su - postgres -c psql**
Создаем роль репликатора: **create role replicator with login replication password 'pass';** Получаем ответ: **CREATE ROLE**
Выходим командой **exit**

В volume мастера в файл **pg_hba.conf** добавляем запись:
host replication replicator 172.19.0.0/16 md5

Рестартуем мастер командой **docker restart pgmaster**

Сделаем бэкап для реплик, выполнив следующие команды:
docker exec -it pgmaster bash
mkdir /pgslave
pg_basebackup -h pgmaster -D /pgslave -U replicator -v -P --wal-method=stream (запросит пароль - pass)
exit

Настройка асинхронной репликации

Копируем папку бэкапа к себе на хост (копирование производится в папку **C:/Users/имя_пользователя/volumes/pgslave**):
docker cp pgmaster:/pgslave volumes/pgslave/

Создаем директорию **\\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes\pgslave_one_data** для первого слейва и переносим туда содержимое скопированной на предыдущем шаге директории. Дополнительно создаем файл **standby.signal**
В файле **postgresql.conf** добавляем:
primary_conninfo = 'host=pgmaster port=5432 user=replicator password=pass application_name=pgslave_one'

Стартуем первую реплику **pgslave_one** командой
docker run -dit -v "pgslave_one:/var/lib/postgresql/data" -e POSTGRES_PASSWORD=pass -p "15432:5432" --network=pgnet --restart=unless-stopped --name=pgslave_one postgres

Создаем директорию **\\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes\pgslave_two_data** для второго слейва и переносим туда содержимое скопированной из бэкапа директории. Дополнительно создаем файл **standby.signal**
В файле **postgresql.conf** добавляем:
primary_conninfo = 'host=pgmaster port=5432 user=replicator password=pass application_name=pgslave_two'

Стартуем вторую реплику **pgslave_two** командой
docker run -dit -v "pgslave_two:/var/lib/postgresql/data" -e POSTGRES_PASSWORD=pass -p "25432:5432" --network=pgnet --restart=unless-stopped --name=pgslave_two postgres
Проверяем работу реплик. Заходим на мастер **docker exec -it pgmaster su - postgres -c psql**
Выполняем команду select **application_name, sync_state from pg_stat_replication;**
Видим две записи:

```
application_name | sync_state
------------------------+------------
 pgslave_one       | async
 pgslave_two       | async
(2 rows)
```

Нагрузочное тестирование методов чтения

Проведем нагрузочное тестирование с использованием наработок с ДЗ по нагрузочному тестированию.
Подключимся нашим приложением к master postgres и создадим около 165к пользователей.
При загрузке в master имеем такую картину по ресурсам

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
239d919f4817	pgslave_two	16.09%	101.6MiB / 6.142GiB	1.62%	80MB / 28.4MB	0B / 0B	7
2cd87c89138d	pgslave_one	15.37%	107.1MiB / 6.142GiB	1.70%	80MB / 28.8MB	0B / 0B	7
28edce988976	pgmaster	30.67%	169.3MiB / 6.142GiB	2.69%	91.1MB / 169MB	0B / 0B	42

В процессе создания пользователей происходит активная запись в master и асинхронная репликация в две реплики. Нагрузка CPU мастера, а также Net Output имеет высокие значения, так как происходит передача данных от мастера к репликам. В свою очередь, на репликах большее значение занимает Net Input (половина от Net Output мастера) так как происходит получение данных с мастера в процессе репликации.
Количество записей в таблице на мастере и репликах после завершения создания пользователей – одинаковые.

Нагрузим приложение по методам **чтения /user/get/{id} и /user/search** (приложение подключено к master).

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
239d919f4817	pgslave_two	0.00%	194.7MiB / 6.142GiB	3.10%	165MB / 56.6MB	0B / 0B	7
2cd87c89138d	pgslave_one	0.00%	196.1MiB / 6.142GiB	3.12%	165MB / 56.9MB	0B / 0B	7
28edce988976	pgmaster	207.38%	270.7MiB / 6.142GiB	4.30%	182MB / 355MB	0B / 0B	42

По использованию ресурсов видно, что чтение производится с мастера, реплики не задействованы.
Перенесем чтение в приложении на реплику **pgslave_one** и проведем повторное нагрузочное тестирование.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
239d919f4817	pgslave_two	0.00%	194.7MiB / 6.142GiB	3.10%	165MB / 56.6MB	0B / 0B	7
2cd87c89138d	pgslave_one	213.56%	249.6MiB / 6.142GiB	3.97%	165MB / 57.9MB	0B / 0B	41
28edce988976	pgmaster	0.01%	212.4MiB / 6.142GiB	3.38%	183MB / 362MB	0B / 0B	10

По использованию ресурсов видно, что чтение производится с реплики, **pgslave_one**. Мастер при чтении не задействован.
Очищаем таблицу с пользователями.

Синхронная репликация

Переводим мастер в режим синхронной репликации.
Для этого меняем значения в **postgres.conf** мастера:
synchronous_commit = on
synchronous_standby_names = 'FIRST 1 (pgslave_one, pgslave_two)'

Заходим на мастер и перечитываем конфиг
docker exec -it pgmaster su - postgres -c psql
select pg_reload_conf();
Проверяем, что режим репликации изменился **select application_name, sync_state from pg_stat_replication;**
Получаем ответ:

```
postgres=# select application_name, sync_state from pg_stat_replication;
 application_name | sync_state
------------------+------------
 pgslave_one      | sync
 pgslave_two      | potential
(2 rows)
```

Как видно из вывода, реплика **pgslave_one** стала основной для репликации, **pgslave_two** – резервной (на случай выхода из строя pgslave_one).
Создадим нагрузку на запись в master. В процессе записи убиваем мастер командой **docker kill pgmaster** после чего останавливаем запись.
Выбираем слейв **pgslave_one** и промоутим его до мастера.
docker exec -it pgslave_one su - postgres -c psql
select pg_promote();
Проверяем отсутствие репликации с нового мастера **pgslave_one**
select application_name, sync_state from pg_stat_replication;

```
postgres=# select application_name, sync_state from pg_stat_replication;
 application_name | sync_state
------------------+------------
(0 rows)
```

Синхронная репликация

Восстанавливаем репликацию на новом мастере **pgslave_one**
В файле **postgresql.conf** для **pgslave_one** устанавливаем:
synchronous_commit = on
synchronous_standby_names = 'FIRST 1 (pgslave_two)'
И перечитываем конфиг
select pg_reload_conf();

Подключаем **pgslave_two** к новому мастеру **pgslave_one**. Для этого в **postgresql.conf** для **pgslave_two** меняем
primary_conninfo = 'host=pgslave_one port=5432 user=replicator password=pass application_name=pgslave_two'
Подключаемся к **pgslave_two**
docker exec -it pgslave_two su - postgres -c psql и перечитываем конфиг **select pg_reload_conf();**

Подключаемся обратно к **pgslave_one**
docker exec -it pgslave_one su - postgres -c psql и проверяем статус репликации
select application_name, sync_state from pg_stat_replication;

```
postgres=# select application_name, sync_state from pg_stat_replication;
 application_name | sync_state
------------------+------------
 pgslave_two      | sync
(1 row)
```

Репликация с нового мастера восстановлена.
Сравниваем количество записей в новом мастере, реплике и в логах приложения.
Для этого на новом мастере **pgslave_one** выполняем **select count(*) from users**; Получаем результат 440
На реплике **pgslave_two** выполняем **select count(*) from users;** Получаем результат 440
Смотрим данные в логах приложения – 440 записей выполнено успешно.

```
2023-12-30T23:34:11.118+03:00  WARN 5420 --- [nio-8080-exec-1] c.v.s.controller.UserController         : Failure!! Saved success: 440 values. Failure: 2. Total: 442
```

Итоги и выводы

Репликация позволяет перенести нагрузку чтения с мастера на реплики – подтверждено в рамках практической работы.

Асинхронная репликация практически не влияет на скорость записи в мастер. При таком виде репликации существует риск потери транзакций при падении мастера в процессе активной записи, так как он не ждет подтверждения от синхронных реплик, а между записью в мастер с подтверждением записи и записью в реплику существует временной лаг.

При синхронной репликации вероятность потери транзакций при падении мастера в процессе активной записи отсутствует (если в живых остается синхронная реплика). Такой вид репликации более надежен с точки зрения устойчивости к потере данных (что подтвердилось в рамках практической работы), но запись в этом случае замедляется из-за необходимости ожидания подтверждения записи от реплик.