Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Распределённые системы хранения данных

Лабораторная работа 2

Вариант 30

Выполнил:
Кривоносов Егор Дмитриевич

Группа: P33111

Преподаватель:
Николаев Владимир Вячеславович

2022 г.

Санкт-Петербург

Текст задания

На выделенном узле создать и сконфигурировать новый кластер БД, саму БД, табличные пространства и новую роль в соответствии с заданием. Произвести наполнение базы.

Инициализация кластера БД

- Имя узла — pg106.
- Имя пользователя — postgres1.
- Директория кластера БД — $HOME/u08/dnk13.
- Кодировка, локаль — ANSI1251, русская
- Перечисленные параметры задать через переменные окружения.

Конфигурация и запуск сервера БД (ПОДРОБНОЕ ОПИСАНИЕ)

- Способ подключения к БД — TCP/IP socket, номер порта 9030.
- Остальные способы подключений запретить.
- Способ аутентификации клиентов — по паролю MD5.
- Настроить следующие параметры сервера БД: max_connections, shared_buffers, temp_buffers, work_mem, checkpoint_timeout, effective_cache_size, fsync, commit_delay. Параметры должны быть подобраны в соответствии со сценарием OLTP: 1000 транзакций/сек. с записью размером по 4 КБ, акцент на высокую доступность данных;
- Директория WAL файлов — $HOME/u03/dfl14.
- Формат лог-файлов — log.
- Уровень сообщений лога — ERROR.
- Дополнительно логировать — контрольные точки.

Дополнительные табличные пространства и наполнение

- Создать новое табличное пространство для индексов: $HOME/u03/dfl14.
- На основе template1 создать новую базу — whitebunny.
- От имени новой роли (не администратора) произвести наполнение существующих баз тестовыми наборами данных. Предоставить права по необходимости. Табличные пространства должны использоваться назначению.
- Вывести список всех табличных пространств кластера и содержащиеся в них объекты.

Выполнение

1. Подключение:

1) ssh s284261@se.ifmo.ru -p 2222 (Пароль: ******)
2) ssh postgres1@pg106 (Пароль: ******)

2. Инициализация кластера

PGDATA=$HOME/u08/dnk13
PGLOCALE=ru_RU.ANSI1251
PGENCODE=WIN1251
PGUSERNAME=postgres1
PGHOST=pg106
export PGDATA PGLOCALE PGENCODE PGUSERNAME PGHOST

env | sort

```
PGDATA=/var/postgres/postgres1/u08/dnk13
PGENCODE=WIN1251
PGHOST=pg106
PGLOCALE=ru_RU.ANSI1251
PGUSERNAME=postgres1
```

mkdir -p $PGDATA

initdb --locale=$PGLOCALE --encoding=$PGENCODE --username=$PGUSERNAME

```
bash-3.2$ initdb --locale=$PGLOCALE --encoding=$PGENCODE --username=$PGUSERNAME
The files belonging to this database system will be owned by user "postgres1".
This user must also own the server process.

The database cluster will be initialized with locale "ru_RU.ANSI1251".
The default text search configuration will be set to "russian".

Data page checksums are disabled.

fixing permissions on existing directory /var/postgres/postgres1/u08/dnk13 ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Europe/Moscow
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D /var/postgres/postgres1/u08/dnk13 -l logfile start
```

pg_ctl -D /var/postgres/postgres1/u08/dnk13 -l logfile start

```
bash-3.2$ pg_ctl -D /var/postgres/postgres1/u08/dnk13 -l logfile start
waiting for server to start.... done
server started
bash-3.2$
```

3. pg_hba.conf

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# rshd lab2 task
host    all             all             all                     md5
# "local" is for Unix domain socket connections only
local   all             all                                     reject
# IPv4 local connections:
host    all             all             127.0.0.1/32            reject
# IPv6 local connections:
host    all             all             ::1/128                 reject
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     reject
host    replication     all             127.0.0.1/32            reject
host    replication     all             ::1/128                 reject
```

Перед тем как делать это, сначала лучше создать пользователя с паролем, иначе мы просто не сможем зайти. (method поменять на `trust` как и было)

4. postgresql.conf

- Способ подключения к БД — TCP/IP socket, номер порта 9030.

```
64  port = 9030                 # (change requires restart)
```

Устанавливаем порт сервера

- **Способ аутентификации клиентов — по паролю MD5.**

```
95  #authentication_timeout = 1min      # 1s-600s
96  password_encryption = md5   # scram-sha-256 or md5
97  #db_user_namespace = off
```

Устанавливается для правильного шифрования паролей.

- **Директория WAL файлов — $HOME/u03/dfl14.**

```
245 archive_mode = on

247 archive_command = 'cp %p $HOME/u03/dfl14/%f'
```

Команда локальной оболочки, выполняемая для архивирования завершенного сегмента файла WAL. Если `acrhive_mode = off`, тогда `archive_command` - игнорируется. Если archive_command является пустой строкой (по умолчанию) при включенном archive_mode,

архивирование WAL временно отключается, но сервер продолжает накапливать файлы сегмента WAL в ожидании, что вскоре будет предоставлена команда.

- **Формат лог-файлов — log.**

```
433  log_destination = 'stderr'
439  logging_collector = on
```

Параметр включает сборщик сообщений (*logging collector*). Это фоновый процесс, который собирает отправленные в stderr сообщения и перенаправляет их в журнальные файлы. Такой подход зачастую более полезен чем запись в syslog, поскольку некоторые сообщения в syslog могут не попасть. (Типичный пример с сообщениями об ошибках динамического связывания, другой пример — ошибки в скриптах типа `archive_command`.)

```
444  # These are only used if logging_collector is on:
445  log_directory = 'log'           # directory where l
```

При включённом `logging_collector`, определяет каталог, в котором создаются журнальные файлы. Можно задавать как абсолютный путь, так и относительный от каталога данных кластера.

- **Уровень сообщений лога — ERROR.**

```
477  log_min_messages = error
```

Управляет минимальным уровнем сообщений, записываемых в журнал сервера. Допустимые значения `DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL и PANIC`. Каждый из перечисленных уровней включает все идущие после него. Чем дальше в этом списке уровень сообщения, тем меньше сообщений будет записано в журнал сервера. По умолчанию используется `WARNING`.

- **Дополнительно логировать — контрольные точки.**

```
536  log_checkpoints = on
```

Включает протоколирование выполнения контрольных точек и точек перезапуска сервера. При этом записывается некоторая статистическая информация. Например, число записанных буферов и время, затраченное на их запись. По умолчанию выключен.

- **Настроить следующие параметры сервера БД: max_connections, shared_buffers, temp_buffers, work_mem, checkpoint_timeout, effective_cache_size, fsync. Параметры должны быть подобраны в соответствии со сценарием OLTP: 1000 транзакций/сек. с записью размером по 4 КБ, акцент на высокую доступность данных**

```
65  max_connections = 1000
```

Значение 1000 подключений я взял из размышления, что каждый сеанс может создать в теории по 1 транзакции. А это уже 1000 TPS. (на самом деле можно увеличить количество подключений ещё больше, просто предположив, что не все пользуются, но моя логика заключалась в том, что каждый сеанс = пользуется БД).

Если каждый сеанс создаст по 1 транзакции и чтобы ограничить количество, сколько может выполняться максимально транзакций за раз от всех 1000 сеансов ниже нужно выставить параметр `max_prepared_transactions` на 1000. Ведь если транзакций будет выполняться больше 1000, это будет уже отклонение от задания 1000 TPS (1 транзакция * 1000 сеансов).

```
134  max_prepared_transactions = 1000
```

- Дополнительный параметр, который нужно добавить.
Задаёт максимальное число транзакций, которые могут одновременно находиться в «подготовленном» состоянии. Если же подготовленные транзакции применяются, то `max_prepared_transactions`, вероятно, должен быть не меньше, чем max_connections, чтобы подготовить транзакцию можно было в каждом сеансе.

```
132
133  temp_buffers = 2MB
```

Задаёт максимальное число временных буферов для каждого сеанса. В настоящее время используется только для хранения временных таблиц в памяти.

Предположив что 4 КБ у нас используется на запись для каждой транзакции (по заданию), а так как один сеанс может поддерживать максимум 1000 транзакций. Потому что у нас стоит ограничение в `max_prepared_transactions` на 1000 транзакций (что на самом деле мало вероятно, что будет транзакций 1000 от одного сеанса), на каждый сеанс логично было бы выделять 4 * 1000 ≈ 4МБ пространства буфера для хранения временных таблиц (немного больше чем по заданию для корректной работы) или 512 буферов (1 буфер = 8 КБ). Но так как вероятность того, что 1 сеанс произведет 1000 транзакций за секунду очень мала нам не нужно выделять предельные значения для данного параметра, ведь мы просто будем большую часть нашей памяти впустую.
Поэтому было принято решение выделить среднее значение для `temp_buffers` и `work_mem` в 2 MB.

```
138  work_mem = 2MB
```

Если нам нужно выполнить сложную сортировку, увеличиваем значение work_mem для получения хороших результатов. Сортировка в памяти происходит намного быстрее, чем сортировка данных на диске. Установка очень высокого значения может стать причиной узкого места в памяти для вашей среды, поскольку этот параметр относится к операции сортировки пользователя. Так как у нас в теории может быть 1000 пользователей установка этого параметра глобально может привести к очень высокому использованию памяти. Рекомендуется настраивать для каждого сеанса отдельно.

Аналогично было выделено для work_mem памяти в 2МБ.

```
126
127  shared_buffers = 2GB                    # min 128kB
```

Задаёт объём памяти, который будет использовать сервер баз данных для буферов в разделяемой памяти. Разумным начальным значением `shared_buffers` будет 25% от объёма памяти. Существуют варианты нагрузки, при которых эффективны будут и ещё большие значения `shared_buffers`, но так как использует и кеш операционной системы, выделять для `shared_buffers` более 40% ОЗУ вряд ли будет полезно.

Теперь имея представление о затратах моей системы, можно предположить какой объем памяти понадобится системе. 2 GB будет задействовано максимум от `work_mem` и `temp_buffers` на 1000 транзакций (то есть 4 GB в сумме). Так как 25% мы будем выделять под `sharred_buffers`, а оставшуюся память под `effective_cacha_size`. Благоприятным значением системы будет объем памяти 8GB.

25% * 8GB = 2GB - выделяем под shared_buffers.

При увеличении `shared_buffers` обычно требуется соответственно увеличить `max_wal_size`, чтобы растянуть процесс записи большого объёма новых или изменённых данных на более продолжительное время. Но при увеличении `max_wal_size` время восстановление системы будет достаточно высокое.

```
240  max_wal_size = 4GB
241  min_wal_size = 80MB
```

```
236  checkpoint_timeout = 5min
```

Увеличение этого параметра может привести к увеличению времени, которое потребуется для восстановления после сбоя. Уменьшение приводит к учащению

контрольных точек, и в то же время повышению нагрузки, поэтому я данный параметр решил оставить по умолчанию.
Дефолтное значение в 5 минут идеально сбалансировано на мой взгляд для подобного узла, поскольку при увеличении его значения время восстановления может измениться.

```
393  effective_cache_size = 2GB
```

Определяет представление планировщика об эффективном размере дискового кеша, доступном для одного запроса. Точно должен быть >= `shared_buffers`. Как говорилось выше при расчетах, было принято решение выделить под кэш 2GB.

```
207  fsync = on
```
(по дефолту) - не нужно рисковать данными при этом гнаться за производительностью.
Выключение параметра приводит к росту производительности, но появляется значительный риск потери всех данных при внезапном выключении питания. Риск не оправдан при достаточно большим взаимодействием с БД.

5. Работа с БД

Сначала мы должны создать БД, к которой будем подключаться с помощью команды: psql -p 9030 -d whitebunny

- **Создать новое табличное пространство для индексов: $HOME/u03/dfl14.**

mkdir -p u03/dfl14
CREATE TABLESPACE indexspace LOCATION '/var/postgres/postgres1/u03/dfl14';

```
whitebunny=> \db
                List of tablespaces
    Name    |   Owner   |             Location
------------+-----------+----------------------------------
 indexspace | postgres1 | /var/postgres/postgres1/u03/dfl14
```

- **На основе template1 создать новую базу — whitebunny.**

createdb -p 9030 -T template1 whitebunny

CREATE DATABASE whitebunny WITH TEMPLATE = template1;

- **От имени новой роли (не администратора) произвести наполнение существующих баз тестовыми наборами данных. Предоставить права по необходимости. Табличные пространства должны использоваться назначению.**

CREATE TABLE bunny (id bigserial primary key, name text, type text, age int);
CREATE INDEX ON bunny (name) TABLESPACE indexspace;

CREATE ROLE s284261 LOGIN PASSWORD ‘1337’;
GRANT INSERT ON bunny TO s284261;
GRANT USAGE, SELECT ON SEQUENCE bunny_id_seq TO s284261;

psql -h pg106 -p 9030 -d whitebunny -U s284261
psql -h pg106 -p 9030 -d whitebunny -U s284261 -f insert.sql

```
bash-3.2$ psql -p 9030 -d whitebunny -U s284261 -f insert.sql
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

Добавить мы в таблицу мы можем без каких либо проблем.

```
whitebunny=> select * from bunny;
ERROR:  permission denied for table bunny
```

А посмотреть мы не можем, так как мы не выдавали пользователю s284261 таких права.

- **Вывести список всех табличных пространств кластера и содержащиеся в них объекты.**

SELECT * FROM pg_tablespace;

```
whitebunny=> select * from pg_tablespace
whitebunny-> ;
  oid  |  spcname   | spcowner | spcacl | spcoptions
-------+------------+----------+--------+------------
  1663 | pg_default |       10 |        |
  1664 | pg_global  |       10 |        |
 16386 | indexspace |       10 |        |
(3 rows)
```

SELECT relname FROM pg_class WHERE reltablespace IN (SELECT oid FROM pg_tablespace);

```
whitebunny=> SELECT relname FROM pg_class WHERE reltablespace IN (SELECT  oid FROM pg_tablespace);
                relname
---------------------------------------
 bunny_name_idx
 pg_toast_1262
 pg_toast_1262_index
 pg_toast_2964
 pg_toast_2964_index
 pg_toast_1213
 pg_toast_1213_index
 pg_toast_1260
 pg_toast_1260_index
 pg_toast_2396
 pg_toast_2396_index
 pg_toast_6000
 pg_toast_6000_index
 pg_toast_3592
 pg_toast_3592_index
 pg_toast_6100
 pg_toast_6100_index
 pg_database_datname_index
 pg_database_oid_index
 pg_db_role_setting_databaseid_rol_index
 pg_tablespace_oid_index
 pg_tablespace_spcname_index
 pg_authid_rolname_index
 pg_authid_oid_index
 pg_auth_members_role_member_index
 pg_auth_members_member_role_index
 pg_shdepend_depender_index
 pg_shdepend_reference_index
 pg_shdescription_o_c_index
 pg_replication_origin_roiident_index
 pg_replication_origin_roname_index
 pg_shseclabel_object_index
 pg_subscription_oid_index
 pg_subscription_subname_index
 pg_authid
 pg_subscription
 pg_database
 pg_db_role_setting
 pg_tablespace
 pg_auth_members
 pg_shdepend
 pg_shdescription
 pg_replication_origin
 pg_shseclabel
(44 rows)
```

Вывод

В ходе выполнения лабораторной работы была проделана следующая работа: На выделенном узле создан и сконфигурирован новый кластер БД, сама БД, табличные пространства и новая роль в соответствии с заданием. Произведено наполнение базы.