Авторы: Алексей Береснев, Илья Баштанов, Павел Толмачев

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Возможности и ограничения

Архитектура и принцип работы

Распределенные транзакции

Поколения узлов

Проверка состояния узлов

Отказ узла

Режим 2+1

Возможности

Синхронный симметричный кластер

- на всех узлах одинаковые данные, все узлы ведущие

Распределенные транзакции

- все узлы работают как единая СУБД с ACID-гарантиями

Отказоустойчивость и высокая доступность

Масштабируемость чтения

Расширение multimaster позволяет преобразовать сервер в синхронный симметричный кластер без разделения ресурсов — данные будут одинаковы на всех узлах кластера, клиенты могут читать и писать данные на любой из узлов. В кластере поддерживаются распределенные транзакции, дающие обычные гарантии ACID, поэтому приложения могут работать с кластером как с единой СУБД.

Multimaster обеспечивает отказоустойчивость за счет автоматического восстановления данных на вернувшихся в строй узлах; кластер имеет высокую доступность — у него нулевое время простоя. Менять конфигурацию (в том числе и обновлять в рамках одной и той же основной версии) узлов можно без остановки обслуживания. Перенаправляя на разные узлы читающие транзакции, можно повысить пропускную способность таких операций.

https://postgrespro.ru/docs/enterprise/13/multimaster

Ограничения

Postgres PROFESSIONAL

Не поддерживаются

- ОС Windows и решения 1С
- репликация нескольких баз данных
- большие объекты (Large Objects)
- операции неблокирующего построения и перестроения индексов и работы с табличными пространствами

Особенности работы с данными

- транзакции могут занимать длительное время
- на уровне Read Committed могут происходить сбои сериализации
- меняется поведение последовательностей
- OID могут отличаться на разных узлах

Некоторые другие особенности и ограничения

У расширения есть ряд ограничений.

Не поддерживается ОС Windows и продукты 1С.

Можно реплицировать только одну базу данных (но можно сделать несколько кластеров для разных баз данных).

Не поддерживаются большие объекты (Large Objects).

Не поддерживаются операции CREATE INDEX и REINDEX с предложением CONCURRENTLY, CREATE и DROP TABLESPACE.

Так как каждая транзакция подтверждается несколькими узлами, это занимает некоторое время. Чем больше узлов в кластере, тем дольше будет происходить фиксация изменений.

На уровне изоляции Read Committed транзакции могут завершаться ошибкой сериализации.

Последовательности по умолчанию выдают немонотонные значения, чтобы избежать конфликтов уникальных идентификаторов между узлами.

Идентификаторы OID могут быть различными на разных узлах.

Существуют и некоторые другие ограничения использования мультимастера.

https://postgrespro.ru/docs/enterprise/13/multimaster#MULTIMASTER-LIMITATIONS

Подготовка сервера

Остановим основной сервер.

```
student$ sudo systemctl stop postgrespro-ent-13.service
```

В этой демонстрации часть команд выполняется от имени пользователя ОС postgres. Обратите внимание на приглашения перед командами.

Для демонстрации инициализированы и настроены три экземпляра Postgres Pro Enterprise. Каталоги данных:

- ent-13-MM-1, порт 5001
- ent-13-MM-2, порт 5002
- ent-13-MM-3, порт 5003

Дополнительные параметры первого экземпляра:

```
postgres$ cat /var/lib/pgpro/ent-13-MM-1/conf.d/multimaster.conf
```

```
cluster_name = 'node1'
port = 5001
shared_preload_libraries = 'multimaster'
wal_level = logical
max_connections = 100 # значение по умолчанию
max_prepared_transactions = 300
max_wal_senders = 10 # значение по умолчанию
max_replication_slots = 10 # значение по умолчанию
wal_sender_timeout = 0
max_worker_processes = 320
```

Остальные экземпляры настроены аналогично.

Инициализация кластера multimaster

Инициализируем кластер multimaster из трех узлов:

```
student$ psql -p 5001 -U postgres
```

```
student$ psql -p 5002 -U postgres
```

```
student$ psql -p 5003 -U postgres
```

Подготовим роль и базу данных для работы мультимастера на каждом из узлов.

Для первого узла:

```
=> CREATE USER multimaster_user WITH SUPERUSER;
```

```
CREATE ROLE
```

```
=> CREATE DATABASE multimaster_db OWNER multimaster_user;
```

```
CREATE DATABASE
```

```
=> \c multimaster_db multimaster_user
```

```
You are now connected to database "multimaster_db" as user "multimaster_user".
```

Для второго узла:

```
=> CREATE USER multimaster_user WITH SUPERUSER;
CREATE ROLE
=> CREATE DATABASE multimaster_db OWNER multimaster_user;
CREATE DATABASE
=> \c multimaster_db multimaster_user
You are now connected to database "multimaster_db" as user "multimaster_user".
```

И для третьего узла:

```
=> CREATE USER multimaster_user WITH SUPERUSER;
CREATE ROLE
=> CREATE DATABASE multimaster_db OWNER multimaster_user;
CREATE DATABASE
```

```
    => \c multimaster_db multimaster_user
    You are now connected to database "multimaster_db" as user "multimaster_user".
```

Теперь объединим узлы в кластер. Для этого на любом из узлов добавим расширение multimaster в БД multimaster_db:

```
=> CREATE EXTENSION multimaster;

CREATE EXTENSION
```

Теперь запустим функцию инициализации кластера. Первый параметр — это строка подключения к текущему узлу, второй и третий — подключение к остальным узлам:

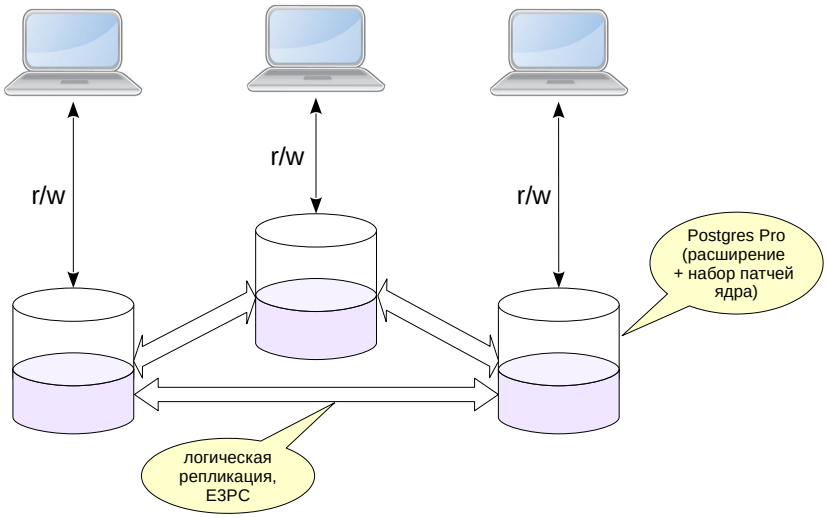
```
=> SELECT mtm.init_cluster(
    'dbname=multimaster_db user=multimaster_user port=5001',
   '{"dbname=multimaster_db user=multimaster_user port=5002",
     "dbname=multimaster_db user=multimaster_user port=5003"}');

 init_cluster
--------------

(1 row)
```

Функция возвращает либо пустой ответ в случае успешной инициализации кластера мультимастера, либо ошибку инициализации.

Архитектура



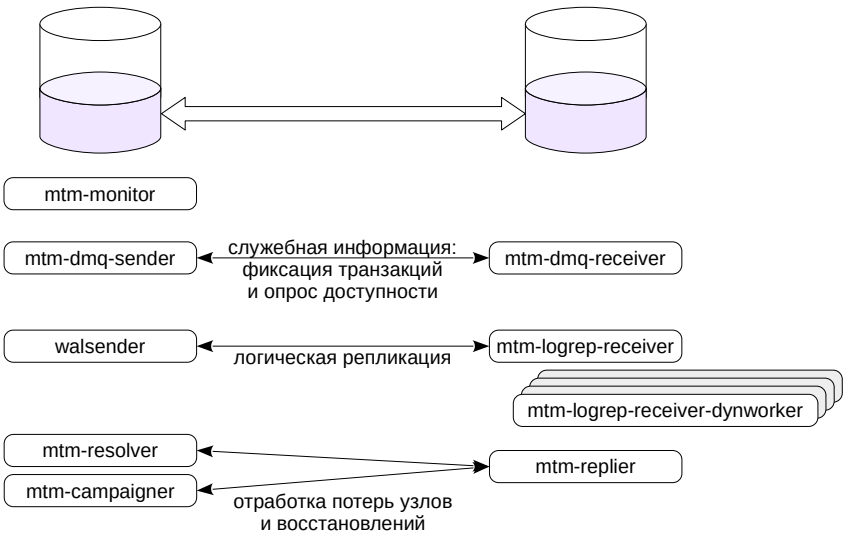
Расширение multimaster состоит из патчей ядра СУБД (входят в Postgres Pro) и непосредственно кода расширения.

Данные передаются между узлами с помощью доработанного механизма логической репликации.

Для обеспечения согласованности данных подтверждение распределенных транзакций выполняется с помощью улучшенного трехфазного протокола фиксации (E3PC). Для этого в кластере изначально должно быть не менее трех узлов.

Высокая степень доступности достигается тем, что узлы можно выводить из кластера и добавлять к кластеру «на лету». При этом клиенты могут быть подключены к любому узлу — все узлы принимают и читающую, и пишущую нагрузку.

Процессы



За разные задачи в мультимастере отвечают разные процессы:

- Процесс mtm-monitor запускается на каждом узле и управляет остальными процессами кластера.
- Процесс mtm-logrep-receiver получает поток логической репликации от walsender. Изменения применяются пулом процессов mtm-logrep-receiver-dynworker.
- Процесс mtm-dmq-sender посылает mtm-dmq-receiver служебную информацию, необходимую для реализации распределенных транзакций, а также выполняет периодический опрос доступности узлов (DMQ — distributed message queue, распределенная очередь сообщений).
- Процессы mtm-resolver и mtm-campaigner служат для согласованного исключения узлов из кластера и возвращения узлов в кластер; им отвечает процесс mtm-replier.

На рисунке показаны только процессы-источники одного узла и соответствующие им процессы-приемники другого узла. На самом деле и те процессы, и другие работают на каждом узле. Более того, их количество зависит от числа узлов в кластере: например, на каждом узле будет запущен процесс walsender для каждого из остальных узлов кластера.

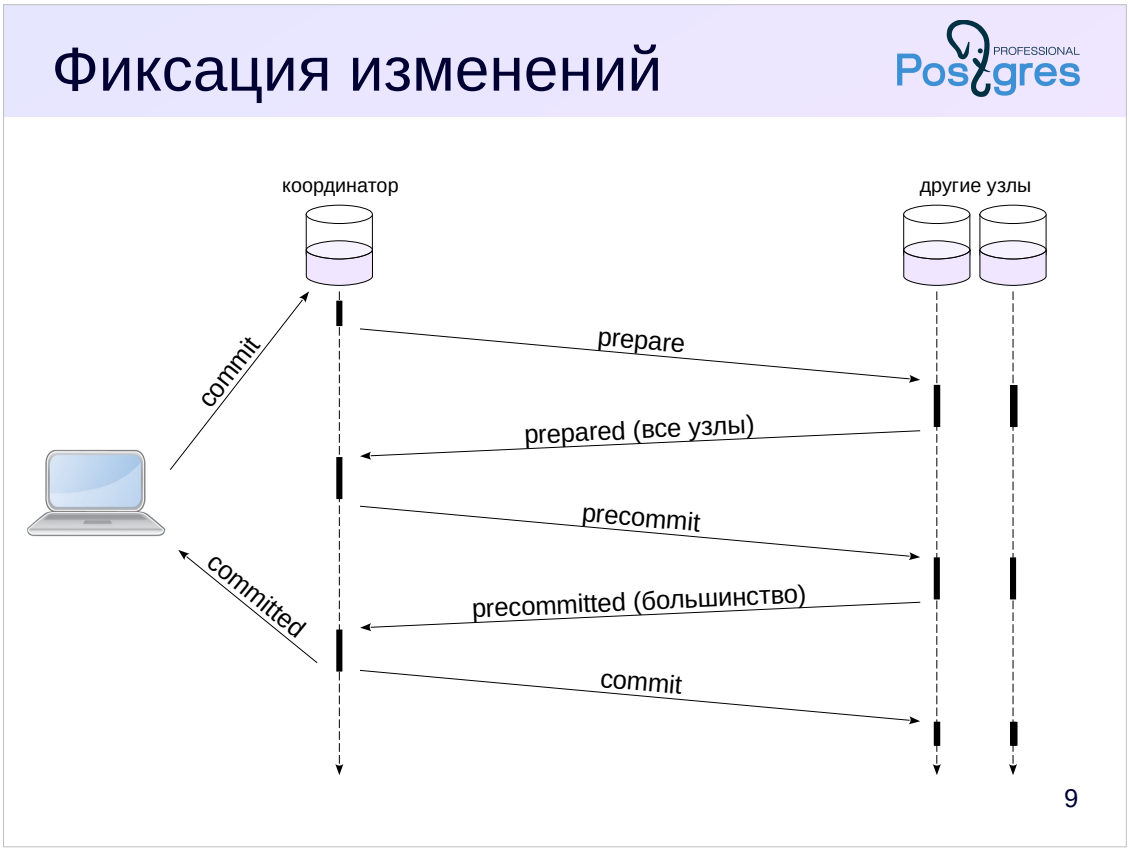
Процессы

Посмотрим на процессы, которые запускаются на каждом узле кластера:

```
student$ ps -o pid,command --ppid 289166
```

```
    PID COMMAND
 289168 postgres: node1: checkpointer
 289169 postgres: node1: background writer
 289170 postgres: node1: walwriter
 289171 postgres: node1: autovacuum launcher
 289172 postgres: node1: stats collector
 289174 postgres: node1: mtm-dmq-sender
 289175 postgres: node1: logical replication launcher
 289176 postgres: node1: cfs-worker-0
 290084 postgres: node1: multimaster_user multimaster_db [local] idle
 290315 postgres: node1: mtm-monitor
 290405 postgres: node1: mtm-logrep-receiver-1-2
 290416 postgres: node1: mtm-logrep-receiver-1-3
 290422 postgres: node1: mtm-replier
 290423 postgres: node1: mtm-campaigner
 290424 postgres: node1: mtm-resolver
 290425 postgres: node1: multimaster_user multimaster_db [local] mtm-dmq-receiver node2
 290427 postgres: node1: walsender multimaster_user [local] idle
 290428 postgres: node1: multimaster_user multimaster_db [local] mtm-dmq-receiver node3
 290453 postgres: node1: walsender multimaster_user [local] idle
```

Количество процессов довольно велико, следует обеспечить их достаточными ресурсами.



Протокол трехфазной фиксации расширяет стандартный протокол двухфазной фиксации, реализованный в PostgreSQL, и работает следующим образом:

1. **Prepare**. Когда на узле (который считается координатором транзакции) выполняется оператор COMMIT, транзакция подготавливается (PREPARE TRANSACTION) и передается на остальные доступные узлы. Эти узлы отвечают подтверждением или отказом.

Если какой-либо узел ответил отказом, транзакция обрывается.

2. **Precommit**. Узел-координатор рассылает остальным узлам команду Precommit, которая выражает намерение зафиксировать транзакцию.

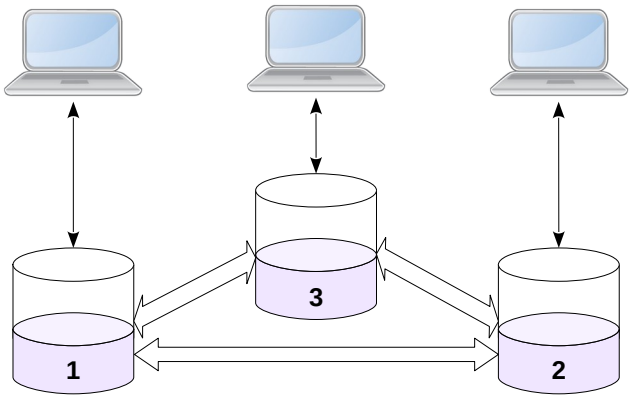
Для продолжения достаточно, чтобы подтверждением ответило большинство узлов (кворум). Если достаточного количества подтверждений не набирается, транзакция обрывается (ROLLBACK PREPARED).

3. **Commit**. Транзакция фиксируется на всех узлах (COMMIT PREPARED).

В отличии от двухфазной фиксации, в трехфазную добавлен этап Precommit. Если после него произойдет сбой одного из узлов и третий этап Commit не будет выполнен на сбойном узле, на остальных узлах заключительный этап завершится по тайм-ауту и транзакция успешно зафиксируется.

https://www.sciencedirect.com/science/article/pii/S0022000098915665?via%3Dihub

Поколения узлов



Поколение 1: {1,2,3}

Большинство узлов (но и не только оно) определяется с помощью *поколений.* Поколение — это подмножество узлов, считающихся в данный момент рабочими.

Поколение определяется номером (монотонно возрастающее значение) и перечнем представителей этого поколения. Узел всегда входит в какое-либо поколение. Если узел видит, что существует поколение с бóльшим номером, он старается перейти в него.

Новое поколение формируется большинством доступных узлов кластера на момент его запуска, после сбоя узлов и при добавлении в кластер нового узла.

Транзакцию можно зафиксировать только после того, как она будет подготовлена на всех представителях поколения.

На слайде проиллюстрирована ситуация кластера из трех узлов (поколение 1).

Проверка состояния кластера

Проверим статус узла кластера:

```
=> SELECT * FROM mtm.status();
```

```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+---------------
          1 | online | {1,2,3}   |       1 | {1,2,3}     | {1,2,3}            | {1,2,3}
(1 row)
```

На любом из узлов кластера можно получить список всех узлов:

```
   => SELECT * FROM mtm.nodes();
```

```
    id |                       conninfo                       | is_self | enabled | connected | sender_pid | receiver_pid | n_workers | receiver_mode
   ----+------------------------------------------------------+---------+---------+-----------+------------+--------------+-----------+---------------
     1 | dbname=multimaster_db user=multimaster_user port=5001 | f       | t       | t         |     290452 |       290403 |         0 | normal
     2 | dbname=multimaster_db user=multimaster_user port=5002 | f       | t       | t         |     290454 |       290410 |         0 | normal
     3 | dbname=multimaster_db user=multimaster_user port=5003 | t       | t       | t         |            |              |           |
   (3 rows)
```

Проверим работу кластера — создадим таблицу на первом узле и добавим туда несколько строк:

```
=> CREATE TABLE multimaster_tbl (id integer);
```

```
CREATE TABLE
```

```
=> INSERT INTO multimaster_tbl VALUES (1),(2),(3);
```

```
INSERT 0 3
```

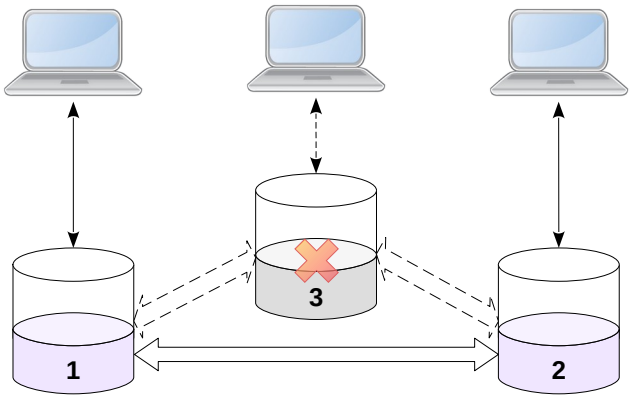
Проверим доступность таблицы multimaster_tbl на втором узле:

```
  => SELECT * FROM multimaster_tbl;
```

```
   id
  ----
    1
    2
    3
  (3 rows)
```

Таблица и строки доступны на втором узле — кластер работает.

Отказ узла



Поколение 1: {1,2,3}
Поколение 2: {1,2}

При отказе одного из узлов (выключение узла, недоступность узла по сети) оставшиеся узлы формируют новое поколение и в дальнейшем транзакции подтверждаются большинством из оставшихся узлов.

Отказ узла

Для демонстрации отказа узла выключим экземпляр Postgres Pro на третьем сервере:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-3 stop
```

```
waiting for server to shut down.... done
server stopped
```

Проверим доступные узлы кластера:

```
=> SELECT * FROM mtm.status();
```

```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          1 | online | {1,2}     |       2 | {1,2}       | {1,2}              | {1,2,3}
(1 row)
```

Номер поколения увеличился.

Повторим запись в таблицу multimaster_tbl:

```
=> INSERT INTO multimaster_tbl VALUES (11),(22),(33);
```

```
INSERT 0 3
```

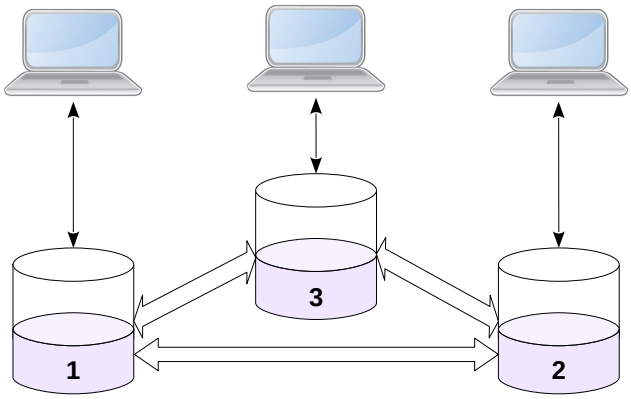
Проверим корректную работу кластера — прочитаем все строки из этой таблицы на первом узле:

```
=> SELECT * FROM multimaster_tbl;
```

```
 id
----
  1
  2
  3
 11
 22
 33
(6 rows)
```

Добавленные строки видны: кластер продолжает работать даже при отсутствии одного из узлов.

Восстановление узла



Поколение 1: {1,2,3}
Поколение 2: {1,2}
Поколение 3: {1,2,3}

После восстановления третьего узла он определяет номер текущего поколения кластера, получает список узлов, находящихся в этом поколении и выполняет восстановление с одного из этих узлов.

После окончания восстановления третий узел предлагает включить его в кластер, формируя новое поколение 3.

Восстановление узла

Текущее состояние узлов кластера:

```
=> SELECT * FROM mtm.status();
```

```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          1 | online | {1,2}     |       2 | {1,2}       | {1,2}              | {1,2,3}
(1 row)
```

Вернем третий узел в состав кластера. Для этого запустим экземпляр Postgres Pro на третьем узле:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-3/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-3 start
```

```
waiting for server to start.... done
server started
```

```
student$ psql -p 5003 -U multimaster_user -d multimaster_db
```

```
=> SELECT * FROM mtm.status();
```

```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          1 | online | {1,2,3}   |       2 | {1,2}       | {1,2}              | {1,2,3}
(1 row)
```

После запуска узел подключится к кластеру, получит все необходимые изменения...

...и вернется в строй. Номер поколения снова увеличится:

```
=> SELECT * FROM mtm.status();
```

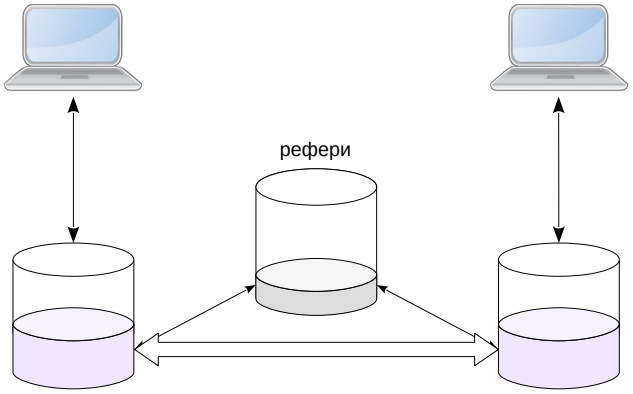
```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          3 | online | {1,2,3}   |       3 | {1,2,3}     | {1,2,3}            | {1,2,3}
(1 row)
```

Все добавленные ранее строки видны на третьем узле.

```
=> SELECT * FROM multimaster_tbl;
```

```
 id
----
  1
  2
  3
 11
 22
 33
(6 rows)
```

Режим 2+1



Для работы синхронного симметричного кластера требуется, чтобы все узлы кластера были примерно одинаковы по их аппаратным характеристикам.

Помимо этого, существует режим работы кластера мультимастера 2+1 (его еще называют режимом работы с рефери). В этом случае в кластер входят два полноценных узла, а третий узел работает только как голосующий — он принимает решения о подтверждении и отмене транзакций.

На узле-рефери достаточно установить Postgres Pro Enterprise и добавить расширение referee. Узел-рефери не хранит никакие данные кластера, поэтому он создает небольшую нагрузку и не требователен к ресурсам.

Режим 2+1

Для перехода в режим работы 2+1 удалим из кластера один из узлов (третий):

```
=> SELECT mtm.drop_node(3);
```

```
 drop_node
-----------

(1 row)
```

На удаленном узле добавим расширение referee:

```
=> CREATE EXTENSION referee;
```

```
CREATE EXTENSION
```

Остановим остальные два узла:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-1 stop
```

```
waiting for server to shut down.... done
server stopped
```

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-2 stop
```

```
waiting for server to shut down.... done
server stopped
```

На каждом из двух узлов в файл конфигурации необходимо добавить строку подключения к узлу-рефери:

```
student$ cat << EOF | sudo -u postgres tee -a /var/lib/pgpro/ent-13-MM-1/conf.d/multimaster.conf
multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user host=node3 port=5003'
EOF
```

```
multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user host=node3 port=5003'
```

```
student$ cat << EOF | sudo -u postgres tee -a /var/lib/pgpro/ent-13-MM-2/conf.d/multimaster.conf
multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user host=node3 port=5003'
EOF
```

```
multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user host=node3 port=5003'
```

Запускаем любой из основных узлов. Например второй:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-2/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-2 start
```

```
waiting for server to start.... done
server started
```

```
student$ psql -p 5002 -U multimaster_user -d multimaster_db
```

Запускаем другой узел:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-1/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-1 start
```

```
waiting for server to start.... done
server started
```

```
student$ psql -p 5001 -U multimaster_user -d multimaster_db
```

Проверим статус кластера:

```
=> SELECT * FROM mtm.status();
```

```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          2 | online | {1,2}     |       4 | {1,2}       | {1,2}              | {1,2}
(1 row)
```

Проверим также, что ранее созданная таблица на месте:

```
=> SELECT * FROM multimaster_tbl;
```

```
 id
----
  1
  2
  3
 11
 22
 33
(6 rows)
```

Снова добавим строки в таблицу multimaster_tbl:

```
  => INSERT INTO multimaster_tbl VALUES (111),(222),(333);
  INSERT 0 3
=> SELECT * FROM multimaster_tbl;
 id
-----
   1
   2
   3
  11
  22
  33
 111
 222
 333
(9 rows)
```

Кластер продолжает работать. Но он уже состоит из двух узлов и одного рефери.

Итоги

Postgres Pro Multimaster — это кластерное решение со всеми ведущими узлами, обеспечивающее автоматическое восстановление узлов после сбоев

Не требует установки дополнительного ПО

Сфера применения — системы высокой доступности

Практика

1. Настройте кластер для режима работы 2+1.
2. Имитируйте отказ одного из двух узлов и проверьте работоспособность кластера.
3. Верните отключенный узел в строй.

1. Подготовьте три дополнительных экземпляра Postgres Pro Enterprise: для каждого из них инициализируйте каталог данных, настройте аутентификацию и задайте параметры. Подробности можно посмотреть в решении практики.

1. Настройка кластера для режима работы 2+1

Остановим основной сервер.

```
student$ sudo systemctl stop postgrespro-ent-13.service
```

Инициализируем кластер из двух узлов и третий узел для рефери.

Сначала подготовим три дополнительных экземпляра сервера Postgres Pro.

Остановим экземпляр, если он работает:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-1 stop

pg_ctl: directory "/var/lib/pgpro/ent-13-MM-1" does not exist
```

Создаем каталог PGDATA:

```
student$ sudo rm -rf /var/lib/pgpro/ent-13-MM-1

student$ sudo mkdir /var/lib/pgpro/ent-13-MM-1

student$ sudo chown postgres: -R /var/lib/pgpro/ent-13-MM-1
```

Инициализируем каталог данных:

```
postgres$ /opt/pgpro/ent-13/bin/initdb -U postgres -k -D /var/lib/pgpro/ent-13-MM-1

The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locales
  COLLATE:  en_US.UTF-8
  CTYPE:    en_US.UTF-8
  MESSAGES: en_US.UTF-8
  MONETARY: ru_RU.UTF-8
  NUMERIC:  ru_RU.UTF-8
  TIME:     ru_RU.UTF-8
The default collation provider is "icu".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are enabled.

fixing permissions on existing directory /var/lib/pgpro/ent-13-MM-1 ... 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:

    /opt/pgpro/ent-13/bin/pg_ctl -D /var/lib/pgpro/ent-13-MM-1 -l logfile start
```

Вставим первой строкой настройки аутентификации для роли multimaster_user:

```
postgres$ sed -i '1 i\local    replication,multimaster_db     multimaster_user        trust' /var/lib/pgpro/ent-13-MM-1/pg_hba.conf
```

Конфигурационный файл:

```
postgres$ echo include_dir \'conf.d\' >> /var/lib/pgpro/ent-13-MM-1/postgresql.conf

postgres$ mkdir /var/lib/pgpro/ent-13-MM-1/conf.d

student$ cat << EOF | sudo -u postgres tee /var/lib/pgpro/ent-13-MM-1/conf.d/multimaster.conf
        cluster_name = 'node1'
        shared_preload_libraries = 'multimaster'
        wal_level = logical
        max_connections = 100 # значение по умолчанию
        max_prepared_transactions = 300
        max_wal_senders = 10 # значение по умолчанию
        max_replication_slots = 10 # значение по умолчанию
        wal_sender_timeout = 0
        max_worker_processes = 320
        port = 5001
EOF

        cluster_name = 'node1'
        shared_preload_libraries = 'multimaster'
        wal_level = logical
        max_connections = 100 # значение по умолчанию
        max_prepared_transactions = 300
        max_wal_senders = 10 # значение по умолчанию
        max_replication_slots = 10 # значение по умолчанию
```

```
        wal_sender_timeout = 0
        max_worker_processes = 320
        port = 5001
```

Запускаем экземпляр:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-1/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-1 start

waiting for server to start.... done
server started
```

Создаем роль и базу данных для мультимастера:

```
student$ psql -p 5001 -U postgres

=> CREATE USER multimaster_user WITH SUPERUSER;

CREATE ROLE

=> CREATE DATABASE multimaster_db OWNER multimaster_user;

CREATE DATABASE

=> \c multimaster_db multimaster_user

You are now connected to database "multimaster_db" as user "multimaster_user".
```

Аналогично нужно подготовить еще два экземпляра на портах 5002 и 5003.

На третьем узле добавим расширение referee:

```
    => CREATE EXTENSION referee;

    CREATE EXTENSION
```

На каждом из двух основных узлов добавим в файл конфигурации строку подключения к узлу-рефери:

```
student$ cat << EOF | sudo -u postgres tee -a /var/lib/pgpro/ent-13-MM-1/conf.d/multimaster.conf
multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user port=5003'
EOF

multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user port=5003'

student$ cat << EOF | sudo -u postgres tee -a /var/lib/pgpro/ent-13-MM-2/conf.d/multimaster.conf
multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user port=5003'
EOF

multimaster.referee_connstring = 'dbname=multimaster_db user=multimaster_user port=5003'
```

Перезапустим первый и второй узлы, чтобы применить настройки:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-1 stop

waiting for server to shut down.... done
server stopped

postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-2 stop

waiting for server to shut down.... done
server stopped

postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-1/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-1 start

waiting for server to start.... done
server started

student$ psql -p 5001 -U multimaster_user -d multimaster_db

postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-2/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-2 start

waiting for server to start.... done
server started

student$ psql -p 5002 -U multimaster_user -d multimaster_db
```

Инициализируем кластер мультимастера.

```
=> CREATE EXTENSION multimaster;

CREATE EXTENSION
```

Теперь запустим функцию инициализации кластера. Первый параметр — это строка подключения к текущему узлу, второй — подключение ко второму узлу:

```
=> SELECT mtm.init_cluster(
    'dbname=multimaster_db user=multimaster_user port=5001',
   '{"dbname=multimaster_db user=multimaster_user port=5002"}');

 init_cluster
--------------

(1 row)


=> SELECT * FROM mtm.status();

 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          1 | online | {1,2}     |       3 | {1,2}       | {1,2}              | {1,2}
(1 row)
```

Создадим таблицу multimaster_tbl и добавим в нее несколько строк:

```
=> CREATE TABLE multimaster_tbl (id integer);
```

```
CREATE TABLE
```

```
=> INSERT INTO multimaster_tbl VALUES (1),(2),(3);
```

```
INSERT 0 3
```

Проверим доступность таблицы multimaster_tbl на втором узле:

```
=> SELECT * FROM multimaster_tbl;
```

```
 id
----
  1
  2
  3
(3 rows)
```

---

2. Имитация сбоя и проверка работоспособности

Имитируем отказ узла. Для этого отключим один из узлов, например, второй:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -D /var/lib/pgpro/ent-13-MM-2 stop
```

```
waiting for server to shut down.... done
server stopped
```

```
=> SELECT * FROM mtm.status();
```

```
 my_node_id | status | connected | gen_num | gen_members | gen_members_online | gen_configured
------------+--------+-----------+---------+-------------+--------------------+----------------
          1 | online | {1}       |       4 | {1}         | {1}                | {1,2}
(1 row)
```

Добавим еще несколько строк в таблицу:

```
=> INSERT INTO multimaster_tbl VALUES (11),(22),(33);
```

```
INSERT 0 3
```

Оставшийся узел сохранил работоспособность.

---

3. Возвращение узла в строй

Вернем в строй второй узел:

```
postgres$ /opt/pgpro/ent-13/bin/pg_ctl -w -l /var/lib/pgpro/ent-13-MM-2/pgpro-ent.log -D /var/lib/pgpro/ent-13-MM-2 start
```

```
waiting for server to start.... done
server started
```

```
student$ psql -p 5002 -U multimaster_user -d multimaster_db
```

На втором узле проверим, что таблица доступна:

```
=> SELECT * FROM multimaster_tbl;
```

```
 id
----
  1
  2
  3
 11
 22
 33
(6 rows)
```

Кластер мультимастера продолжает работать в режиме 2+1.