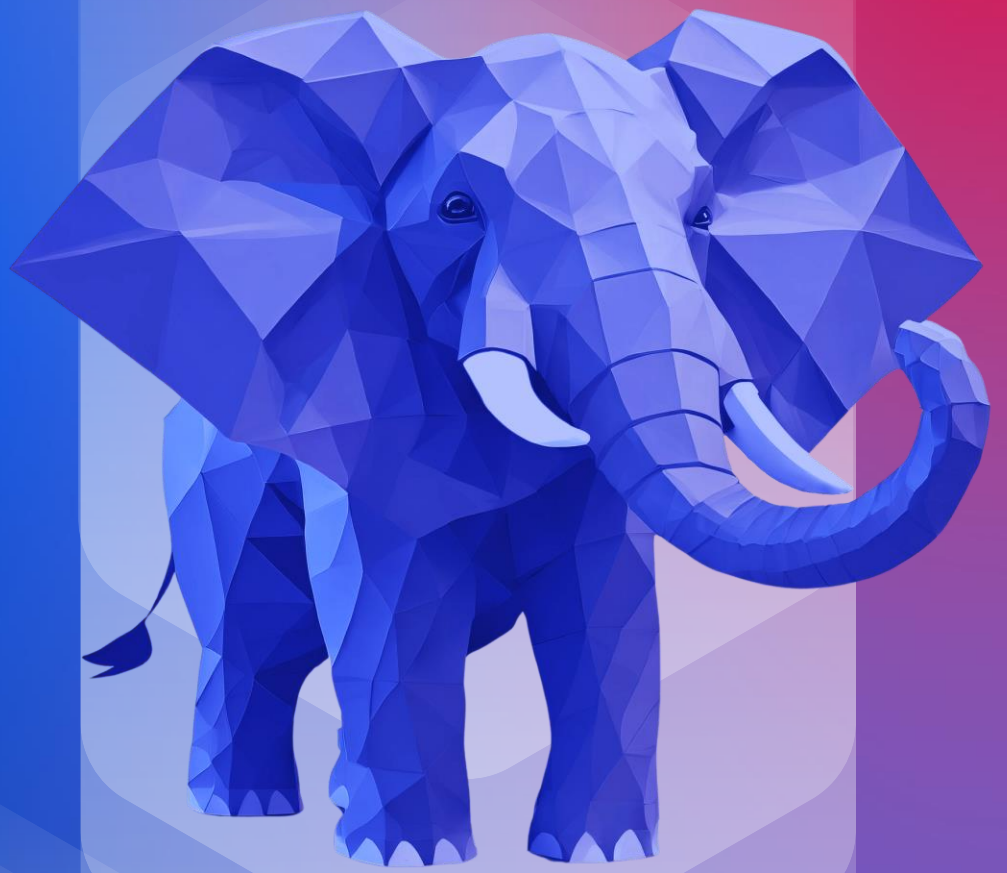
PostgresPro

Shardman – технология распределенной реляционной СУБД



Забелин Андрей

a.zabelin@postgrespro.ru

Shardman

PostgresPro

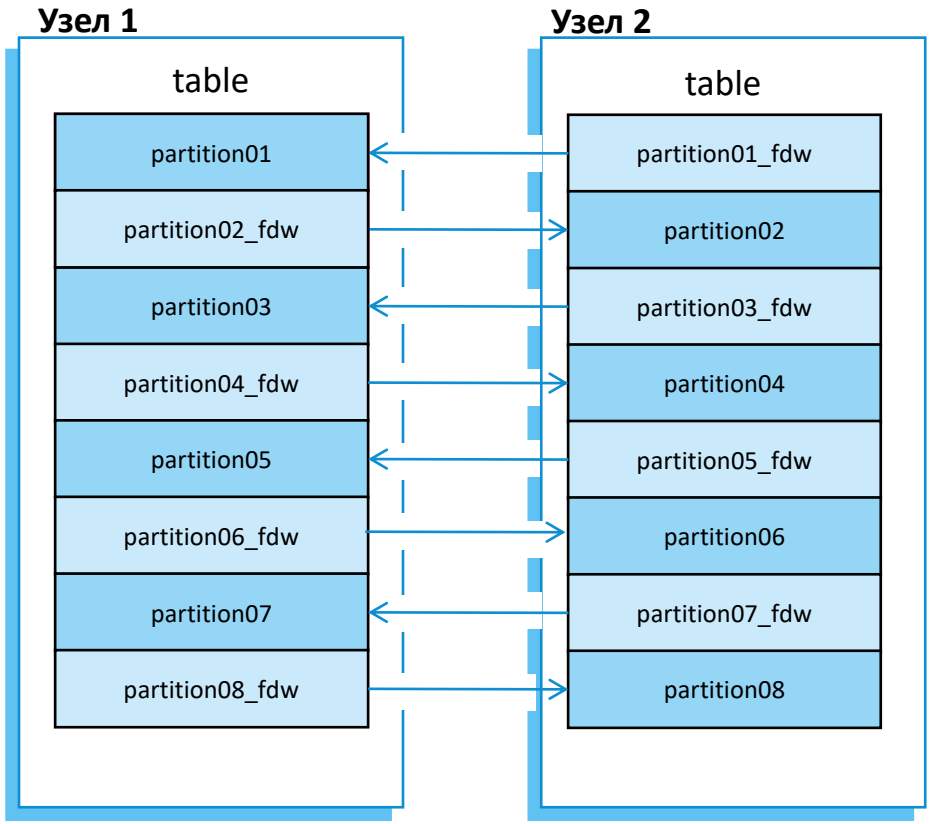
- Распределенная СУБД PostgreSQL
  - PostgreSQL
  - Расширение Shardman
  - Расширение postgres_fdw
- Shardmanctl – управление
- Etcd – хранение состояния и конфиурации
- Shardmand – отказоустойчивость

Shardman: преимущества

PostgresPro

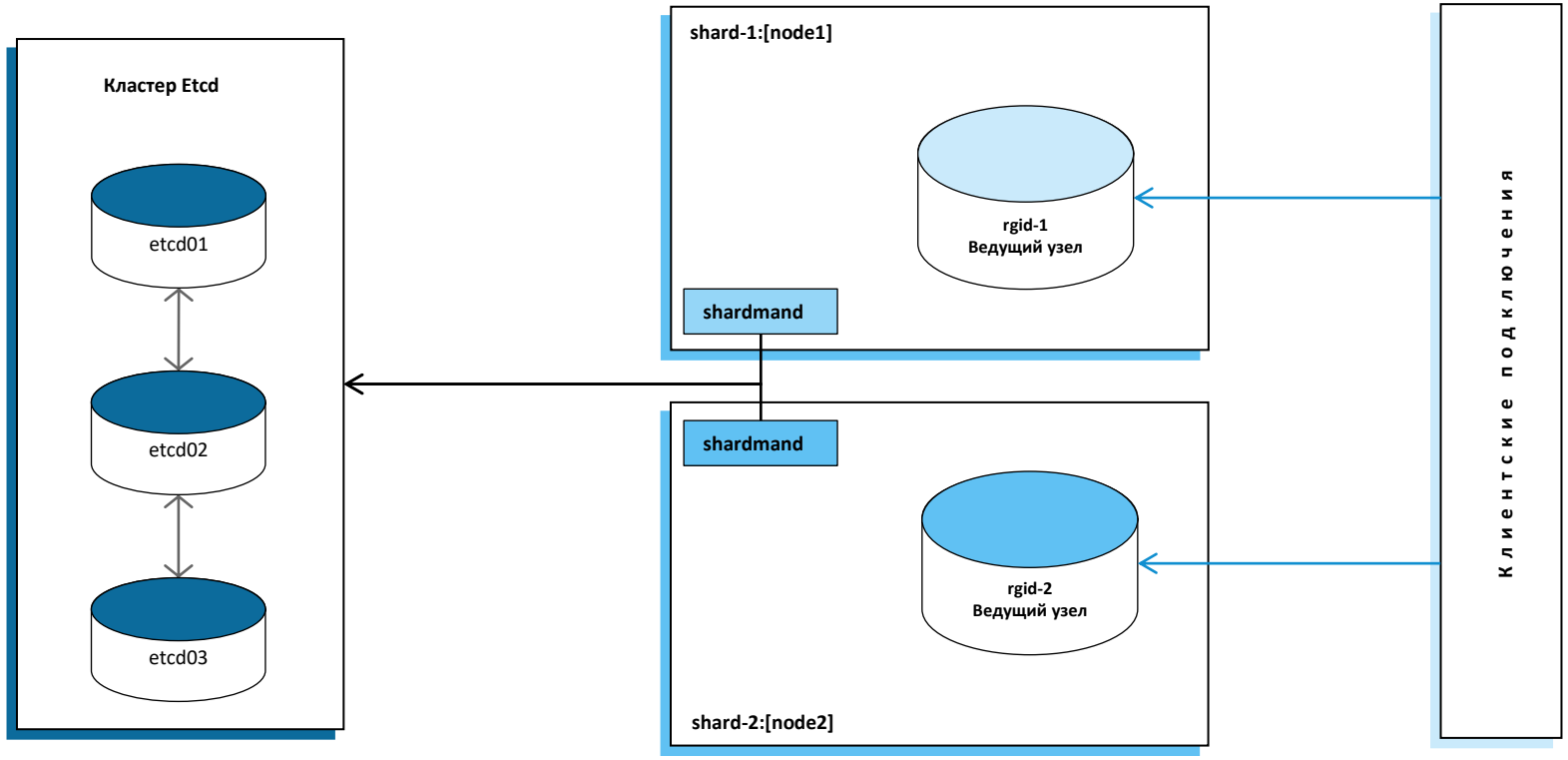
- Прозрачное горизонтальное масштабирование
- OLTP нагрузка
- Доступ в кластер через любой узел
- Избыточность данных:
  встроенная отказоустойчивость
- Целостность данных:
  распределенные транзакции, строгие гарантии ACID

Shardman: распределённая БД

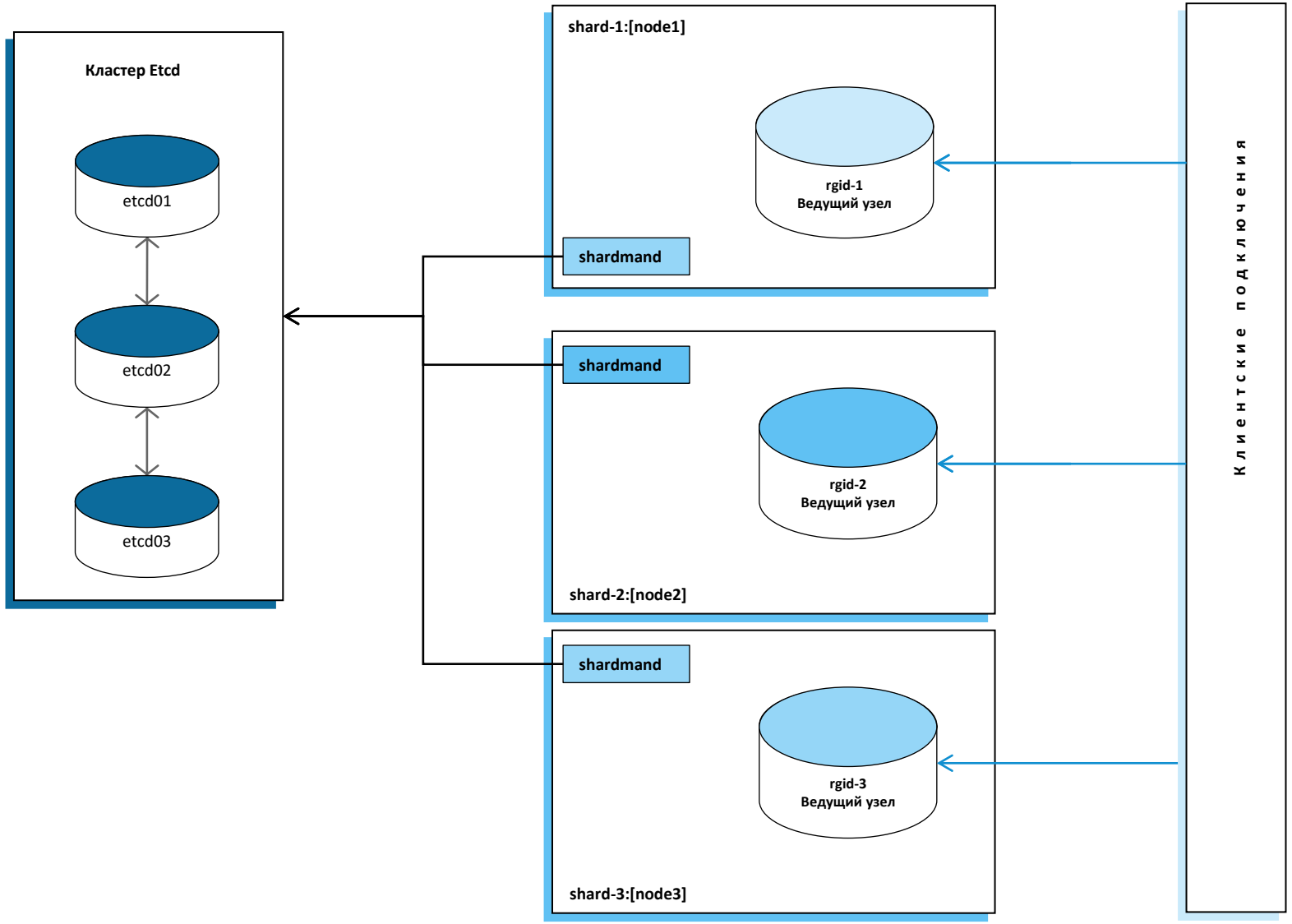


- Каждый узел содержит только часть данных таблиц
- Таблица состоит из набора секций, которые находятся физически на разных узлах кластера
- Секции с других узлов доступны через fdw
- Число секций фиксируется при создании таблицы

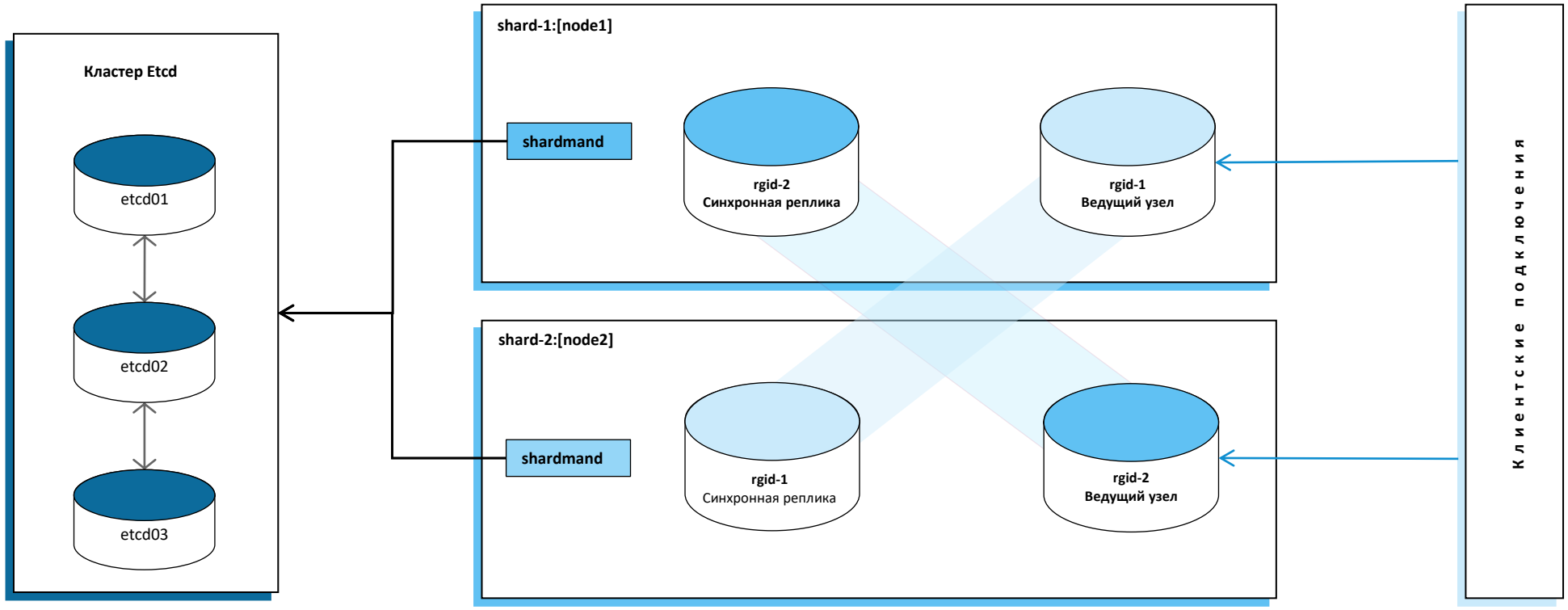
Архитектура Shardman



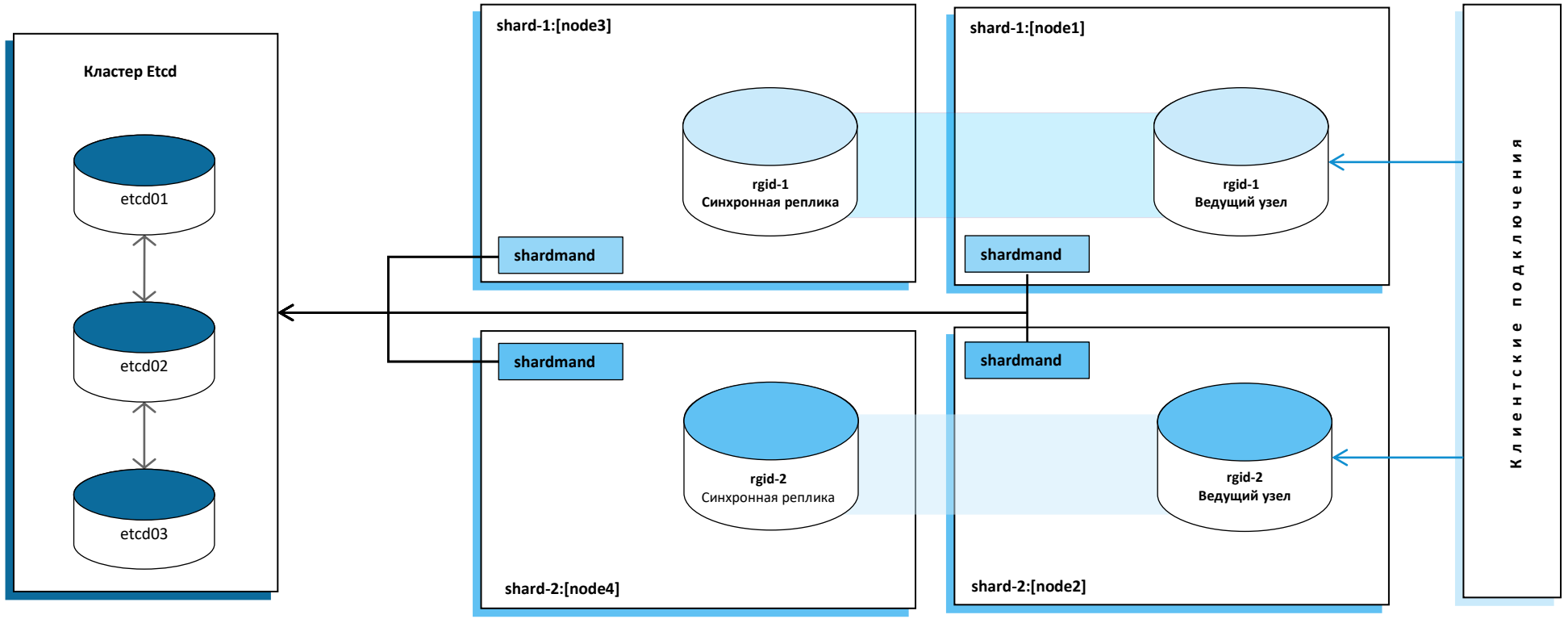
Масштабирование Shardman



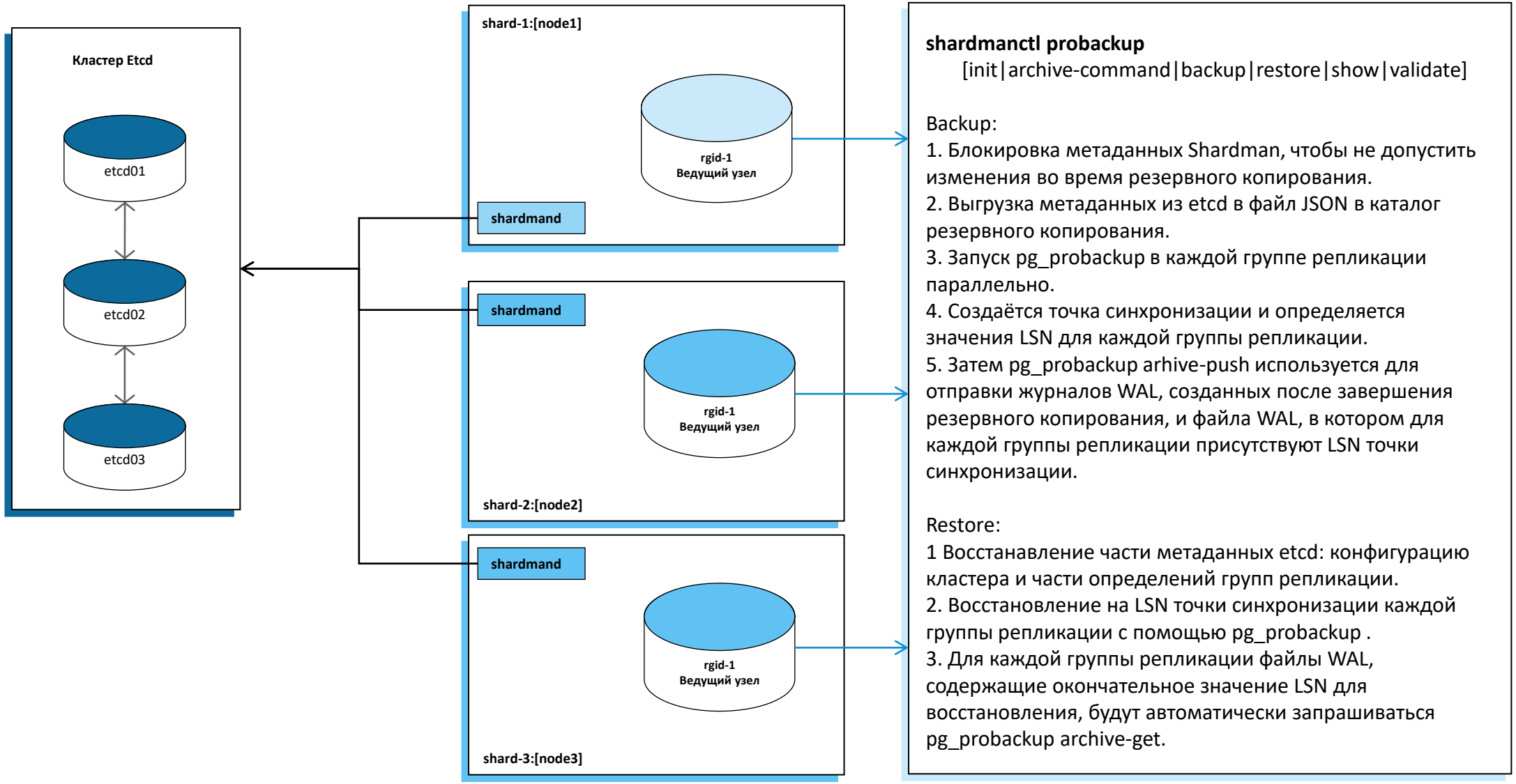
Отказоустойчивость Shardman



Отказоустойчивость Shardman



Резервное копирование Shardman



shardmanctl probackup
[init|archive-command|backup|restore|show|validate]

Backup:
1. Блокировка метаданных Shardman, чтобы не допустить изменения во время резервного копирования.
2. Выгрузка метаданных из etcd в файл JSON в каталог резервного копирования.
3. Запуск pg_probackup в каждой группе репликации параллельно.
4. Создаётся точка синхронизации и определяется значения LSN для каждой группы репликации.
5. Затем pg_probackup arhive-push используется для отправки журналов WAL, созданных после завершения резервного копирования, и файла WAL, в котором для каждой группы репликации присутствуют LSN точки синхронизации.

Restore:
1 Восстанавление части метаданных etcd: конфигурацию кластера и части определений групп репликации.
2. Восстановление на LSN точки синхронизации каждой группы репликации с помощью pg_probackup .
3. Для каждой группы репликации файлы WAL, содержащие окончательное значение LSN для восстановления, будут автоматически запрашиваться pg_probackup archive-get.

Shardman: виды таблиц

Шардированная таблица — это секционированная таблица, где часть секций являются обычными локальными секциями таблицы, а остальные — внешними секциями таблицы через fdw.

Количество секций определяется при создании шардированной таблицы, оно должно быть больше или равно планируемого количества узлов в распределенной СУБД. Количество секций нельзя изменить.

Несколько шардированных таблиц могут быть размещены совместно.

Секции совмещённых таблиц , соответствующие одному и тому же ключу шардирования, находятся на одном узле. Совместное размещение необходимо для того, чтобы операция соединения нескольких таблиц (JOIN) выполнялись на узле, где находятся фактические данные.

Глобальные таблицы дублируются на все узлы и используются для редко обновляемых данных

Например наиболее подходящими для этого являются таблицы справочники. Когда выполняется соединение шардированной таблицы с глобальной таблицей, оно выполняется на узле, где находятся данные. При изменении данных в глобальной таблице, изменения данных одновременно происходят на всех узлах кластера.

bookings

book_ref	book_date	amount
12345	01.01.2023	300
34567	20.03.2023	1000
56789	13.07.2023	1300

tickets

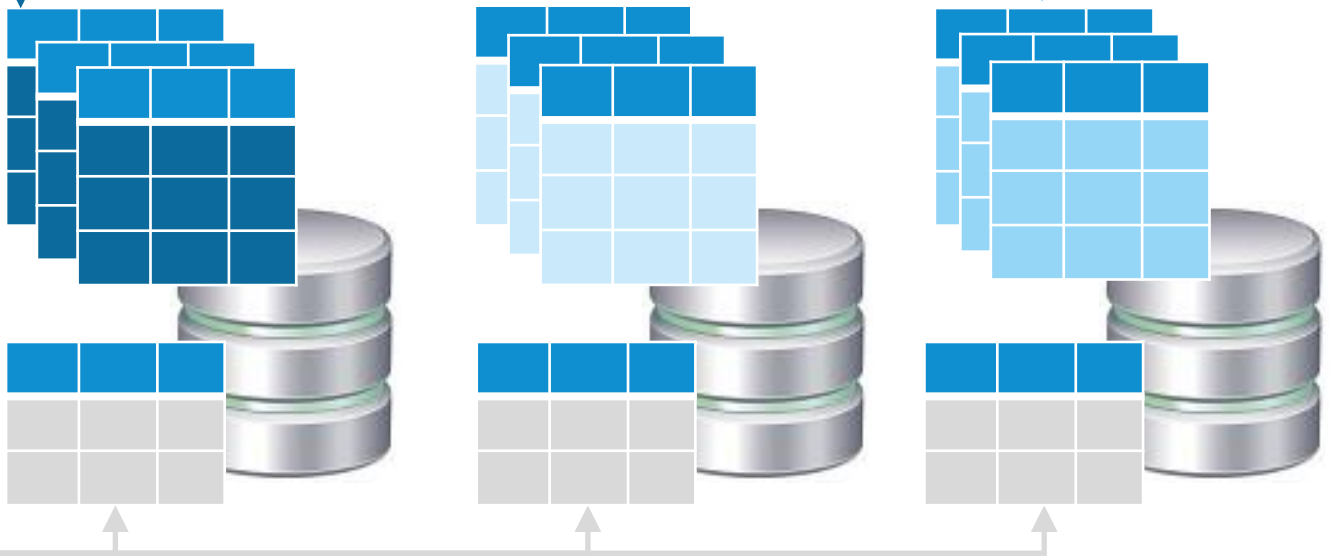
book_ref	ticket_no	passenger_name
12345	12345678	Андрей
34567	34567890	Сергей
56789	56789123	Иван
34567	34567891	Екатерина
56789	56789124	Анна

ticket_flights

book_ref	ticket_no	flight_id
12345	12345678	U755
34567	34567890	F444
56789	56789123	R234
34567	34567891	F444
56789	56789124	R234

flights

flight_id	departure_airport	arrival_airport
U755	Москва	Самара
F444	Владивосток	Москва
R234	Самара	Сочи



Shardman: пример схемы данных

PostgresPro

```
CREATE TABLE bookings (
   book_ref character(6) NOT NULL PRIMARY KEY,
   book_date timestamp with time zone NOT NULL,
   amount numeric(10,2) NOT NULL
) WITH (distributed_by='book_ref', num_parts=4);
```

```
CREATE TABLE tickets (
   ticket_no character(13) NOT NULL,
   book_ref character(6)
       REFERENCES bookings(book_ref),
   passenger_name text NOT NULL,
   PRIMARY KEY (book_ref, ticket_no)
) WITH (distributed_by='book_ref',
colocate_with='bookings’);
```

```
CREATE TABLE ticket_flights (
   ticket_no character(13) NOT NULL,
   flight_id bigint NOT NULL,
   book_ref character(6) NOT NULL,
   FOREIGN KEY (book_ref, ticket_no)
       REFERENCES tickets(book_ref, ticket_no),
   PRIMARY KEY (book_ref, ticket_no, flight_id)
) WITH (distributed_by='book_ref',
colocate_with='bookings’);
```

```
CREATE TABLE flights (
   flight_id bigserial NOT NULL PRIMARY KEY,
   departure_airport character(10),
   arrival_airport character(10)
) WITH (global);
```

Shardman: запросы

PostgresPro

Запрос возвращает все билеты из указанного бронирования:

```
SELECT t.*, b.*
FROM bookings b
JOIN tickets t
   ON t.book_ref = b.book_ref
WHERE b.book_ref = '0824C5';
```

План запроса:

Если данные находятся на другом шарде:

```
Foreign Scan (actual rows=2 loops=1)
  Relations: (bookings_2_fdw b) INNER JOIN (tickets_2_fdw t)
  Network: FDW bytes sent=433 received=237
```

Shardman: запросы

PostgresPro

Запрос возвращает все перелёты по всем билетам в указанном бронировании

```
SELECT tf.*, t.*
FROM tickets t
JOIN ticket_flights tf
    ON tf.ticket_no = t.ticket_no
    AND tf.book_ref = t.book_ref
WHERE t.book_ref = '0824C5';
```

План запроса:

Если данные находятся на другом шарде:

```
Foreign Scan (actual rows=12 loops=1)
  Relations: (tickets_2_fdw t) INNER JOIN (ticket_flights_2_fdw tf)
  Network: FDW bytes sent=547 received=1717
```

Shardman: запросы

PostgresPro

Запрос возвращает статистика по количеству пассажиров на одно бронирование
(сначала посчитаем количество пассажиров в каждом бронировании, а затем количество бронирований с каждым вариантом количества пассажиров)

```
SELECT tt.cnt, count(*)
FROM (
    SELECT count(*) cnt
    FROM tickets t
    GROUP BY t.book_ref
    ) tt
GROUP BY tt.cnt
ORDER BY tt.cnt;
```

План запроса:

```
Sort (actual rows=5 loops=1)
  Sort Key: (count(*))
  Sort Method: quicksort  Memory: 25kB
  Network: FDW bytes sent=798 received=14239951
  -> HashAggregate (actual rows=5 loops=1)
      Group Key: (count(*))
      Batches: 1  Memory Usage: 40kB
      Network: FDW bytes sent=798 received=14239951
      -> Append (actual rows=593433 loops=1)
          Network: FDW bytes sent=798 received=14239951
          -> GroupAggregate (actual rows=148504 loops=1)
              Group Key: t.book_ref
              -> Index Only Scan using tickets_0_book_ref_idx on tickets_0 t (rows=207273)
                  Heap Fetches: 0
          -> Async Foreign Scan (actual rows=148256 loops=1)
              Relations: Aggregate on (tickets_1_fdw t_1)
              Network: FDW bytes sent=266 received=1917350
          -> Async Foreign Scan (actual rows=148270 loops=1)
              Relations: Aggregate on (tickets_2_fdw t_2)
              Network: FDW bytes sent=266
          -> Async Foreign Scan (actual rows=148403 loops=1)
              Relations: Aggregate on (tickets_3_fdw t_3)
              Network: FDW bytes sent=266
```

Shardman: запросы

PostgresPro

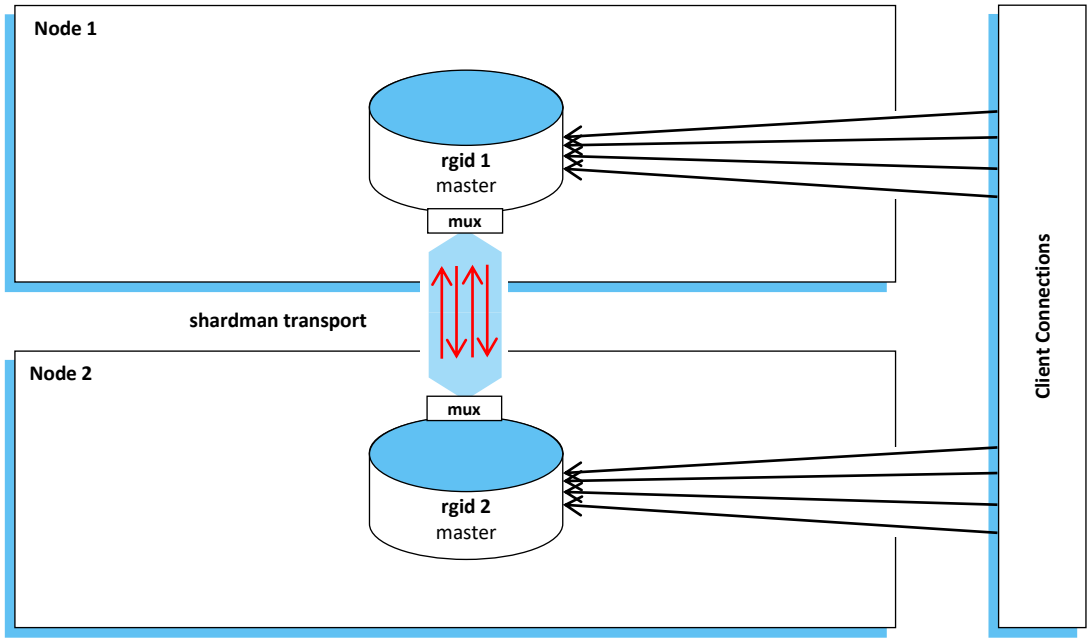
Запрос возвращает, какие сочетания имён встречаются чаще всего, и какую долю от числа всех пассажиров составляют такие сочетания

```
SELECT passenger_name,
    round( 100.0 * cnt / sum(cnt) OVER (), 2)
  AS percent
FROM (
 SELECT passenger_name,
      count(*) cnt
 FROM tickets
 GROUP BY passenger_name
) t
ORDER BY percent DESC;
```

План запроса:

```
Sort (actual rows=27909 loops=1)
  Sort Key: (round(((100.0 * ((count(*)))::numeric) / sum((count(*))) OVER (?)), 2)) DESC
  Sort Method: quicksort  Memory: 3076kB
  Network: FDW bytes sent=816 received=2376448
  -> WindowAgg (actual rows=27909 loops=1)
      Network: FDW bytes sent=816 received=2376448
      -> Finalize HashAggregate (actual rows=27909 loops=1)
          Group Key: tickets.passenger_name
          Batches: 1  Memory Usage: 5649kB
          Network: FDW bytes sent=816 received=2376448
          -> Append (actual rows=74104 loops=1)
              Network: FDW bytes sent=816 received=2376448
              -> Partial HashAggregate (actual rows=18589 loops=1)
                  Group Key: tickets.passenger_name
                  Batches: 1  Memory Usage: 2833kB
                  -> Seq Scan on tickets_0 tickets (actual rows=207273 loops=1)
              -> Async Foreign Scan (actual rows=18435 loops=1)
                  Relations: Aggregate on (tickets_1_fdw tickets_1)
                  Network: FDW bytes sent=272 received=2376448
              -> Async Foreign Scan (actual rows=18567 loops=1)
                  Relations: Aggregate on (tickets_2_fdw tickets_2)
                  Network: FDW bytes sent=272
              -> Async Foreign Scan (actual rows=18513 loops=1)
                  Relations: Aggregate on (tickets_3_fdw tickets_3)
                  Network: FDW bytes sent=272
```

Shardman: мультиплексирование fdw



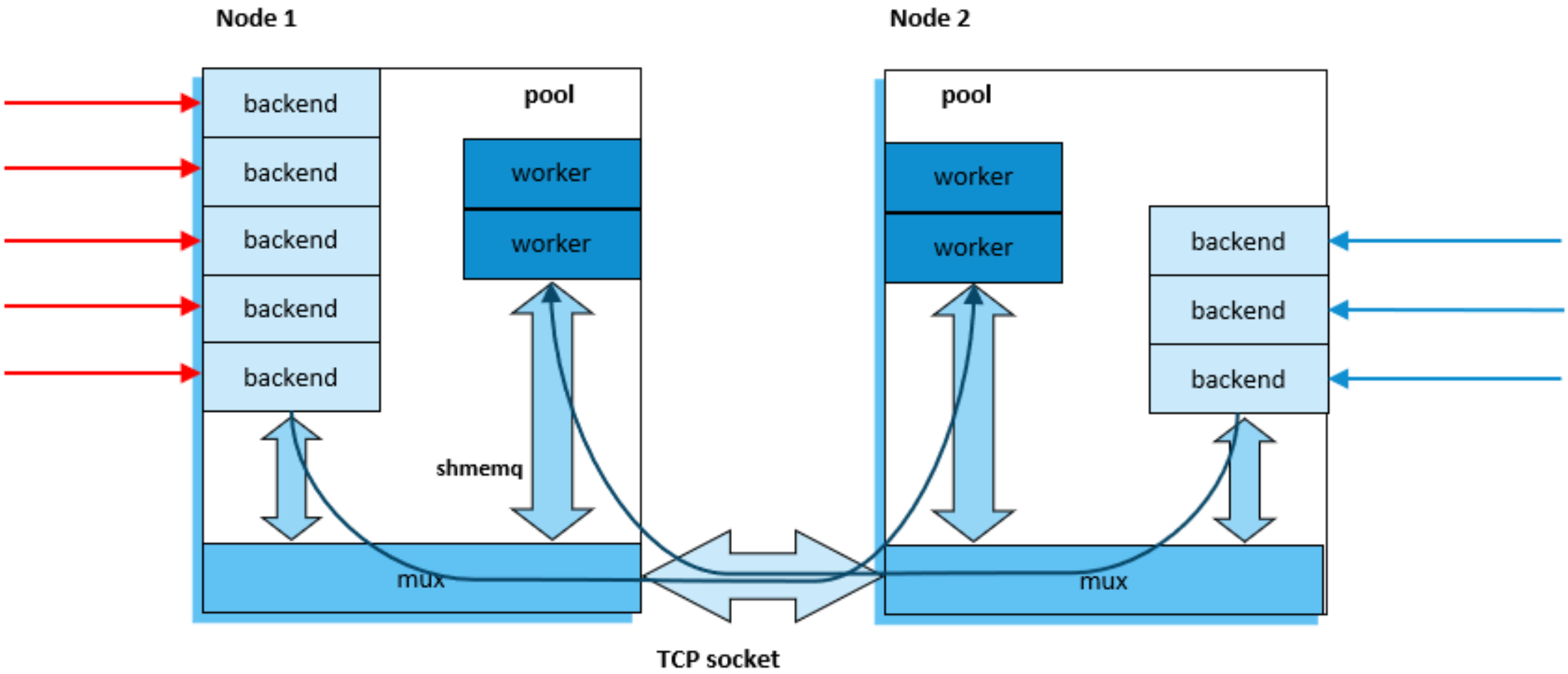
Общее число подключений кластере – **M+(N x N)**

Общее число процессов в кластере – **M+(N x W)**

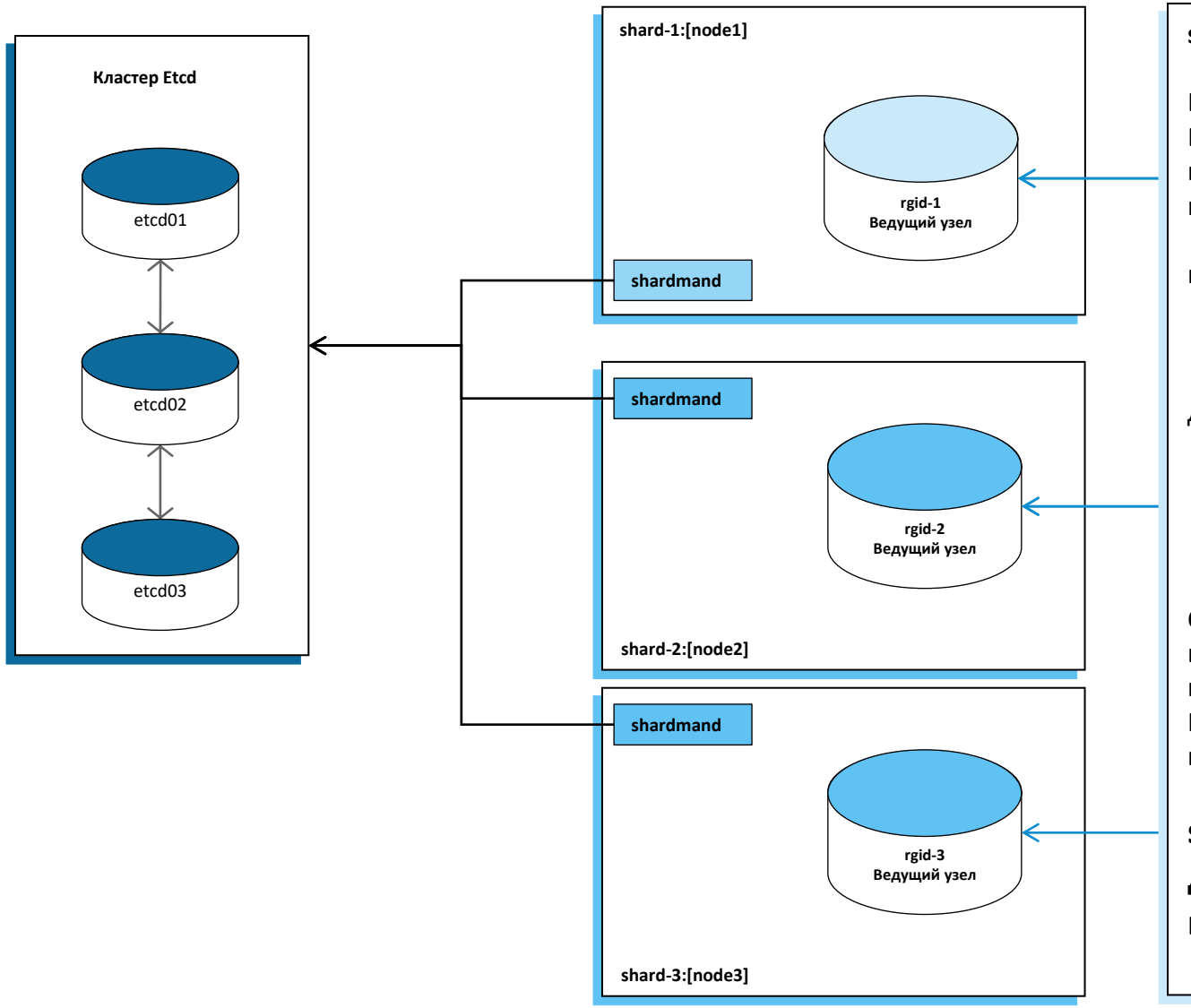
M – количество подключений,
N – количество узлов,
W – количество воркеров

2 узла, 8 подключений = 8+4
2 узла, 50 подключений = 50+4

Shardman: мультиплексирование fdw



Загрузка данных Shardman



shardmanctl load --schema load_schema.yaml

В yaml файле определён источник данных (CSV или другая БД) и некоторые параметры работы утилиты:
migrate.jobs количество параллельных процессов загрузки
migrate.batch задаёт количество строк в одной порции

migrate.schema определяет список схем
migrate.schemas.tables : список таблиц в схеме
два типа таблиц: global и sharded.

Для шардированной таблицы нужно указать:
distributedby - имя столбца, используемого для секционирования таблицы
partitions - количество секций, которые будут созданы для этой таблицы

Сначала загружаются глобальные (global) таблицы, затем шардированные (sharded) таблицы и в конце шардированные таблицы с параметром colocatedwith. Порядок загрузки таблиц одного типа можно изменить с помощью параметра priority

shardmanctl load умеет распределять данные при миграции данных сразу в тот шард, где они должны храниться

Shardman: возможности Postgres Pro Enterprise

PostgresPro

- механизм сжатия данных CFS,
- pg_probackup : инкрементальный бэкап на уровне страниц, используя PTRACK,
- расширение pgpro_stats : сбор статистики планирования и выполнения распределённых запросов, затрагивающих несколько узлов в кластере

Дополнительная информация

PostgresPro

- Описание
  https://postgrespro.ru/products/shardman
- Документация
  https://postgrespro.ru/docs/shardman/14
- Презентации
  PGCONF.RUSSIA 2024 08–09 АПРЕЛЯ https://pgconf.ru/2024

PostgresPro

Спасибо за внимание!

PostgresPro

Q&A