Максим Милютин
milyutinma@gmail.com

Аналитические open-source решения на базе PostgreSQL:

Greenplum vs. CitusDB vs. TimescaleDB vs. OpenGauss

О докладе / докладчике

- О себе:
  - Работаю в Московском исследовательском центре Huawei
  - Занимаюсь исследованиями движков СУБД (разработка на базе OpenGauss / GaussDB)
  - Являюсь контрибьютором в PostgreSQL / Greenplum и расширения pg_wait_sampling
- О докладе:
  - Не связан (почти) с моей текущей рабочей деятельностью, родился на предыдущем месте работы - компания Arenadata
  - Мотивация:
    - Запрос от сообщества - дать оценку аналитическим решениям на базе PostgreSQL (Greenplum vs. CitusDB)
    - Сформировать методологию сравнения аналитических СУБД для объективной оценки Greenplum
  - Сравнение с позиции разработчика СУБД

Основный темы для сравнения

- Построение кластера для организации распределенной обработки запросов по канонам MPP (massive parallel processing)
- Колоночный движок хранения
- Специфичные для аналитических СУБД фичи

Объекты сравнения

- Greenplum - MPP форк PostgreSQL, развиваемый с середины 2000-х
  - текущая 6 версия базируется на PostgreSQL 9.4
  - создавался как альтернатива коммерческой Terradata
- CitusDB - расширение для горизонтального масштабирования PostgreSQL [1]
- TimescaleDB - расширение для хранения / обработки timeseries данных
  - содержит интересные функции для аналитической обработки
- OpenGauss - открытый форк GaussDB (форк Postgres-XC) для замены Oracle
  - имеет продвинутый колоночный движок
  - не поддерживает распределенную обработку запросов в отличие от закрытой GaussDB

1. https://dl.acm.org/doi/10.1145/3448016.3457551

Организация кластера для MPP

- Схема кластера
  - Гомогенная vs. гетерогенная структура
  - Способы распределения строк в таблицах
  - Масштабирование / балансировка кластера
- Распределенное исполнение запросов по канонам MPP
- Поддержка распределенных транзакций
  - Глобальное консистентное чтение в кластере
  - Атомарность распределенной модификации данных
- Высокая доступность кластера
  - Поддержка физического бэкапа и PITR

Структура кластера

- Гетерогенная
  - Два вида узлов: координатор и сегменты / воркеры / узлы данных (Greenplum, CitusDB, TimescaleDB)
  - Координаторов может быть несколько (GaussDB)
- Гомогенная
  - Все узлы совмещают роль координатора и сегментов (CitusDB с 11 версии)
- Проблема сильной связности между узлами
  - Greenplum: кастомный протокол для интерконнекта:
    - на базе UDP - *UDPIFC*
    - прокси для уменьшения кол-ва используемых портов
  - CitusDB: внешний пуллер потоков (pgBouncer) между узлами

Распределение строк в таблицах. Greenplum

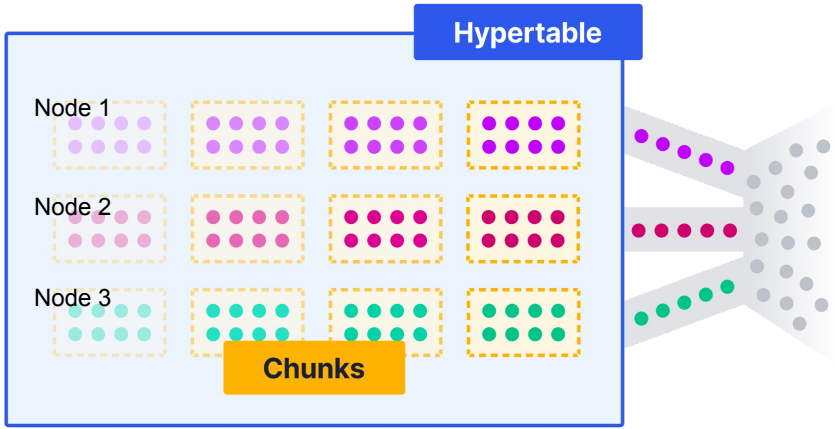
- По хэш значению выбранной колонки / колонок распределения
- Схема консистентного хэширования
- Есть опция случайного распределения
  - совершенная равномерность && нет конкретной политики распределения
- Поддерживаются реплицированные таблицы (с 6 версии)
  - для словарей / таблиц измерений в схеме “звезда”

Распределение строк в таблицах. CitusDB

- По хэш значению выбранной колонки / колонок распределения
- Range-based схема хэширования
  - каждому логическому шарду (секции) соответствует определённый диапазон хэш значений
- Множество логических шардов расположены на одном узле
- *“Co-located”* таблицы, у которых границы хэш значений ключа распределения совпадают
  - для выполнения локального join`а
- *“Referenced”* таблицы как реплицированные

Распределение строк в таблицах. TimescaleDB

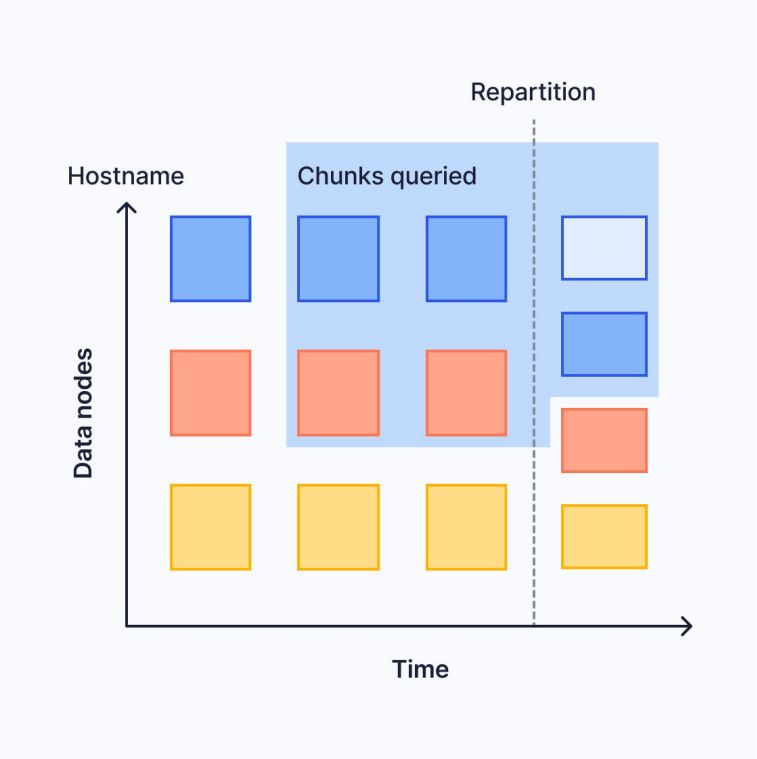
- *“Hypertable”* - хранящая метрики таблица, секционированная по диапазону от времени (“*time*” колонка) и опционально по хэшу от *“space”* колонки
- Строки распределяются по хэшу от *“space”* колонки - матричная структура
  - используется range-based схема хэширования
- По умолчанию, один диапазон хэш-значений на один узел
  - перемещая секции из одного узла на другой, можно добиться схемы множества диапазонов на один узел



Масштабирование / балансировка кластера. CitusDB

- Миграция логических шардов на новый / существующий узел через логическую репликацию
- Во время миграции шард может принимать пишущую / читающую нагрузку
- Блокировка на запись (как правило, короткоживущая) берётся лишь во время конечной синхронизации и перезаписи метаданных о расположении шарда
- Ручное масштабирование / сжатие кластера, ребалансировка данных

Масштабирование / балансировка кластера. TimescaleDB



- Расширение кластера через добавление узла в схему распределения hypertable
- Новый узел будет использоваться, начиная со следующего временного интервала новых секций
  - запросы, включающие границу интервалов, будут обращаться к нескольким узлам
- Миграция секций на другой узел через логическую репликацию (экспериментальная фича)
- Сжатие кластера через миграцию существующих секций и отвязку узла от hypertable

Схема кластера. Сводная таблица

	Greenplum	CitusDB	TimescaleDB	OpenGauss
Структура кластера	Гетерогенная с одним координатором	Оба варианта	Гетерогенная с одним координатором	GaussDB: гетерогенная с множеством координаторов
Распределение строк	• по хэшу • случайно • реплицировано	• по хэшу • реплицировано	по хэшу от “space” колонки	GaussDB: • по хэшу • по диапазону • по списку значений • реплицировано
Масштабирование / балансировка	Только расширение под эксклюзивной блокировкой на ребалансируемые таблицы	Полная поддержка с короткой блокировкой	Полная поддержка. Расширение только для новых интервалов. Балансировка, сжатие экспериментальны	GaussDB: полная поддержка с короткой блокировкой

Распределенное исполнение запросов по канонам MPP

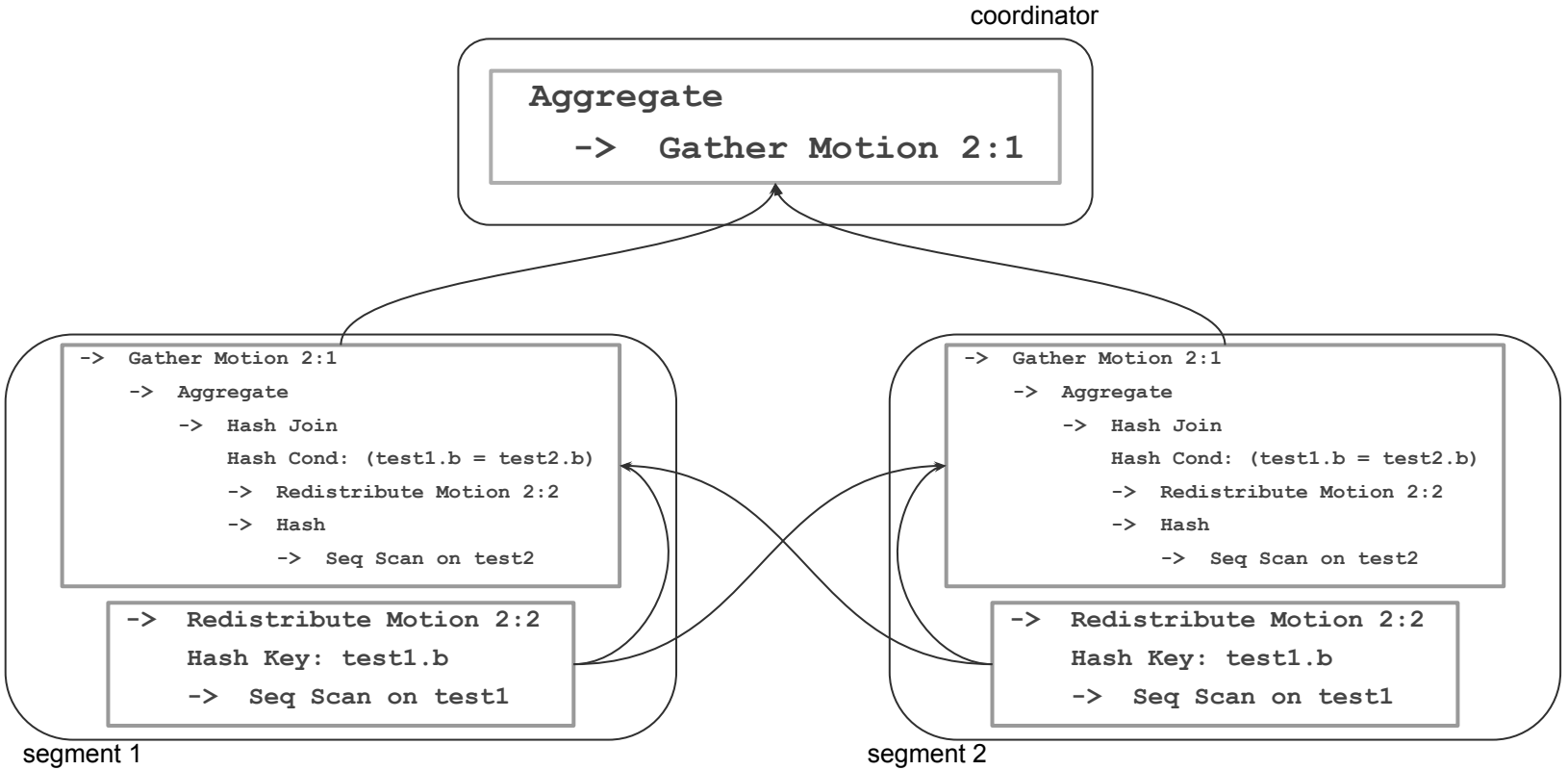
- Поток данных “сегменты → координатор” в примитивном случае
  - Проброс предикатов, локальных join`ов и group by, агрегатов на сегменты
- MPP предполагает перераспределение потока данных от дочерних / промежуточных узлов дерева плана для большего распараллеливания обработки
- Узлы “перераспределения” в Greenpum:
  - *Redistribute Motion* - по хэшу выбранного ключа
  - *Broadcast Motion* - реплицирование по всем узлам кластера
  - *Gather Motion* - сбор данных с сегментов на одном узле (координаторе)
- Более продвинутая техника сочетает Redistribute и Broadcast режимы [1]

1. https://dl.acm.org/doi/abs/10.1145/1376616.1376720

Распределенное исполнение запросов по канонам MPP. Greenplum

```
-- Таблица test1 соединяется не по ключу распределения (shuffle join)
explain (costs off)
select count(1) from test1 join test2 using (b);
                          QUERY PLAN
------------------------------------------------------------
Aggregate                                                    slice0
  ->  Gather Motion 2:1  (slice2; segments: 2)                slice2
        ->  Aggregate
              ->  Hash Join
                  Hash Cond: (test1.b = test2.b)
                  ->  Redistribute Motion 2:2  (slice1; segments: 2)
                        Hash Key: test1.b                    slice1
                        ->  Seq Scan on test1
                  ->  Hash
                        ->  Seq Scan on test2
```

Распределенное исполнение запросов по канонам MPP. Greenplum



Распределенное исполнение запросов по канонам MPP. Greenplum

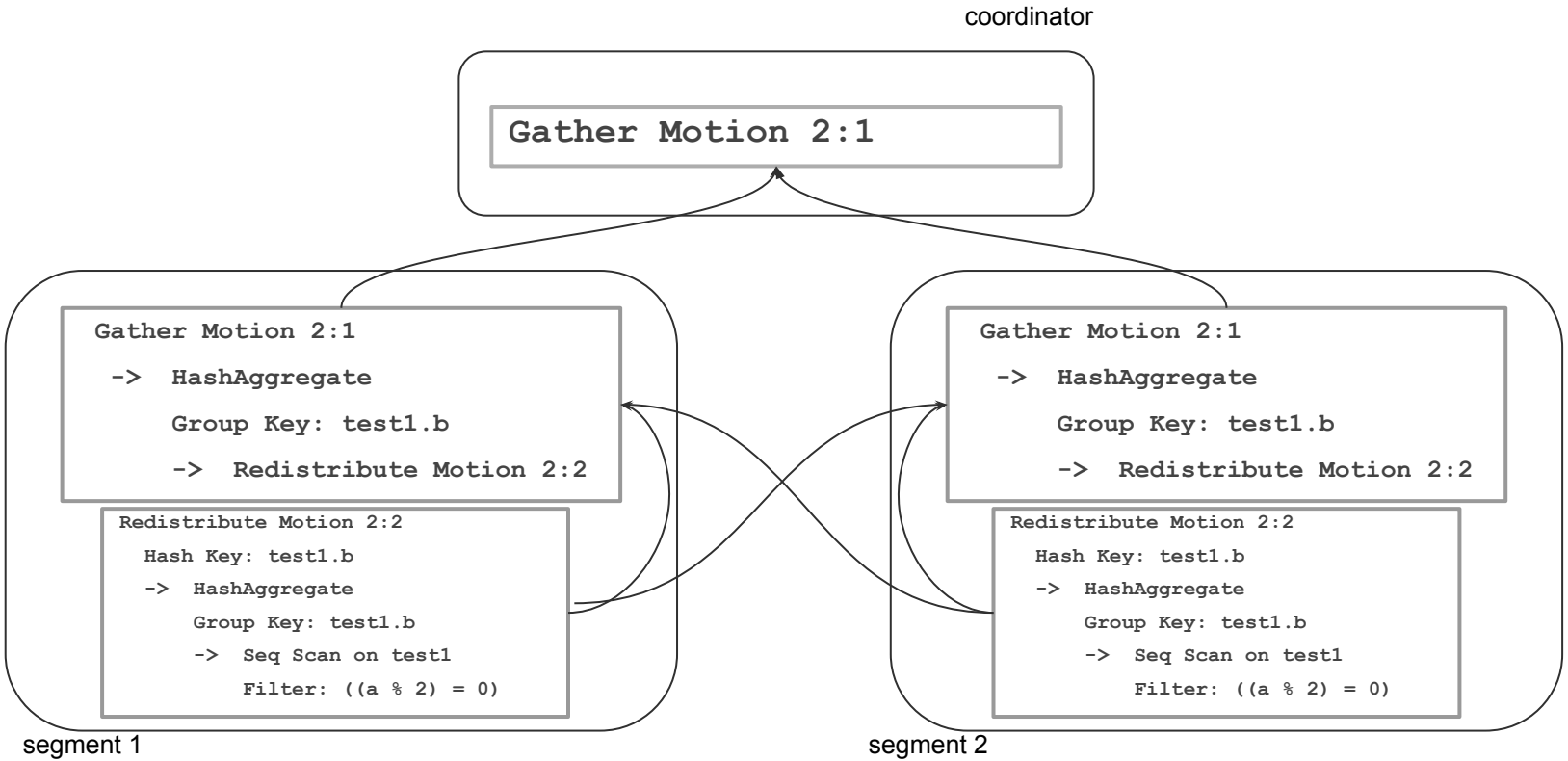
```
-- Таблица test1 группируется не по ключу распределения
explain (costs off)
select count(1) from test1 where a%2=0 group by b;
                          QUERY PLAN
------------------------------------------------------------------
 Gather Motion 2:1  (slice2; segments: 2)
   ->  HashAggregate
         Group Key: test1.b
         ->  Redistribute Motion 2:2  (slice1; segments: 2)
               Hash Key: test1.b
               ->  HashAggregate
                     Group Key: test1.b
                     ->  Seq Scan on test1
                           Filter: ((a % 2) = 0)
```

slice0

slice2

slice1

Распределенное исполнение запросов по канонам MPP. Greenplum



Распределенное исполнение запросов по канонам MPP. Greenplum

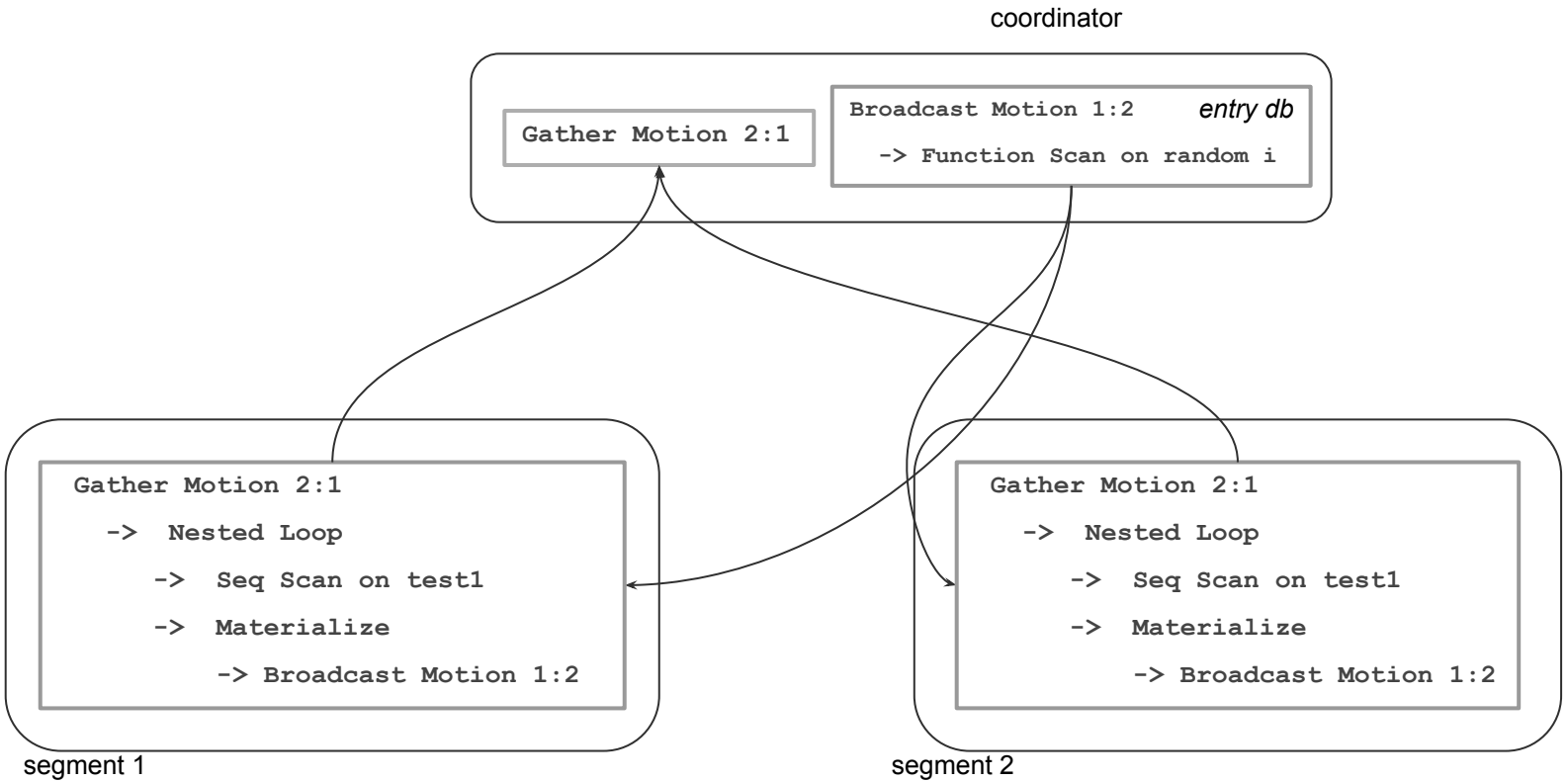
```
-- Volatile функция random() должна выполняться на одном узле
explain (costs off)
select * from test1 cross join random() i;
                          QUERY PLAN
-----------------------------------------------------
Gather Motion 2:1  (slice2; segments: 2)
  ->  Nested Loop
        ->  Seq Scan on test1
        ->  Materialize
              ->  Broadcast Motion 1:2  (slice1)
                    ->  Function Scan on random i
```

slice0

slice2

slice1

Распределенное исполнение запросов по канонам MPP. Greenplum



Распределенное исполнение запросов по канонам MPP

- *Motion* узлы могут находиться в любом месте дерева плана
- Поток данных может быть между сегментами, идти от сегментов к координатору и обратно по многу раз
- Задача планировщика - выбрать оптимальное расположение “правильных” типов *Motion* узлов
  - основная сложность в реализации распределенного SQL
- Greenplum предлагает два вида планировщика:
  - *“legacy”* - на базе постгресового bottow-up оптимизатора
  - *ORCA* - cascade top-down оптимизатор [1]

1. https://dl.acm.org/doi/10.1145/2588555.2595637

Распределенное исполнение запросов по канонам MPP. Citus

- Логика распределенного запроса скрыта в кастомном узле *CitusAdaptive*
  - набор задач (SQL запросы по большей части) для выполнения на сегментах
  - набор подпланов, результаты которых распределяются по всем узлам / перераспределяются по ключу(?)
- Поддерживается перераспределение данных для join`а (shuffle join) опцией *enable_repartition_joins*
  - задача *MapMergeJob* (детали join`а в *explain* скрыты)
- Не поддерживается перераспределение результатов группировки

Распределенное исполнение запросов по канонам MPP. Citus

```
-- Таблица test1 джоинится не по ключу распределения с результатом группировки не по ключу таблицы test2
explain (costs off)
select count(1) from test1 join (select distinct a from test2) t(b) using(b);
```

```
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
Aggregate
  ->  Custom Scan (Citus Adaptive)
        ->  Distributed Subplan 27_1
              ->  HashAggregate
                    Group Key: remote_scan.a
                    ->  Custom Scan (Citus Adaptive)
                          Task Count: 32
                          Tasks Shown: One of 32
                          ->  Task
                                Node: host=localhost port=5001 dbname=postgres
                                ->  HashAggregate
                                      Group Key: a
                                      ->  Seq Scan on test2_102040 test2
        Task Count: 32
        Tasks Shown: One of 32
        ->  Task
              Node: host=localhost port=5001 dbname=postgres
              ->  Aggregate
                    ->  Hash Join
                          Hash Cond: (intermediate_result.a = test1.b)
                          -> Function Scan on read_intermediate_result intermediate_result
                          ->  Hash
                                ->  Seq Scan on test1_102008 test1
```

подплан `select distinct a from test2` результат которого реплицируются / перераспределяются по worker узлам(?)

обращение к результатам подплана

Поддержка распределенных транзакций. Консистентное чтение с кластера

- PostgreSQL из коробки не поддерживает распределенные “снимки данных” (global snapshots)
  - Расширения наследуют этот недостаток
  - PostgresPro предлагает патч синхронизации CSN (Commit Sequence Number - номер завершенной транзакции) на базе алгоритма *Clock-SI* [1]
- Greenplum формирует snapshot на координаторе
  - Имеет аналогичную структуру что и локальный постгресовый snapshot
  - Поддерживается глобальный счетчик транзакций (xid)
  - Каждый сегмент осуществляет трансформацию локального xid в глобальный для проверки строки на глобальную видимость
  - Отображение локального xid в глобальный хранится в специальном логе (типа, лога статуса транзакций pg_xact)

1. https://www.postgresql.org/message-id/21BC916B-80A1-43BF-8650-3363CCDAE09C%40postgrespro.ru , https://www.postgresql.org/message-id/07b2c899-4ed0-4c87-1327-23c750311248%40postgrespro.ru

Поддержка распределенных транзакций. Атомарность модификаций

- PostgreSQL из коробки поддерживает XA протокол двухфазной фиксации транзакций (2PC)
- Требуется сохранение состояния 2PC на координаторе для процесса восстановления
- Greenplum:
  - изменение статусов транзакций сохраняет в WAL координатора
  - состояние координатора резервируется синхронной репликой

High Availability. Greenplum

- Каждый узел, включая координатор, резервируется **только одной** синхронной репликой
  - При выпадении этой реплики ведущий узел переключается в асинхронный режим репликации, чтобы не блокировать запись в сегмент
- Автофейловер для сегментов осуществляет специальный worker на координирующем узле FTS (Fault Tolerance Server)
  - Автоматически только промоутится резервный узел
  - Ручной возврат бывшего мастера с кластер (утилита *gp_restore*)
- Failover для координирующего узла **только ручной**
  - Попытки добавить автоматический [1]

1. https://groups.google.com/a/greenplum.org/g/gpdb-dev/c/CHNFp_yeX1M/m/nof14jCkAAAJ

High availability. TimescaleDB

- Резервирование экземпляра БД через физическую репликацию
  - автофейловер внешним средством, рекомендуется *patroni*
  - можно зарезервировать все виды узлов
- “Нативная” репликация секций
  - гранулировано для секции задается *фактор репликации* - кол-во реплик на всех сегментах кластера
  - консистентное изменение данных происходит с координатора
  - шардинг с избыточностью на логическом уровне
  - требуется внешний арбитр для пометки упавшего узла недоступным
  - после пометки чтение и запись перенаправляется на узел с резервной копией запрашиваемой секции
  - полная реализация дорабатывается

Поддержка физического бэкапа. PITR

- Физический бэкап снимается с каждого узла кластера
- Восстановление до глобальной точки консистентности
- В Greenplum точки консистентности определяются глобальными именованными точками восстановления - вызов *gp_restore_point()*
  - Короткая блокировка на логирование статуса 2PC на координаторе во время создания именованной точки на всех узлах
  - Некоторый worker периодически создает глобальные именованные точки восстановления
- Теоретически можно рассчитывать точки консистентности в WAL координатора и сегментов
  - чтобы иметь управляемый более гранулированный recovery
- Можно вести резервный (standby) кластер - DR (disaster recovery) решение

Кластерные возможности. Сводная таблица

	Greenplum	CitusDB	TimescaleDB	OpenGauss
Распределенное выполнение запросов	Полная поддержка	Сегменты → координатор Частичная поддержка перераспределения данных внутри кластера	Сегменты → координатор	GaussDB: полная поддержка
Консистентное чтение с кластера	Snapshot на базе глобального счетчика транзакций	Нет	Нет	GaussDB: через внешний GTM
Атомарность глобальной модификации	2PC с логированием в WAL координатора	2PC с сохранением состояния во внутреннюю таблицу	2PC с ведением лога состояний во внутренней таблице	GaussDB: через внешний GTM
High Availability	Встроенный автофейловер для сегментов. Ручной - для координатора	Полная поддержка через patroni. Patroni 3.0 поддерживает из коробки	Полная поддержка через patroni. Регулируемый фактор репликации для секций	GaussDB: полная поддержка: встроенные paxos обеспечивает автофейловер и кворумный коммит
Физический бэкап. PITR	Через периодическое создание глобальных именованных точек восстановления	Через периодическое создание глобальных именованных точек восстановления	Через периодическое создание глобальных именованных точек восстановления	GaussDB: через периодическое создание глобальных именованных точек восстановления

Колоночное хранение данных

- Формат хранения данных
- Компрессия данных
- Поддержка Update / Delete операций
  - Организация MVCC. Цена vacuum
- Векторизованный движок исполнения
- “Zone maps”

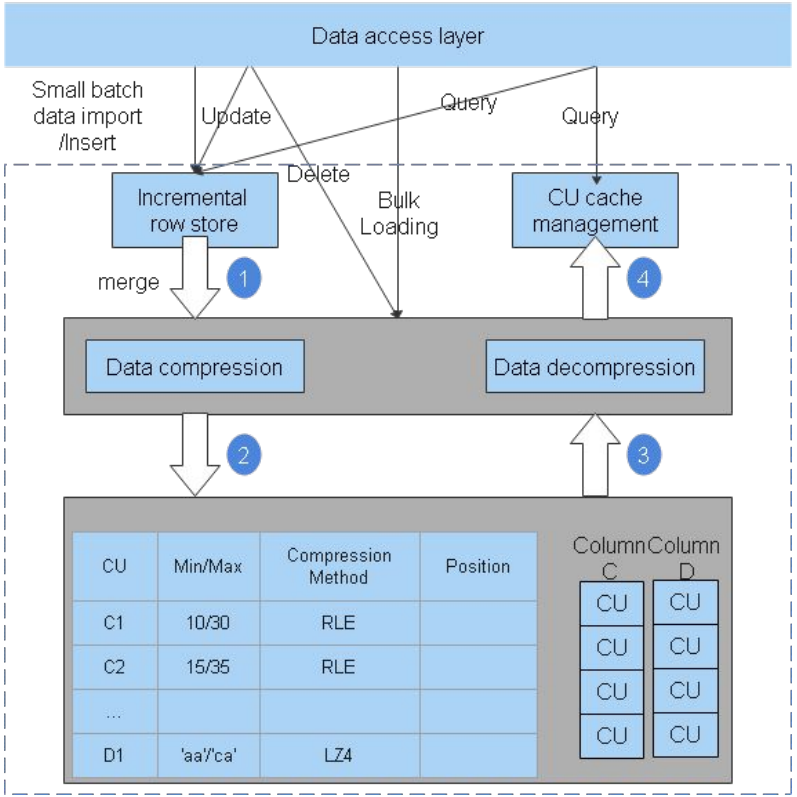
Колоночный формат хранения. Greenplum

- Построен на баз “append-only” таблиц
  - Таблица состоит из 127 сегментов, запись в которые не блокируется друг относительно друга
  - Строка идентифицируется номером в сегментном файле
- Вспомогательные heap таблицы для метаданных
  - *pg_aoseg* - хранение физических свойств сегментных файлов
  - *pg_aovisimap* - битовая карта видимости строк
  - *pg_aoblkdir* - карта разбиения сегментов на блоки для быстрого поиска по номеру строки
- В колоночной таблице каждой колонке соответствует свой набор из 127 сегментов

Колоночный формат хранения. CitusDB

- Колонки делятся на stripe`ы - единицы загрузки данных (по умолчанию, 150к строк)
  - рекомендуется вставлять данные большими партиями
- Stripe`ы делятся на chunk`и - единый сжатый набор значений (по умолчанию, 10к строк)
  - размеры chunk`ов одинаковы внутри stripe`а
  - для быстрого доступа к значению по номеру строки
- Метаданные для stripe`ов и chunk`ов хранятся в таблице метаданных внутри схемы *columnar_internal*
  - включая min и max значения для chunk`а

Колоночный формат хранения. OpenGauss



- Колоночный файл поделен на блоки (Column Units, CU)
- Для каждого блока задается:
  - свой алгоритм сжатия (выбирается автоматически по типу колонки)
  - минимальное и максимальное значение
  - номер первой строки для быстрого поиска по номеру строки внутри колонки
- Данные небольшими партиями вставляются в “incremental row store”
  - Фоновый поток “*delta table”* мигрирует данные из incremental row store в колоночное хранилище
- Большими партиями данные вставляются сразу в колоночное хранилище
- Select / Update / Delete выполняется в двух хранилищах

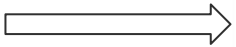
Колоночный формат хранения. TimescaleDB

- Колоночный формат имитируется внутри строчного heap хранилища
  - в одну строку объединяют в массив колонки нескольких изначальных строк
  - *“orderby”* (как правило, *time*) колонка задает порядок значений в общей строке
    - в строке определяются min и max значения “*orderby*” колонки
  - *“segmentby”* (как правило, *space*) колонка группирует значения по этой колонке
    - в строке определяется значение *“segmentby”* колонки и на нее строится индекс
  - сгруппированные значения колонок (как правило, *time* и метрики) собираются в массивы и сжимаются
    - если сжатый размер массива превышает допустимые пределы хранения в обычной heap странице, его хранение переносится в TOAST
    - TOAST хранит данные поколоночно
  - в идеальном случае, поколоночное сжатое хранение метрик и time колонки с heap-строкой, хранящей метаданные в виде значения *space* колонки и min и max значения *time* колонки для блока сжатых данных

Колоночный формат хранения. TimescaleDB

Timestamp	Device ID	Status Code	Temperature
12:00:01	A	0	70.11
12:00:01	B	0	69.70
12:00:02	A	0	70.12
12:00:02	B	0	69.69
12:00:03	A	0	70.14
12:00:03	B	4	69.70

compress = true



Timestamp	Device ID	Status Code	Temperature
[12:00:01, 12:00:01, 12:00:02, 12:00:02, 12:00:03, 12:00:03]	[A, B, A, B, A, B]	[0, 0, 0, 0, 0, 4]	[70.11, 69.70, 70.12, 69.69, 70.14, 69.70]

compress_segmentby = ‘Device ID’
compress_orderby = ‘Timestamp’

Device ID	Timestamp	Status Code	Temperature	Min Timestamp	Max Timestamp
A	[12:00:01, 12:00:02, 12:00:03]	[0, 0, 0]	[70.11, 70.12, 70.14]	12:00:01	12:00:03
B	[12:00:01, 12:00:02, 12:00:03]	[0, 0, 0]	[70.11, 70.12, 70.14]	12:00:01	12:00:03

Колоночный формат хранения. TimescaleDB

```
-- Таблица test секционирована по space колонке "location" и time колонке "time"
-- location - "segmentby" атрибут, time - "orderby" атрибут
-- Для пользователя чтение из сжатой секции прозрачно
explain (costs off)
select avg(metric_value) from test
where location = 'abc' and
time between timestamp '2023-04-03 13:01:17' and timestamp '2023-04-03 13:01:18';
```

```
                                   QUERY PLAN
--------------------------------------------------------------------------------
 Aggregate
   ->  Custom Scan (ChunkAppend) on test
         Chunks excluded during startup: 7
         ->  Custom Scan (DecompressChunk) on _hyper_6_22_chunk
               Filter: (("time" >= '2023-04-03 13:01:17'::timestamp without time zone) AND ...
               ->  Index Scan using compress_hyper_8_26_chunk__compressed_hypertable_8_location__ts on compress_hyper_8_26_chunk
                     Index Cond: (location = 'abc'::text)
                     Filter: ((_ts_meta_max_1 >= '2023-04-03 13:01:17'::timestamp without time zone) AND ...
(8 rows)
```

кастомный метод доступа к “пустой” секции

фильтрация блоков по orderby атрибуту

обращение к индексу реального хранилища

Поддержка Update / Delete операций. MVCC. Greenplum

- Удаление помечает соответствующий бит по номеру строки в pg_aovisimap
- Изменение = удаление + создание (добавление в конец)
- Цепочка версий не поддерживается
  - невозможно реализовать обновление / удаление обновленной версии строки на read committed уровне изоляции - процедура EPQ (EvalPlanQual)

Цена vacuum. Greenplum

- Строка считается мертвой, если удалена && нет конкурентной repeatable read / serializable транзакции на координаторе
  - Можно физически удалить видимую строку - требуются “прямые” руки
- Проверка видимости индексного указателя (ItemPointer) требует загрузки фрагмента карты видимости, соответствующего номеру рассматриваемой строки
  - Индекс в 12 Гб вакуумится ~3 часа [1]
  - Улучшения в этой области [2]

1. https://groups.google.com/a/greenplum.org/g/gpdb-dev/c/CHNFp_yeX1M/m/nof14jCkAAAJ
2. https://github.com/greenplum-db/gpdb/pull/13255

Векторизованный движок исполнения

- Поблочная (векторная) обработка значений с колонок
- Возможные оптимизации [1,2]:
  - Отложенное формирование строк в дереве плана (late materialization)
  - Отложенная декомпрессия
  - Invisible join foreign key with primary key
    - Semi-join → predicate expression (hash lookup)
  - Применение векторных CPU операций и других аппаратных оптимизаций

1. https://ieeexplore.ieee.org/document/8187108
2. https://www.cs.umd.edu/~abadi/papers/abadiphd.pdf

Векторизованный движок исполнения. OpenGauss

```
-- Create row and column based tables with
-- fist column (a) distributed evenly for aggregation
-- second (b) - distributed evenly for applying range predicate
-- third (c) - ordered for applying non-range predicate
-- fourth (d) - distributed evenly within range [0, 9] for grouping
create table row_test as
select (random()*10000)::int as a, (random()*100000)::int as b, i as c, (random()*10)::int as d
from generate_series(1, 10*1000*1000) i;
create table col_test with (orientation=column) as
select (random()*10000)::int as a, (random()*100000)::int as b, i as c, (random()*10)::int as d
from generate_series(1, 10*1000*1000) i;

-- Compare storage sizes
select pg_size_pretty(pg_table_size('row_test'));
 pg_size_pretty
----------------
 422 MB

select pg_size_pretty(pg_table_size('col_test'));
 pg_size_pretty
----------------
 78 MB
```

Векторизованный движок исполнения. OpenGauss

```
explain (costs off, analyze, buffers)
select avg(a) from row_test
where b between 1000 and 9000 and c%2 = 0
group by d;
                                QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate (actual time=5633.408..5633.428 rows=11 loops=1)
   Output: avg(a), d
   Group By Key: row_test.d
   (Buffers: shared hit=46421 read=7634)
   ->  Seq Scan on public.row_test (actual time=0.079..5248.882 rows=400211
loops=1)
         Output: d, a
         Filter: ((row_test.b >= 1000) AND (row_test.b <= 9000) AND ((row_test.c % 2)
= 0))
         Rows Removed by Filter: 9599789
         (Buffers: shared hit=46421 read=7634)
Total runtime: 5355.814 ms
```

```
explain (costs off, analyze, buffers)
select avg(a) from col_test
where b between 1000 and 9000 and c%2 = 0
group by d;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Row Adapter (actual time=967.146..967.151 rows=11 loops=1)
   Output: (avg(a)), d
   ->  Vector Sonic Hash Aggregate (actual time=967.112..967.114 rows=11 loops=1)
         Output: avg(a), d
         Group By Key: col_test.d
         (Buffers: shared hit=559)
         ->  CStore Scan on public.col_test (actual time=1.714..894.895 rows=400098
loops=1)
               Output: d, a
               Filter: ((col_test.b >= 1000) AND (col_test.b <= 9000) AND ((col_test.c % 2) =
0))
               Rows Removed by Filter: 9599902
               (Buffers: shared hit=559)
Total runtime: 933.497 ms
```

Примечание: запросы запускались на максимально прогретой базе

“Zone maps”

- Термин из Oracle [1]
- Хранение предвычисленных агрегатов для блоков в колонках
- Как минимум, min и max значений для каждого блока
  - Полезно для предикатов на сравнение
  - Можно пропустить блок, если предикат не укладывается в min, max диапазон
  - Аналогично BRIN-индексам

1. https://docs.oracle.com/database/121/DWHSG/zone_maps.htm#DWHSG8938

“Zone maps”. GaussDB

```
-- Range predicate on non-ordered column
explain (costs off, analyze, buffers, timing off, verbose)
select avg(a) from col_test
where b between 1000 and 9000
group by d;
                          QUERY PLAN
-------------------------------------------------------------
 Row Adapter (rows=11 loops=1)
   Output: (avg(a)), d
   ->  Vector Sonic Hash Aggregate (rows=11 loops=1)
       Output: avg(a), d
       Group By Key: col_test.d
       (Buffers: shared hit=545)
       ->  CStore Scan on public.col_test (rows=800137
loops=1)
             Output: d, a
             Filter: ((col_test.b >= 1000) AND
                      (col_test.b <= 9000))
             Rows Removed by Filter: 9199863
             (Buffers: shared hit=545)
 Total runtime: 618.891 ms
```

```
-- Range predicate on ordered column
explain (costs off, analyze, buffers, timing off, verbose)
select avg(a) from col_test
where c between 1000 and 9000
group by d;
                          QUERY PLAN
-------------------------------------------------------------
 Row Adapter (rows=11 loops=1)
   Output: (avg(a)), d
   ->  Vector Sonic Hash Aggregate (rows=11 loops=1)
       Output: avg(a), d
       Group By Key: col_test.d
       (Buffers: shared hit=544)
       ->  CStore Scan on public.col_test (rows=8001
loops=1)
             Output: d, a
             Filter: ((col_test.c >= 1000) AND
                      (col_test.c <= 9000))
             Rows Removed by Filter: 9991999
             (Buffers: shared hit=544)
 Total runtime: 7.980 ms
```

ускорение на два порядка

“Zone maps”. CitusDB. Chunk Group Filtering

```
explain (costs off, analyze, timing off, buffers, verbose)
select avg(a) from col_test
where b between 1000 and 9000
group by d;
                                   QUERY PLAN
-----------------------------------------------------------------------
 HashAggregate (actual rows=11 loops=1)
   Output: avg(a), d
   Group Key: col_test.d
   Batches: 1  Memory Usage: 40kB
   Buffers: shared hit=16650
   ->  Custom Scan (ColumnarScan) on public.col_test
               (actual rows=8001618 loops=1)
         Output: a, d
         Filter: ((col_test.b >= 1000) AND (col_test.b <= 9000))
         Rows Removed by Filter: 1998382
         Columnar Projected Columns: a, b, d
         Columnar Chunk Group Filters: ((b >= 1000) AND (b <= 9000))
         Columnar Chunk Groups Removed by Filter: 0
         Buffers: shared hit=16650
 Planning:
   Buffers: shared hit=51
 Planning Time: 0.424 ms
 Execution Time: 1285.609 ms
```

ускорение более чем на два порядка

```
explain (costs off, analyze, timing off, buffers, verbose)
select avg(a) from col_test
where c between 1000 and 9000
group by d;
                                   QUERY PLAN
-----------------------------------------------------------------------
 HashAggregate (actual rows=11 loops=1)
   Output: avg(a), d
   Group Key: col_test.d
   Batches: 1  Memory Usage: 40kB
   Buffers: shared hit=3772
   ->  Custom Scan (ColumnarScan) on public.col_test
               (actual rows=8001 loops=1)
         Output: a, d
         Filter: ((col_test.c >= 1000) AND (col_test.c <= 9000))
         Rows Removed by Filter: 1999
         Columnar Projected Columns: a, c, d
         Columnar Chunk Group Filters: ((c >= 1000) AND (c <= 9000))
         Columnar Chunk Groups Removed by Filter: 999
         Buffers: shared hit=3772
 Planning:
   Buffers: shared hit=51
 Planning Time: 0.285 ms
 Execution Time: 6.276 ms
```

Колоночное хранение. Сводная таблица

	Greenplum	CitusDB	TimescaleDB	OpenGauss
Расположение данных	На основе “append-only” табличного движка	Колонки делятся на сжимаемые по отдельности блоки	Имитируется группировкой значений с колонок в одну строку	Incremental row store и колоночные дата файлы
Компрессия	Ручной выбор алгоритма для каждой колонки и таблицы в целом. Для строк ZLIB, ZSTD. Для колонок RLE, ZLIB, ZSTD	Ручной выбор для всей таблицы. PGLZ, ZSTD, ZSTD, LZ4, LZ4HC	Автоматический выбор алгоритма в соответствии с типом колонки. DeltaDelta, Gorilla, DICTIONARY, “Array” (TOAST) [1,2]	Ручной выбор. Автоматическое выбор алгоритма в соответствии с типом колонки. RLE, DELTA, BITPACK/BYTEPACK, ZLIB, LOCAL DICRIONARY
Update / Delete	Поддерживается	Нет	Нет	Поддерживается
Цена вакуума	Высокая при наличии вторичного индекса	—	—	???
Векторизованный движок исполнения	Нет	Нет	Нет	Есть
“Zone maps”	Нет	Min/max	min/max по time колонке, группировка по space колонке	Min/max

1. https://www.timescale.com/blog/time-series-compression-algorithms-explained/
2. https://github.com/timescale/timescaledb/blob/main/tsl/src/compression/README.md#base-algorithms

Специфичные фичи аналитических СУБД

- Управление ресурсами исполнения запроса aka ресурсные группы
- …

Управление ресурсами исполнения

- PostgreSQL из коробки предлагает достаточно “грубые” ограничения на потребление ресурсов
  - системные: *max_worker_processes*, *max_parallel_workers*
  - на запрос: *max_parallel_maintenance_workers*, *max_parallel_workers_per_gather*
  - на узел плана: *work_mem*
- Более гранулированные ограничения реализуют системными средствами ОС (*cgroups*)
  - ограничения по локальной динамической памяти достигают через внутренний memory detector

Управление ресурсами исполнения. Greenplum

- Ресурсные группы включают:
  - Ограничение по CPU через cgroup
    - доля используемого CPU time
    - выделенное кол-во ядер на запросы
  - Ограничение по потребляемой локальной динамической памяти через внутренний memory detector
  - Ограничение по потоку IO (WIP) [1]
- Задаются для конкретных ролей
- Возможна динамическая миграция сессии из одной ресурсной группы в другую во время исполнения запроса

1. https://groups.google.com/a/greenplum.org/g/gpdb-dev/c/l-5RpOz0fP8/m/dYRTTj8kAgAJ

Специфичные фичи аналитических СУБД. Сводная таблица

	Greenplum	CitusDB	TimescaleDB	OpenGauss
Управление ресурсами исполнения	Ресурсные группы. Ограничения по CPU и локальной динамической памяти	Базовые настройки постгреса	Базовые настройки постгреса	Workload manager
…				

Бенчмарки

Продолжение следует…

Спасибо за внимание!
Вопросы, критика, пожелания…

Максим Милютин milyutinma@gmail.com