PostgreSQL Advanced

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Массивно параллельные кластера PostgreSQL

Алексей Железной

Senior Data Engineer *в Wildberries*
Магистратура - ФКН ВШЭ

Руководитель курсов ***DWH Analyst, ClickHouse для инженеров и архитекторов БД*** *в OTUS*

Преподаватель курсов ***Data Engineer, DWH Analyst, PostgreSQL*** *и пр. в OTUS*

LinkedIn

Правила вебинара

Активно
участвуем

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

- Архитектура, особенности работы Greenplum
- Архитектура, особенности работы ArenadataDB
- Архитектура, особенности работы YugabyteDB
- Практика
- Рефлексия

Цели вебинара

1. Иметь представление MPP СУБД Greenplum
2. Изучить варианты установки

Смысл

Зачем вам это уметь

1. Чтобы выбрать оптимальную инфраструктуру под ваш проект

Greenplum

Greenplum

Greenplum – open-source продукт, массивно-параллельная реляционная СУБД для хранилищ данных

Особенности:

- отличная горизонтальная масштабируемость
- столбцовое хранение данных на основе PostgreSQL
- мощный оптимизатор запросов
- надежность
- высокая скорость обработки SQL-запросов над большими объемами данных
- широко применяется для аналитики Big Data в промышленных масштабах

Greenplum

История:

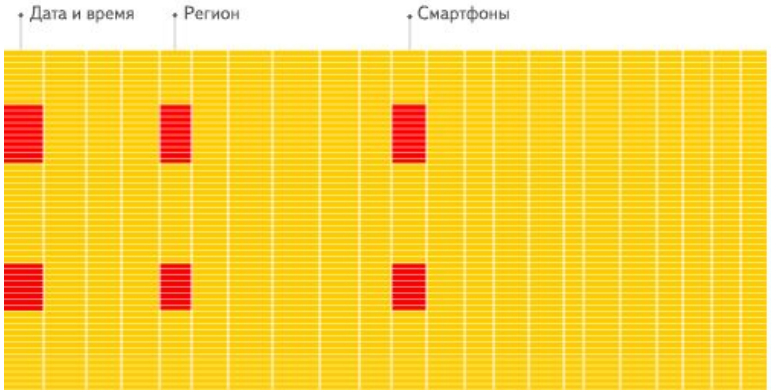
- **2005 год** – первый выпуск технологии одноименной фирмой в Калифорнии (США);
- **2010 год** – корпорация EMC поглотила компанию Greenplum, продолжив работу над проектом;
- **2011 год** – корпорация EMC выпустила для всеобщего пользования бесплатную версию Greenplum Community Edition;
- **2012 год** – корпорация Pivotal приобрела продукт EMC Greenplum Community Edition, продолжая далее развивать его под своим брендом;
- **2015 год** – компания Pivotal опубликовала исходный код СУБД Greenplum под свободной лицензией Apache;
- **2018 год** – интеграция с отечественной платформой визуализации и анализа данных Luxms BI;
- **2018 год** – российская компания «Аренадата Софтвер», разработчик первого отечественного дистрибутива Apache Hadoop, выпустила собственную MPP-СУБД Arenadata DB на основе Greenplum, адаптировав ее для корпоративного использования.
- **2020 год** – корпорация VMware приобрела компанию Pivotal, которая была вендором Greenplum (GP) с 2012 года. С этого момента open-source MPP-СУБД коммерциализируется под торговой маркой VMware Tanzu Greenplum

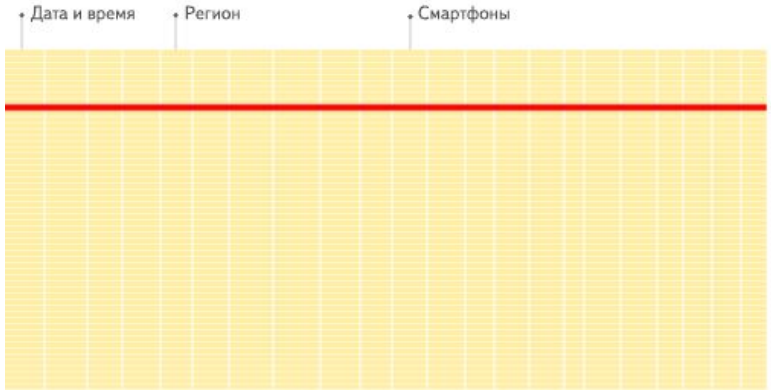
Столбцовые СУБД

Row	WatchID	JavaEnable	Title	GoodEvent	EventTime
#0	89354350662	1	Investor Relations	1	2016-05-18 05:19:20
#1	90329509958	0	Contact us	1	2016-05-18 08:10:20
#2	89953706054	1	Mission	1	
#N	...	...	...	...	

Row:	#0	#1	#2	#N
WatchID:	89354350662	90329509958	89953706054	...
JavaEnable:	1	0	1	...
Title:	Investor Relations	Contact us	Mission	...
GoodEvent:	1	1	1	...
EventTime:	2016-05-18 05:19:20	2016-05-18 08:10:20	2016-05-18 07:38:00	...

Столбцовые СУБД





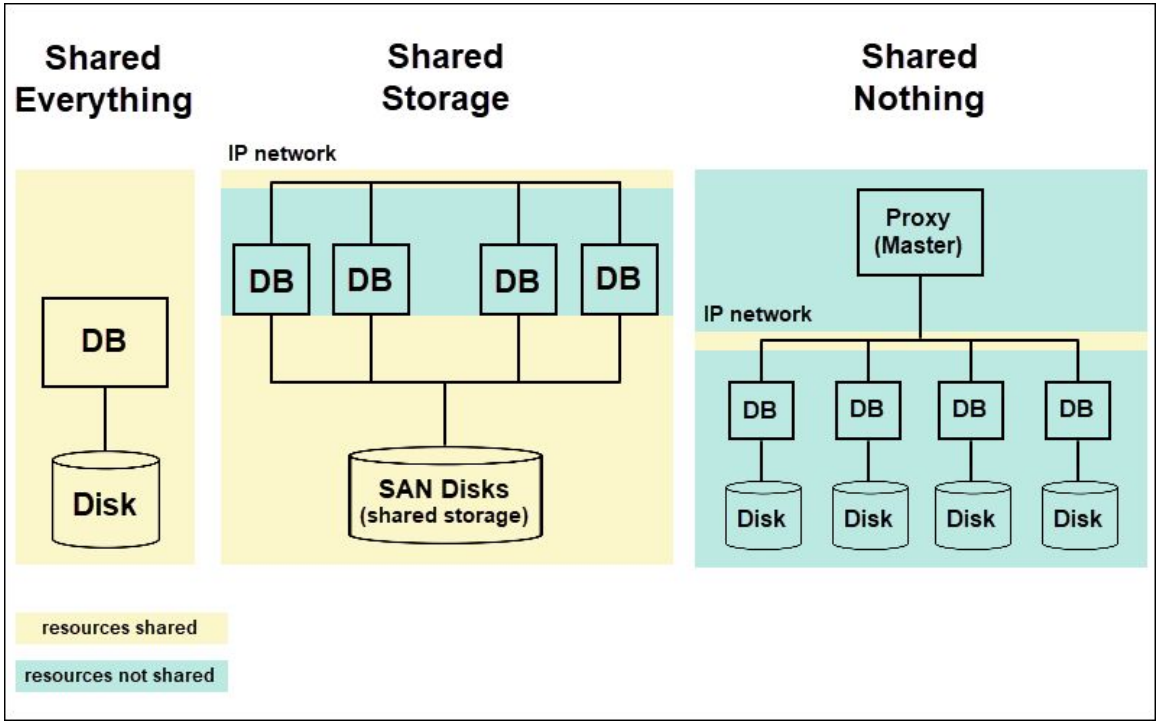
Greenplum

Плюсы:

- Блок меньше размером
- Проще сжимается
- Вертикальное секционирование

Бенчмарки

Свойства кластера



Greenplum - Архитектура

- несколько взаимосвязанных экземпляров базы данных PostgreSQL
- объединенные в кластер по принципу массивно-параллельной архитектуры (Massive Parallel Processing, MPP)
- без разделения ресурсов (Shared Nothing)
- каждый узел кластера, взаимодействующий с другими для выполнения вычислительных операций, имеет собственную память, операционную систему и жесткие диски.

Greenplum - Архитектура

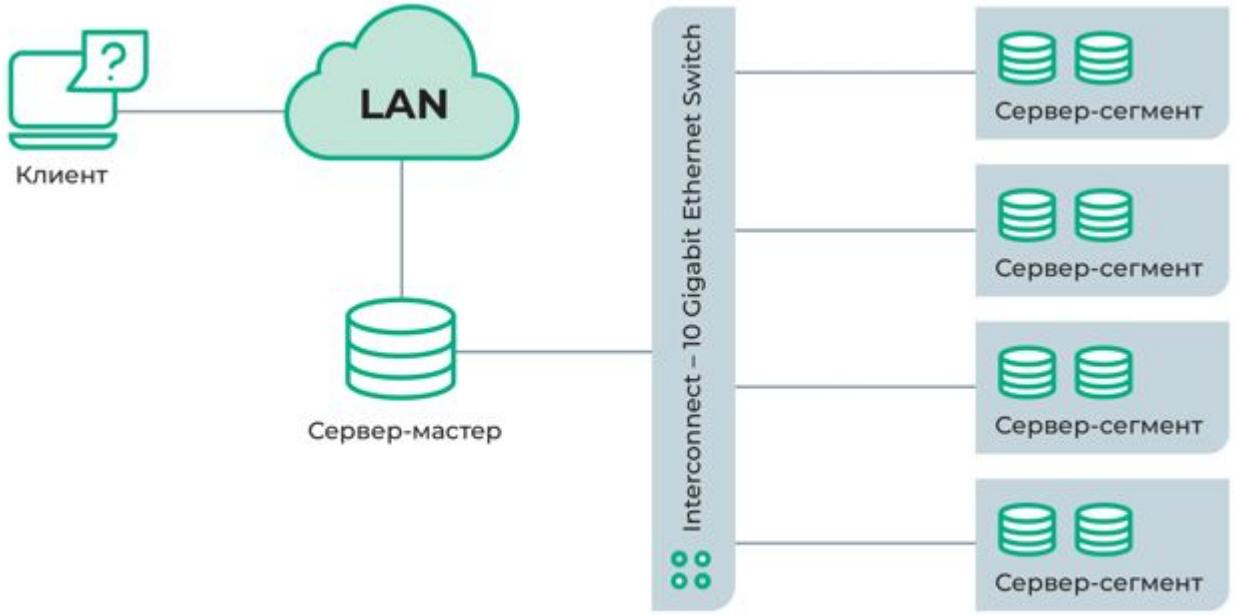
Для повышения надежности к типовой топологии master-slave добавлен резервный главный сервер. Состав кластера Greenplum

- **Мастер-сервер(Master host)**, где развернут главный инстанс PostgreSQL (Master instance). Это точка входа в Greenplum, куда подключаются клиенты, отправляя SQL-запросы. Мастер координирует свою работу с сегментами – другими экземплярами базы данных PostgreSQL. Мастер распределяет нагрузку между сегментами, но сам не содержит никаких пользовательских данных – они хранятся только на сегментах.
- **Резервный мастер(Secondary master instance)** — инстанс PostgreSQL, включаемый вручную при отказе основного мастера.

Greenplum - Архитектура

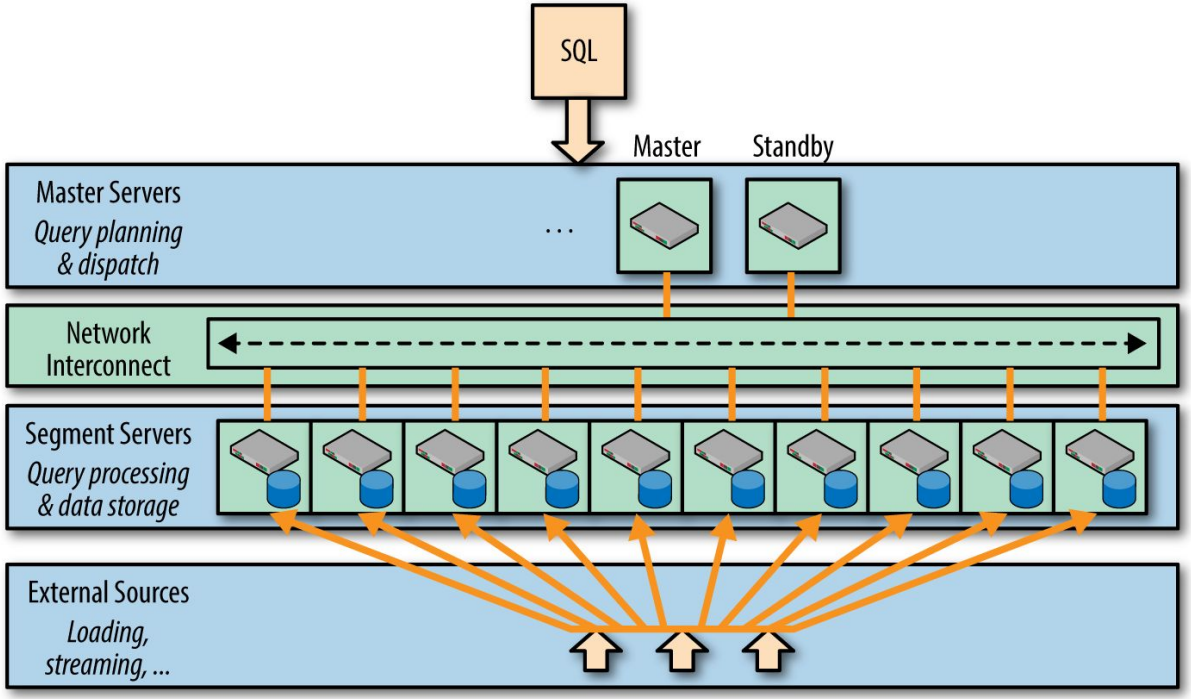
- **Сервер-сегмент (Segment host)**, где хранятся и обрабатываются данные. На одном хост-сегменте содержится 2-8 сегментов Greenplum – независимых экземпляров PostgreSQL с частью данных. Сегменты Greenplum бывают основные (primary) и зеркальные (mirror). Primary-сегмент обрабатывает локальные данные и отдает результаты мастеру. Каждому primary-сегменту соответствует свое зеркало (Mirror segment instance), которое автоматически включается в работу при отказе primary.
- **Интерконнект (interconnect)** – быстрое обособленное сетевое соединение для связи между отдельными экземплярами PostgreSQL.

Greenplum - Архитектура



Pic. 3

Greenplum - Архитектура



Greenplum - Архитектура

- **Несколько разных interconnect-сетей повышают пропускную способность канала взаимодействия сегментов** и обеспечивают отказоустойчивость кластера, т.к. при отказе одной из сетей весь трафик перераспределяется между оставшимися;
- **Повысить производительность кластера можно за счет идентичной настройки хост-сегментов.** Это позволит равномерно распределить данные и рабочие нагрузки по большему числу сегментов с одинаковыми мощностями для распараллеливания задач и их одновременного выполнения.

Greenplum - Архитектура

- **Быстрота работы узла зависит от соотношения числа процессорных ядер к числу данных, расположенных на этом хосте.** Любой запрос на одном сегменте не может занимать более 1-го процессорного ядра.
- **Greenplum работает со скоростью самого медленного сегмента,** поэтому несбалансированное распределение данных (в одной таблице или во всей базе) по сегментам снижает общую производительность кластера.

Выбор правильного ключа для секционирования/шардирования

- **В Greenplum каждая таблица представлена в виде (N+1) таблиц на всех сегментах кластера**, где N – число сегментов + 1 таблица на мастере, где нет пользовательских данных. На каждом сегменте хранится 1/N строк таблицы.
- **Логика разбиения таблицы на сегменты задаётся ключом (полем) дистрибуции, на основе которого любую строку можно отнести к одному из сегментов.** Поскольку именно ключ дистрибуции задает распределение данных по сегментам кластера, выбирать это поле нужно по принципу равномерного распределения значений в нем

Выбор правильного ключа для секционирования/шардирования

Пример неудачного ключа шардирования.

- Есть коллекция переводов — translations. В ней есть поле language, которое хранит язык. Например, коллекция поддерживает 100 языков и мы шардируем по языку.
- Это плохо — cardinality количество возможных значений всего 100 штук, что мало. Но это не самое плохое — может быть, для этих целей cardinality достаточно.
- Хуже, что как только мы пошардировали по языку, мы тут же узнаем, что у нас англоязычных пользователей в 10 раз больше, чем остальных. На несчастный шард, на котором находится английский язык, приходит в десять раз больше запросов, чем на все остальные вместе взятые.

Выбор правильного ключа для секционирования/шардирования

- Поэтому надо учитывать, что иногда шард-ключ естественным образом тяготеет к неравномерному распределению нагрузки.

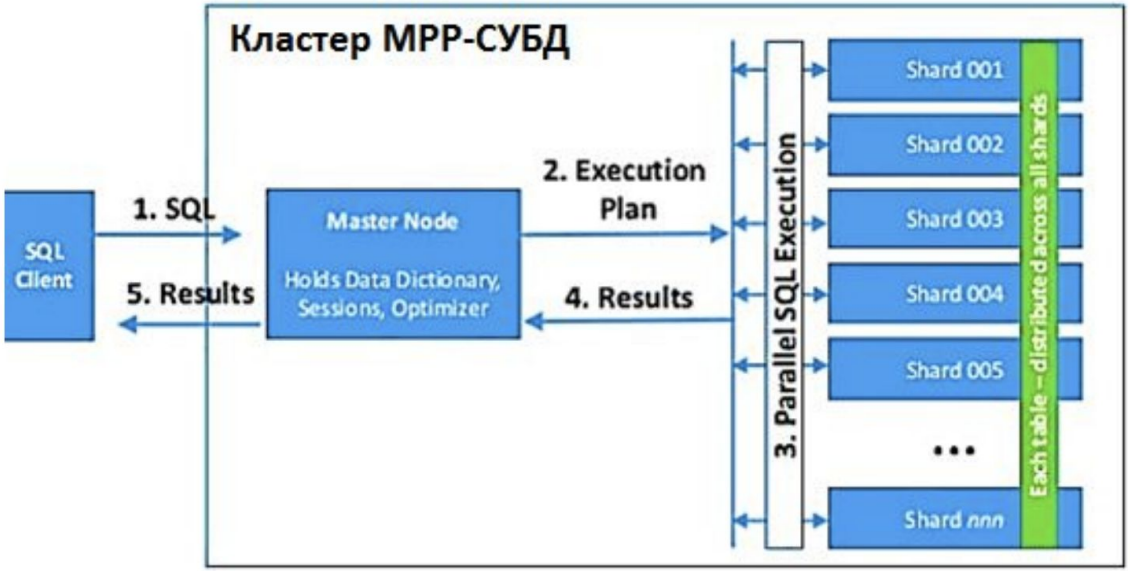
Оптимальным считается хэш от ключа

Нюансы

- Начиная с 6-ой версии (конец 2019 года), появился новый инструмент оптимизации схемы хранения данных — **реплицированные таблицы**.
- Они полностью дублируются на всех сегментах кластера, поэтому их JOIN-соединение будет выполняться локально, без перераспределения данных. ***Это особенно важно для хранения объёмных справочников***

- Также в новой версии появился алгоритм **consistent hashing**, который позволяет при добавлении в кластер новых узлов перераспределять только часть блоков, ускоряя фоновое перераспределение таблиц.

Прохождение запроса



Применение Greenplum

- Системы предиктивной аналитики и регулярной отчетности по большими объемами данных;
- Построение озер (Data Lake) и корпоративных хранилищ данных (КХД);
- Разработка аналитических моделей по множеству разнообразных данных, например, для прогнозирования оттока клиентов (Churn Rate).

Варианты хранилищ

- **Хранилище кучи (heap storage)**, которое обычно использует PostgreSQL для всех таблиц базы данных. Этот способ подходит для таблиц и разделов, которые будут получать конкурентные запросы: интерактивные UPDATE и DELETE, а также одиночные INSERT.
- **Хранилище, оптимизированное для добавления (AO, append-optimized storage)** таблиц и разделов, которые не часто обновляются после начальной загрузки, а последующие вставки выполняются только в пакетных операциях. Следует избегать выполнения одноэлементных операций INSERT, UPDATE или DELETE в AO-таблицах. Допустимы конкурентные пакетные операции INSERT, но не пакетные UPDATE или DELETE. Эта модель предназначена для больших таблиц, которые загружаются один раз, редко обновляются и часто запрашиваются для аналитической обработки запросов.

Варианты хранилищ

- **Хранилище, ориентированное на строки**, с традиционным способом хранения кортежей базы данных. Столбцы хранятся на диске непрерывно, так что вся строка может быть прочитана с диска за один ввод-вывод.
- **Хранилище, ориентированное на столбцы,** сохраняющее все значения каждого столбца на диске в отдельном файле. Если таблица разбита на разделы, для каждого столбца и раздела создается отдельный файл. Когда запрос обращается только к небольшому количеству столбцов из множества колонок такой таблицы, затраты на ввод-вывод данных существенно снижаются по сравнению со строковым хранилищем, т.к. ненужные столбцы не считываются с диска.

Best Practice

- Сперва спроектируйте модель только для вставки, отсекая ежедневный раздел перед загрузкой;
- Для разных разделов в больших таблицах фактов наиболее эффективно **использовать различные варианты** хранения, т.к. некоторые разделы могут быть ориентированы на строки, а другие – наоборот, на столбцы
- Параметры хранилища устанавливаются на уровне раздела
- **В колоночном хранилище, любой столбец является отдельным файлом в каждом сегменте базы данных Greenplum**. Поэтому в таких таблицах рекомендуется хранить «горячие» данные, обращения к которым выполняются часто.

Best Practice

- **Строковые хранилища подходят для «холодных» данных**, которые требуется считывать достаточно редко, и для рабочих нагрузок транзакционного типа с итеративными транзакциями с обновлениями и частыми вставками.
- **Используйте строковое хранилище для широких выборок**, когда в запросе используется много столбцов одной строки, например, в списке SELECT или условии WHERE. Также строковое хранилище подходит для общих или смешанных рабочих нагрузок, поскольку оно предлагает наилучшее сочетание гибкости и производительности.
- **Колоночное хранилище оптимизировано для операций чтения, но не для записи**: таблицы, ориентированные на столбцы, оптимальны для небольшой выборки столбцов из больших таблиц.

Best Practice

- В колоночных таблицах набор значений одного и того же типа данных может храниться в меньшем пространстве, занимая меньше дискового пространства, чем строковые таблицы. Также **колоночные таблицы лучше сжимаются, чем строковые.**
- **Колоночные таблицы отлично подходят для аналитических рабочих нагрузок**, когда выборка ограничена или агрегаты данных вычисляются по небольшому количеству столбцов. Также эта модель рекомендуется для таблиц с отдельными столбцами, которые регулярно обновляются без изменения других столбцов в строке.
- Для оптимальной производительности следует располагать столбцы таблицы так, чтобы добиться выравнивания типов данных на уровне байтов

Best Practice по типам данных

- **Выбирать типы данных, которые используют наименьшее пространство для хранения, чтобы увеличить емкость базы и ускорить выполнение запросов.** Например, TEXT или VARCHAR вместо CHAR. На первый взгляд, между символьными типами данных нет различий в производительности, но TEXT или VARCHAR требуют меньше пространства для хранения. Аналогично с числовыми типами данных: использование BIGINT для данных, которые помещаются в INT или SMALLINT, тратит впустую место в хранилище.

Best Practice по типам данных

- **Размещать столбцы в порядке от наибольшего к наименьшему** – от столбцов с ключами распределения и разделения к фиксированным числовым и переменным типам. Например, BIGINT и TIMESTAMP перед INT и DATE, а только потом TEXT, VARCHAR или NUMERIC(x, y). Например, сперва идут 8-байтовые типы (BIGINT, TIMESTAMP), затем 4-байтовые типы (INT, DATE), далее 2-байтовые типы, после (SMALLINT), а переменный тип данных (VARCHAR) – в последнюю очередь.
- Следует **сжимать большие таблицы**, чтобы повысить производительность ввода-вывода и освободить место в кластере

Варианты интеграции

Недостатки

- **Высокие требования к ресурсам** ЦП, памяти и жестким дискам, а также к сетевой инфраструктуре
- **Низкая производительность при большом объеме простых запросов**, выполняющих одну операцию, т.к. каждая транзакция на мастере порождает множество зеркальных транзакций на сегментах
- **Неоптимальное распределение сегментов**, что может негативно отразиться на производительности кластера при его расширении.

Варианты установки

- Docker (https://hub.docker.com/r/datagrip/greenplum)
- Kubernetes
- Ansible
- Cloud
- Managed instance

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

YugabyteDB

YugabyteDB

Это высокопроизводительная распределенная база данных SQL с открытым исходным кодом, построенная на масштабируемой и отказоустойчивой конструкции, созданной по мотивам Google Spanner. SQL API (YSQL) Yugabyte совместим с PostgreSQL.

Сравнение кластеров - https://docs.yugabyte.com/preview/comparisons/

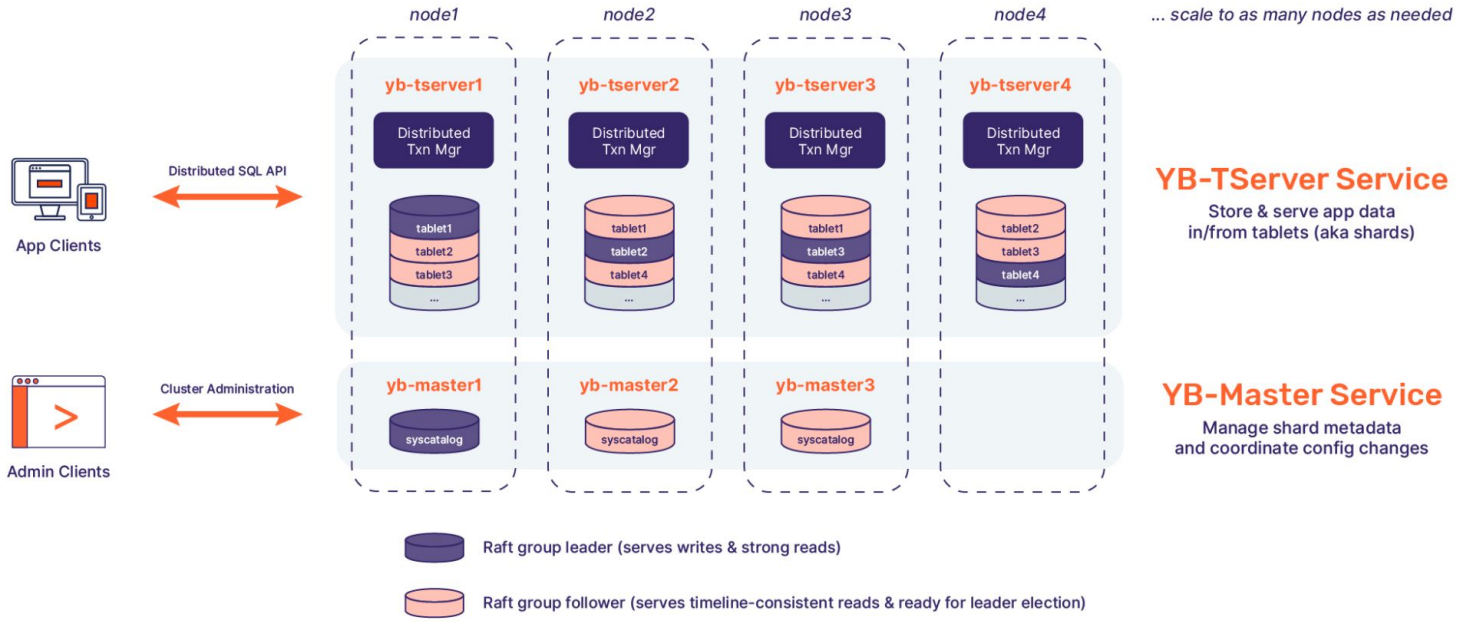
Особенности

- Отказоустойчивость (Raft consensus - http://thesecretlivesofdata.com/raft/) – автоматический режим failover (диск, нода, зона, регион).
- Горизонтальная масштабируемость – добавление реплики
- Распределенные транзакции (архитектура Google Spanner)
- Геораспределенный кластер (мультизональный, мультирегиональный, мультиоблачный)
- Yugabyte SQL (YSQL) – синтаксис PostgreSQL (https://github.com/yugabyte/yugabyte-db/blob/master/architecture/YSQL-Features-Supported.md)
- Yugabyte Cloud QL(YCQL - Apache Cassandra QL)
- 100% open source - Apache 2.0 license (https://github.com/yugabyte/yugabyte-db)

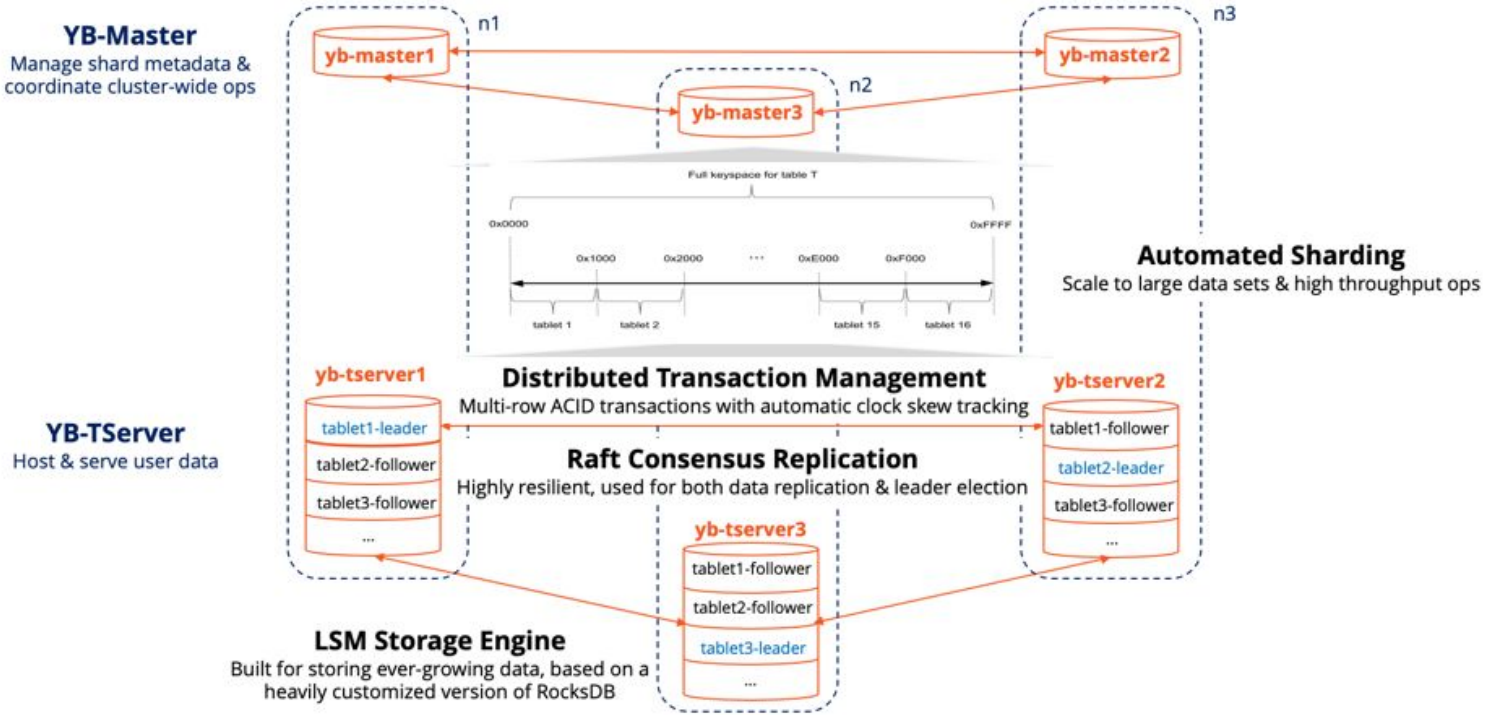
Поддержка ЯП

Язык	ORM	YSQL - драйвер	YCQL -драйвер
Java	Spring/Hibernate	PostgreSQL JDBC	cassandra-driver-core-yb
Go	Gorm	pq	gocql
NodeJS	Sequelize	pg	cassandra-driver
Python	SQLAlchemy	psycopg2	yb-cassandra-driver
Ruby	ActiveRecord	pg	yugabyte-ycql-driver
C#	EntityFramework	npgsql	CassandraCSharpDriver
C++	-	libpqxx	cassandra-cpp-driver
C	-	libpq	-

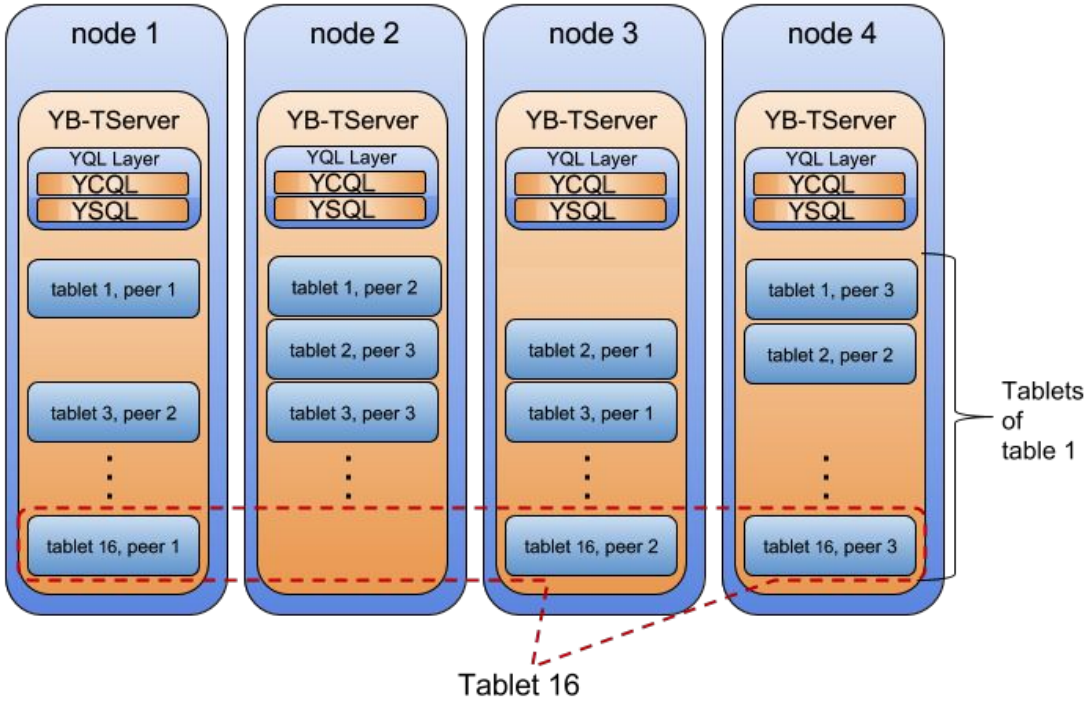
Архитектура



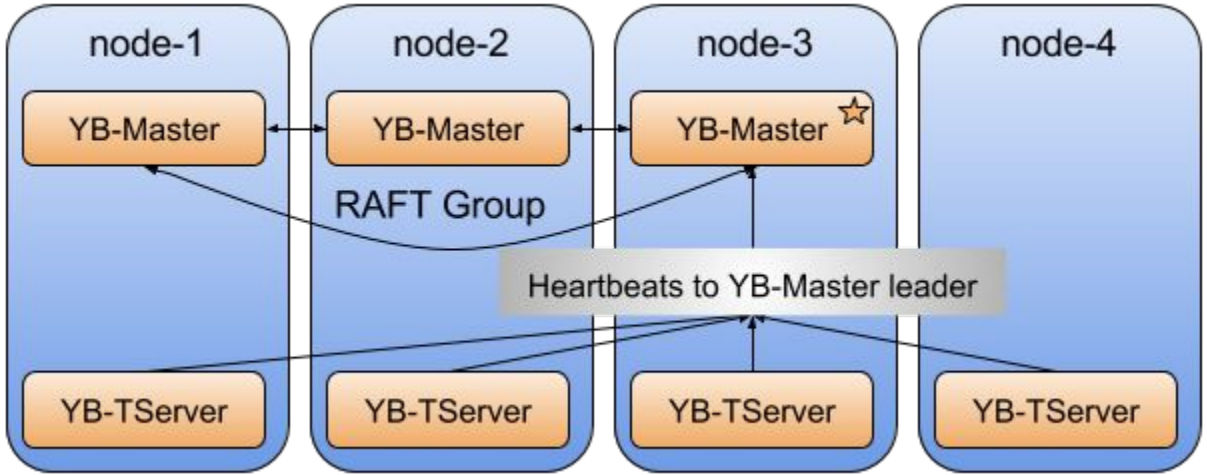
Архитектура. Пример кластера RF3



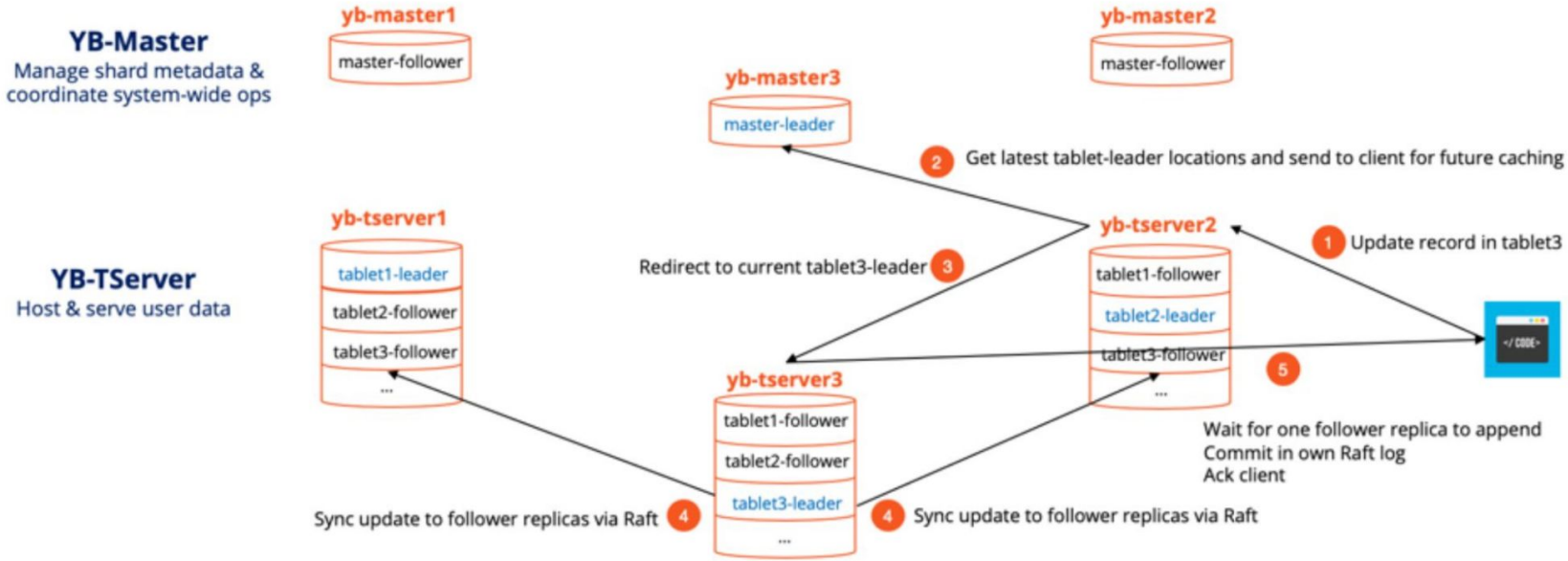
Архитектура. YugabyteDB Tablet Server



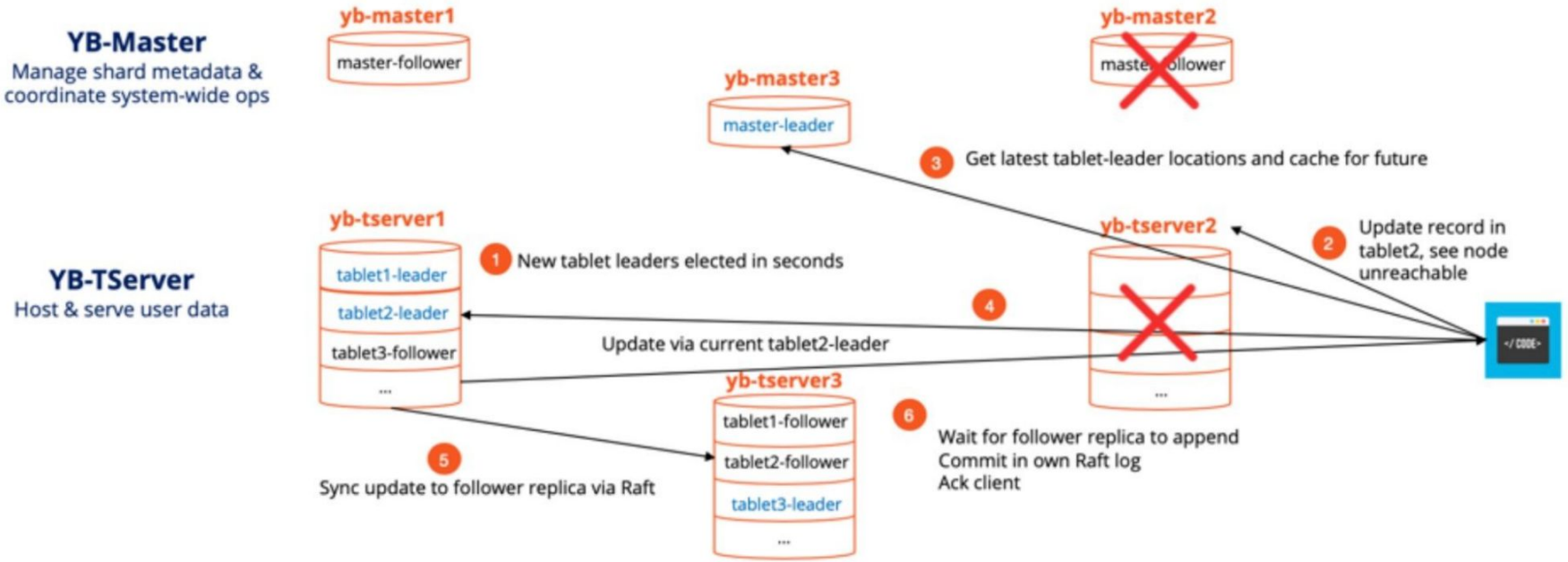
Архитектура. YB-Master



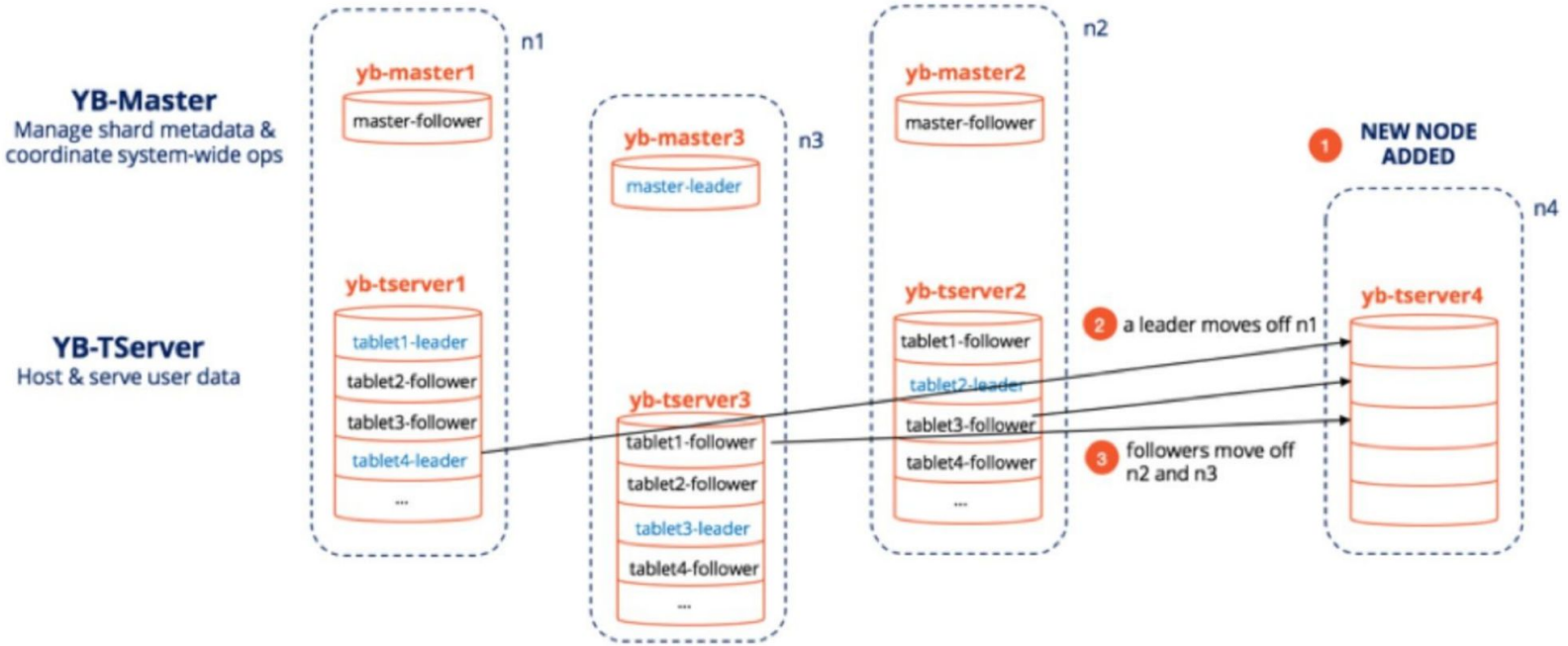
Запись данных в YugabyteDB



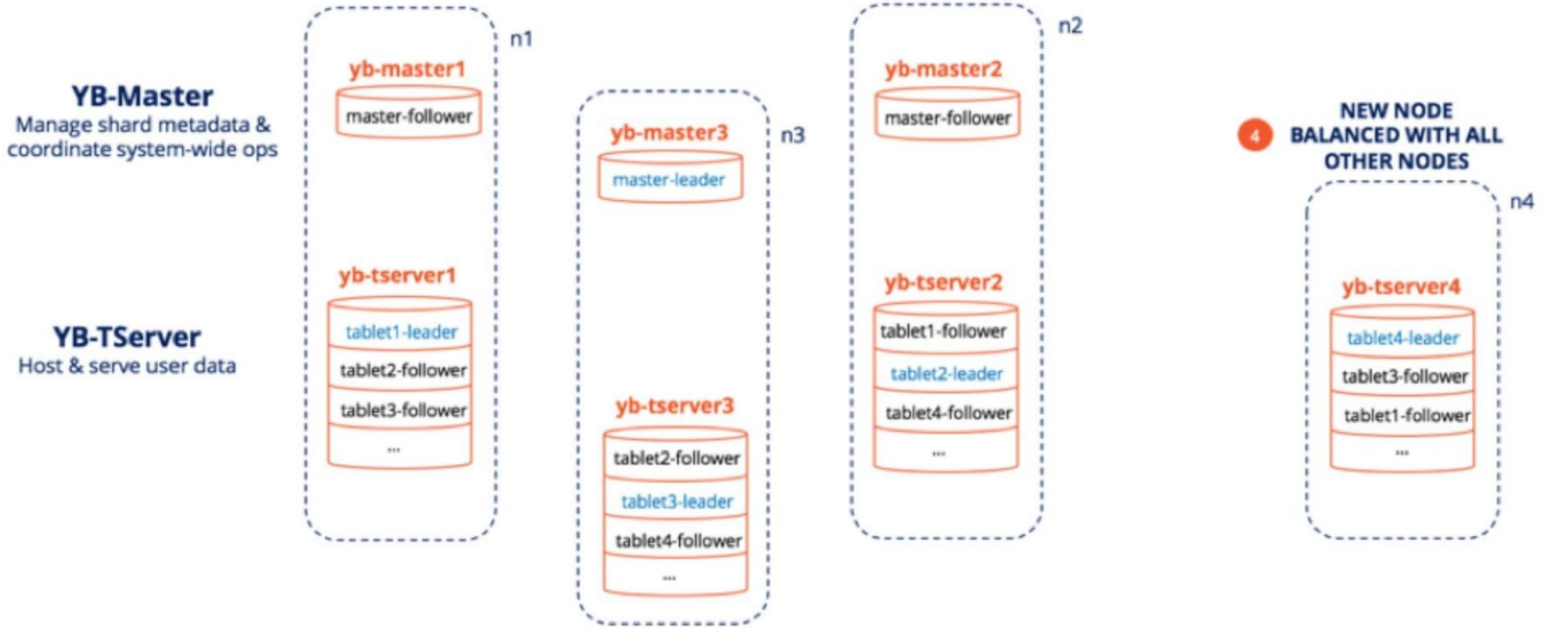
Механизм отказоустойчивости в YugabyteDB



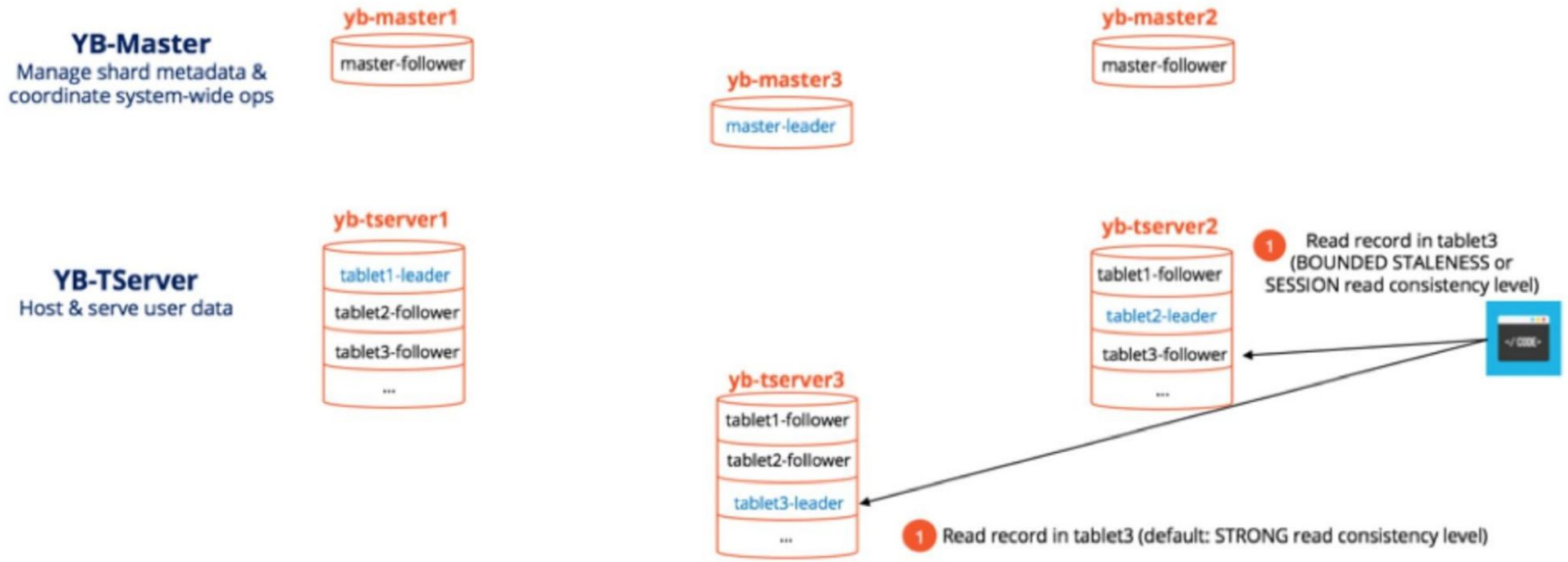
Механизм масштабируемости в YugabyteDB



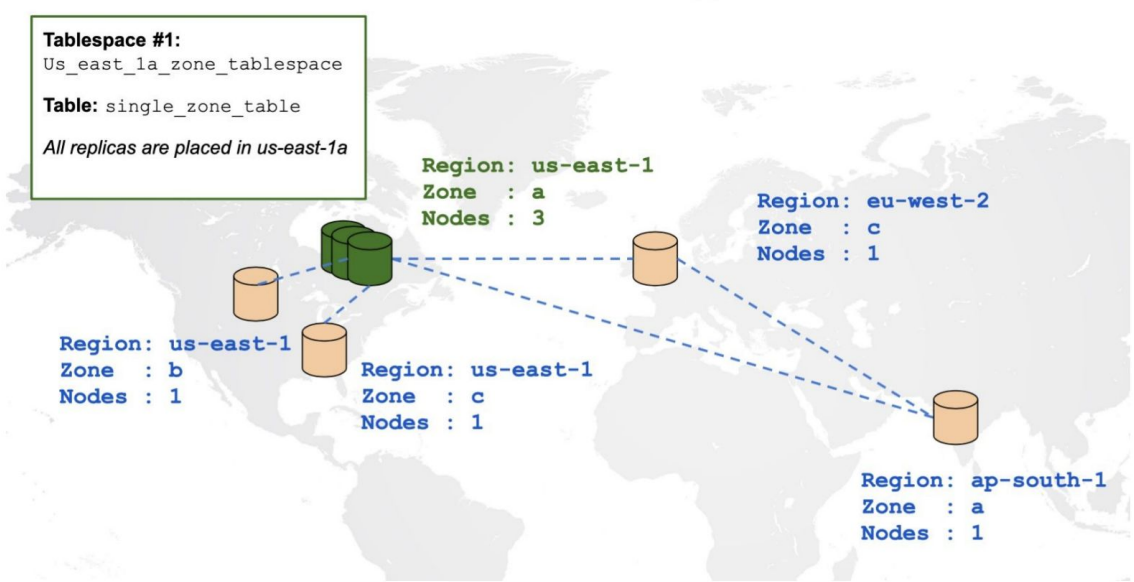
Механизм масштабируемости в YugabyteDB



Чтение данных в YugabyteDB



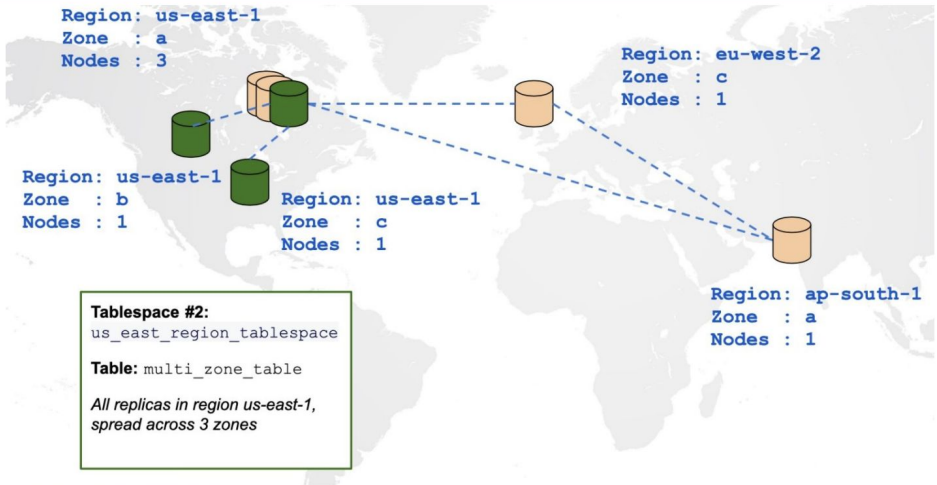
Типовые сценарии. Однозональные данные




```
CREATE TABLESPACE us_east_1a_zone_tablespace
  WITH (replica_placement='{"num_replicas": 3, "placement_blocks": [
    {"cloud":"aws","region":"us-east-1","zone":"us-east-1a","min_num_replicas":3}]}');

CREATE TABLE single_zone_table (id INTEGER, field text)
  TABLESPACE us_east_1a_zone_tablespace SPLIT INTO 1 TABLETS;
```

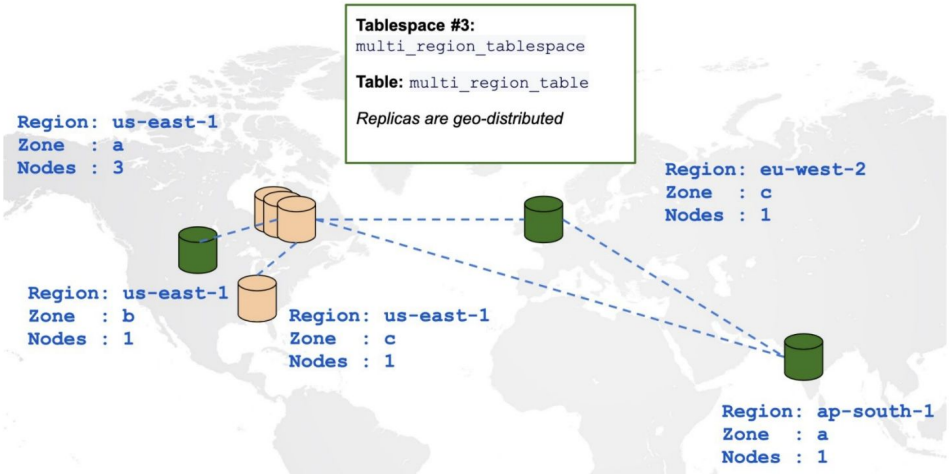
Типовые сценарии. Межзональные данные




```
CREATE TABLESPACE us_east_region_tablespace
  WITH (replica_placement='{"num_replicas": 3, "placement_blocks": [
    {"cloud":"aws","region":"us-east-1","zone":"us-east-1a","min_num_replicas":1},
    {"cloud":"aws","region":"us-east-1","zone":"us-east-1b","min_num_replicas":1},
    {"cloud":"aws","region":"us-east-1","zone":"us-east-1c","min_num_replicas":1}]}');

CREATE TABLE multi_zone_table (id INTEGER, field text)
  TABLESPACE us_east_region_tablespace SPLIT INTO 1 TABLETS;
```

Типовые сценарии. Межрегиональные данные




```
CREATE TABLESPACE multi_region_tablespace
  WITH (replica_placement='{"num_replicas": 3, "placement_blocks": [
    {"cloud":"aws","region":"us-east-1","zone":"us-east-1b","min_num_replicas":1},
    {"cloud":"aws","region":"ap-south-1","zone":"ap-south-1a","min_num_replicas":1},
    {"cloud":"aws","region":"eu-west-2","zone":"eu-west-2c","min_num_replicas":1}]}');

CREATE TABLE multi_region_table (id INTEGER, field text)
  TABLESPACE multi_region_tablespace SPLIT INTO 1 TABLETS;
```

Интересные ссылки

- Построение кластера
- Варианты установки
- Реализация отказоустойчивости и горизонтального масштабирования в YugabyteDB на базе RAFT протокола [1], [2]
- Табличные пространства. Типовые сценарии использования
- Механизмы партиционирования и шардирования в YugabyteDB
- Реализация распределенных ACID-транзакций в YugabyteDB
- Yugabyte SQL в примерах
- Ключевые концепции YugabyteDB

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Arenadata DB

Arenadata DB

Arenadata DB (ADB) — аналитическая, распределённая СУБД, построенная на MPP-системе с открытым исходным кодом Greenplum. Она предназначена для хранения и обработки больших объёмов информации — до десятков петабайт.

Отличие от GP -

- В community-редакции есть дополнительные инструменты мониторинга кластера, средства управления развёртыванием, средства расширения кластера. В enterprise-редакцию включены проприетарные коннекторы, предустановленные расширения, а также система мониторинга запросов Arenadata Command Center.

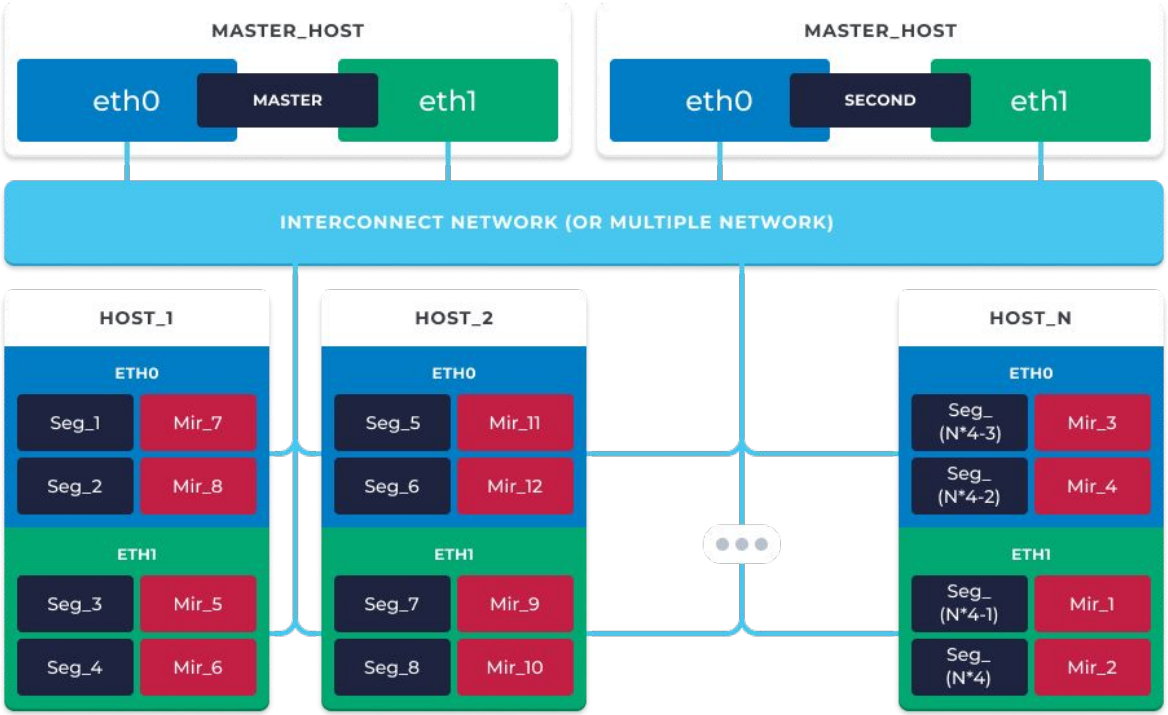
Особенности

- Вся поддержка и экспертиза по внедрению доступна в России и на русском языке;
- Разработан пакет утилит для оффлайн-установки (без доступа к сети Интернет);
- Дистрибутив создан на базе Open-source ядра СУБД Greenplum;
- Полностью российское программное обеспечение;
- Поддержка доступна как удаленно, так и на месте (on-site). Есть набор доступных пакетных сервисов по планированию, установке, аудиту системы;
- Есть возможность доработки и кастомизации продукта под конкретные потребности заказчика;
- Доступна реализация как на “голом железе”, так и в облаке.

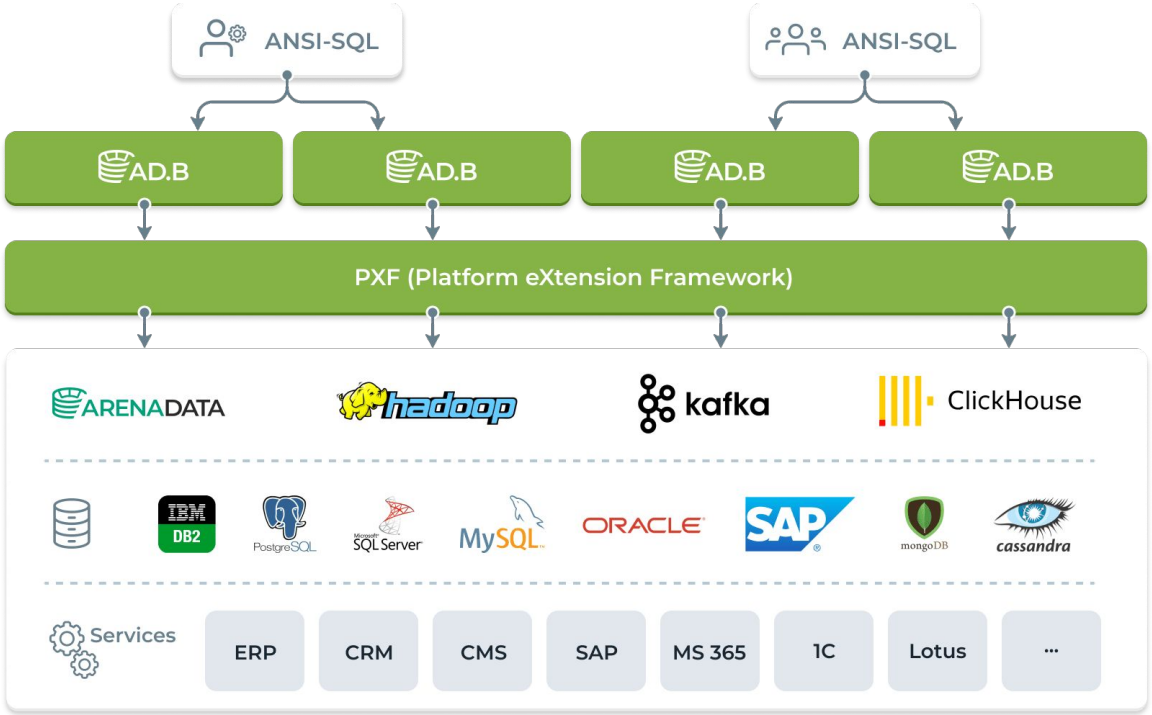
Сферы применения

- Сложные запросы, обрабатывающие большие объёмы данных, в том числе сложные аналитические функции (включая аналитику поведения пользователей);
- Любые типы отчётности (операционная, управленческая, обязательная регуляторная);
- Глубокий AD-HOC анализ;
- Эффективное соединение больших таблиц;
- Работа систем маркетинговых кампаний и систем лояльности;
- Различный скоринг;
- Предсказательная аналитика (спрос, остатки).

Архитектура



Интеграция с внешними решениями



Интересные ссылки

- Типовые варианты оборудования для развёртывания ADB
- https://docs.arenadata.io/adb/index.html
- Arenadata DB. Мощная распределенная аналитическая база данных для больших проектов
- https://vk.com/arenadata
- Варианты установки
- Docker - https://hub.docker.com/r/arenadata/adcm

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Практика

ДЗ

Домашнее задание

- Развернуть Yugabyte или Greenplum в GKE или GCE
- Потесировать dataset с чикагскими такси
- Или залить 10Гб данных и протестировать скорость запросов в сравнении с 1 инстансом PostgreSQL
- Описать что и как делали и с какими проблемами столкнулись

Задание повышенной сложности*

- Развернуть оба варианта и протестировать производительность

Рефлексия

Список материалов для изучения

1. Arenadata
2. Завод, телеком и госсектор: 3 примера внедрения Arenadata
3. About the Greenplum Architecture
4. Introduction To Greenplum Architecture
5. Как начать работать с Managed Service for Greenplum

Цели вебинара

1. Иметь представление MPP СУБД Grennplum
2. Изучить варианты установки

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Алексей Железной

Senior Data Engineer *в Wildberries*
Магистратура - ФКН ВШЭ

Руководитель курсов ***DWH Analyst, ClickHouse для инженеров и архитекторов БД*** *в OTUS*

Преподаватель курсов ***Data Engineer, DWH Analyst, PostgreSQL*** *и пр. в OTUS*

LinkedIn