PostgresPro

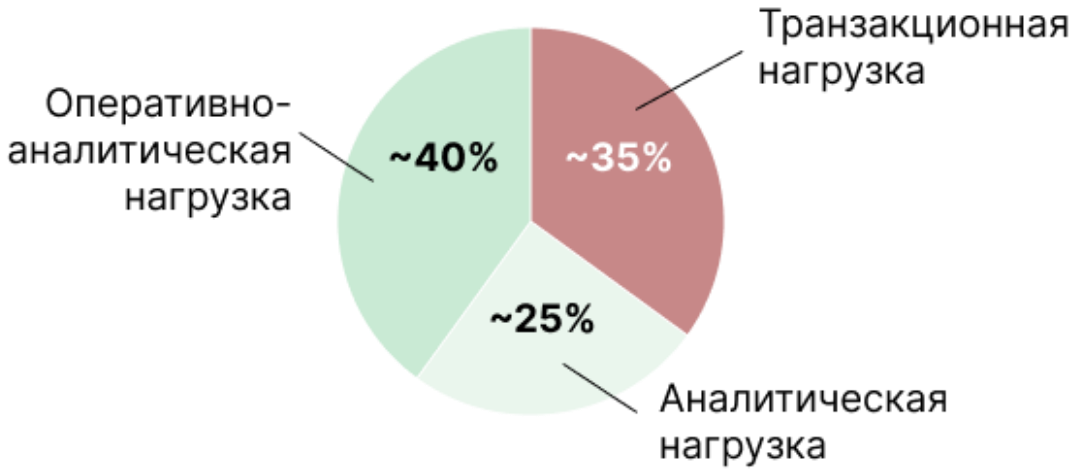
PGCONF.RUSSIA 2024

Распределение транзакционной нагрузки в кластере серверов СУБД

Владимир Сердюк

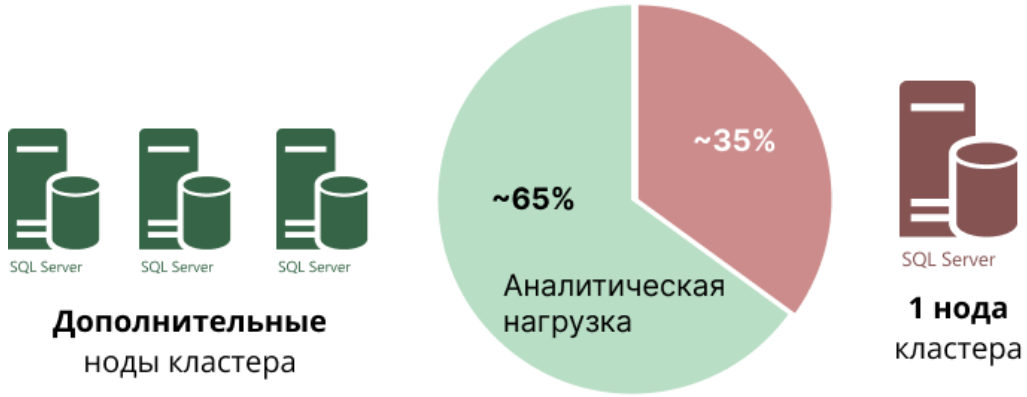
Предпосылки к появлению балансировщика транзакционной нагрузки

Среднестатичтическое распределение нагрузки CPU в OLTP-системах



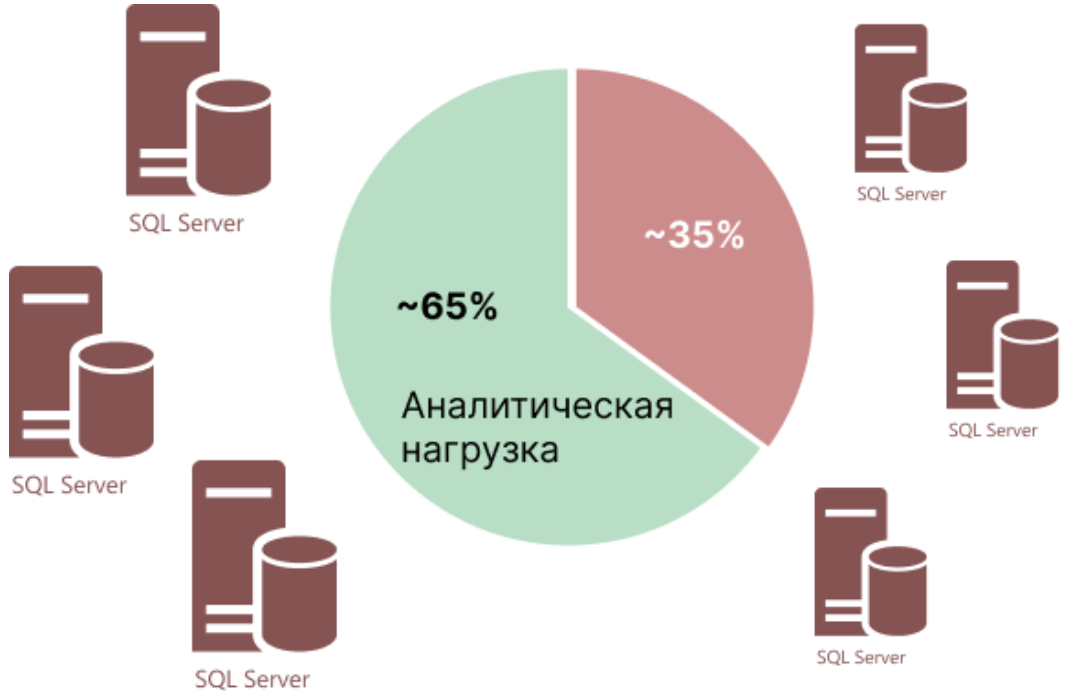
Типичная балансировка нагрузки в кластере СУБД

Нода выделенная под транзакционную не масштабируется и является узким местом в производительности ИТ системы



Предпосылки к появлению балансировщика транзакционной нагрузки

Новый кластер должен распределять между нодами все запросы на чтение, а это может быть до 95% всей нагрузки



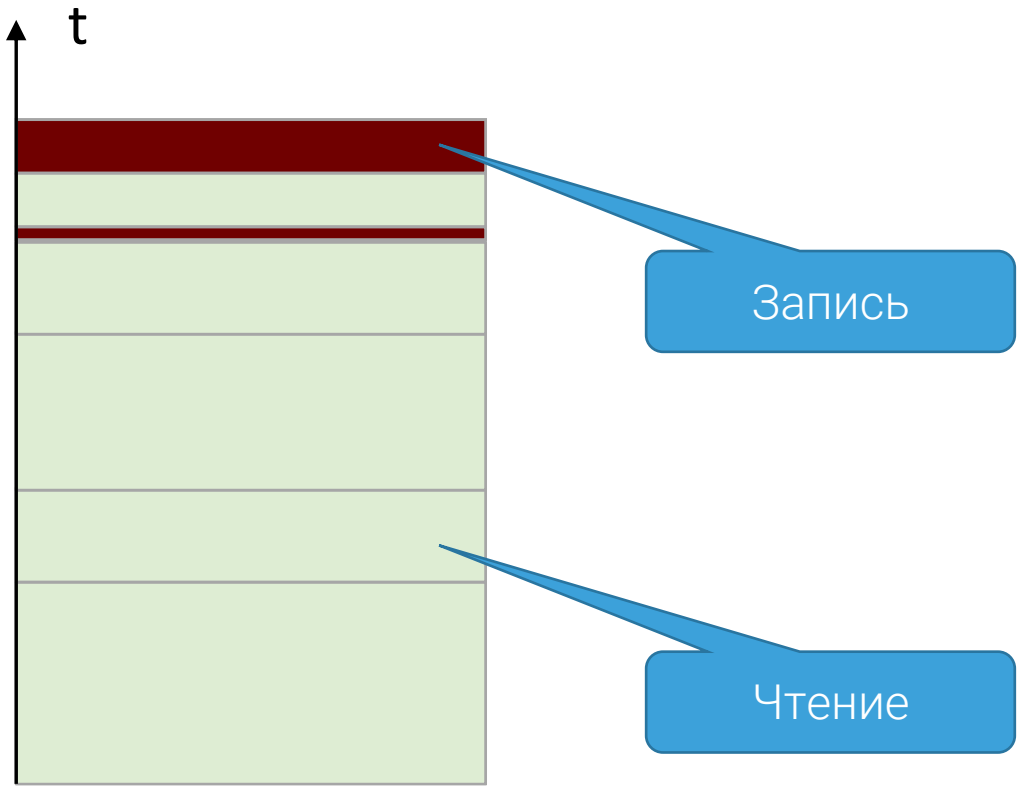
Плюсы

1. Стоимость кластера ниже, чем стоимость одного мощного сервера.
2. Равномерное распределение нагрузки на ИТ-систему по всем серверам кластера
3. В кластере можно использовать достаточно слабые по характеристикам серверы
4. Повышение отказоустойчивости системы.
5. Более плавная амортизация эксплуатационной стоимости серверов

Распределение типа нагрузки в транзакции

В современных OLTP системах, особенно на платформе 1С, транзакции типа «Документ» являются по сути транзакционными мини-отчетами:

1. Большое количество сложных проверок (sql-запросов на чтение) в начале и небольшое количество записей в конце транзакции.
2. Активное использование логики сервера приложений.
3. Не используется сложная логика SQL в запросах на запись.
4. Возможность корректировки данных задним числом.

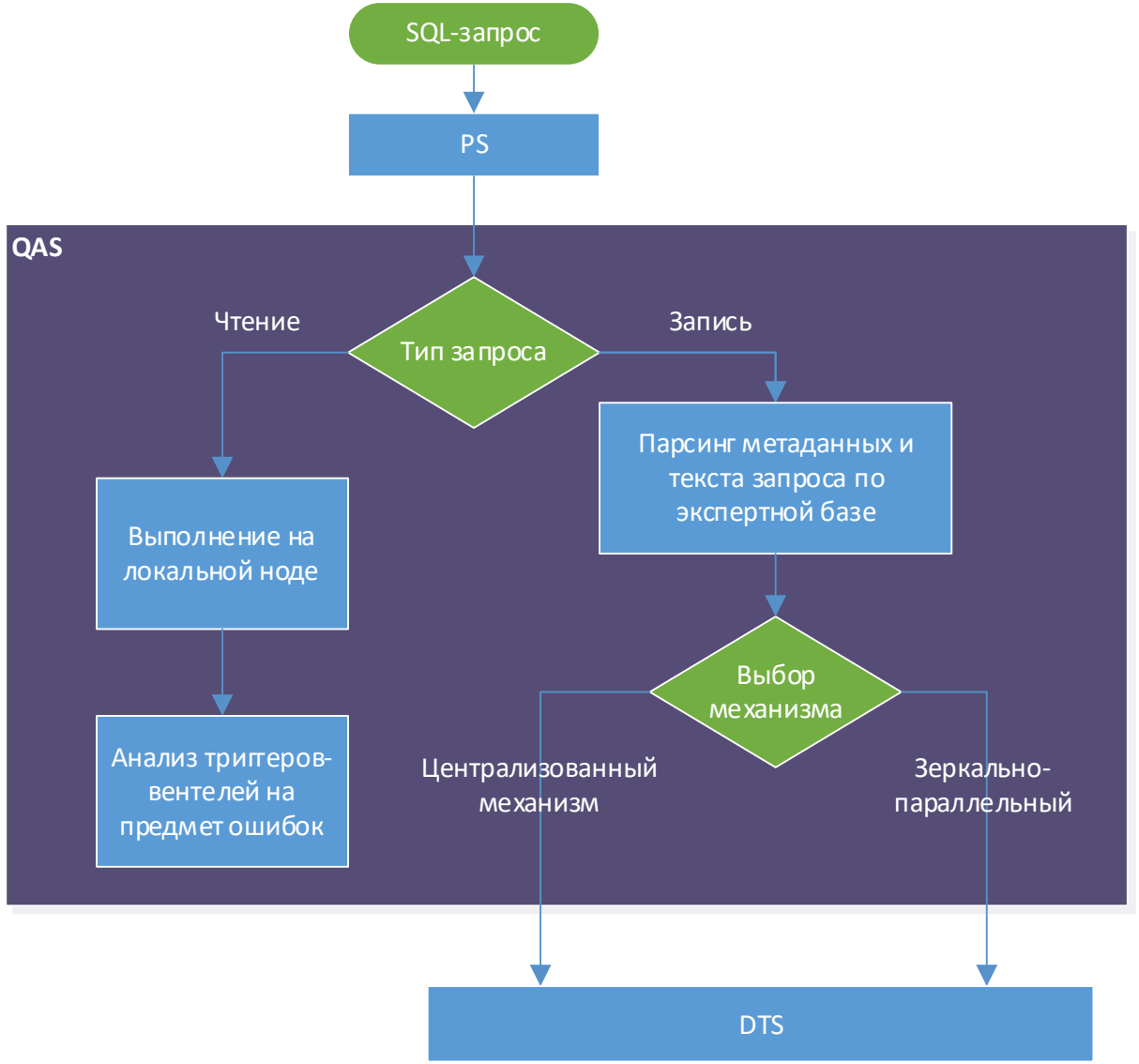


Концептуальная схема прохождения SQL запросов через программные сервисы кластера

PS – Proxy Services. Прокси сервис, отвечает за перехват запросов на уровне драйвера SQL, либо на уровне сервера SQL.

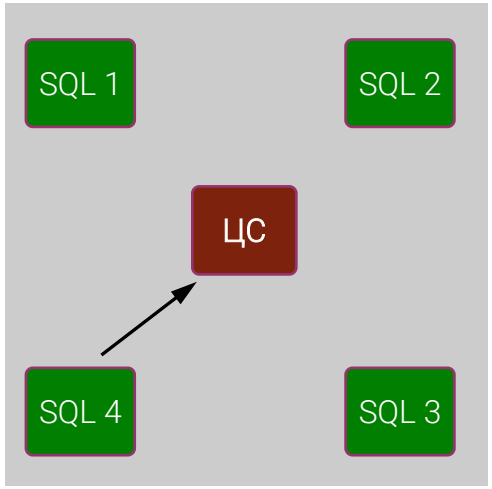
QAS – Query Analysis services. Сервис анализа запросов, его основная функция это понимать запрос на чтение или на запись. Также, он определяет по какому алгоритму обновления данных дальше идти – по централизованному или по зеркально-параллельному.

DTS – Distributed Transaction services. Сервис управления распределенными транзакциями, отвечает за обновления данными на нодах кластера, открытие и закрытие транзакций, обработку ошибок.

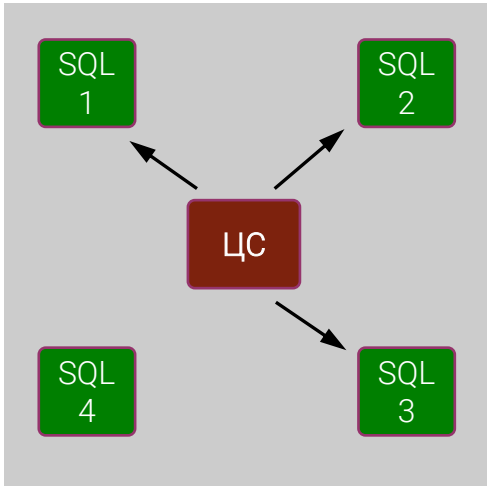


Централизованно асинхронный протокол изменений

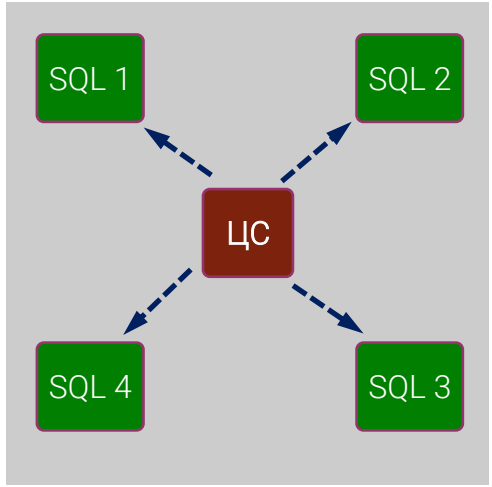
Шаг 1. Перенаправление пакета на запись



Шаг 2. Получение репликационных изменений



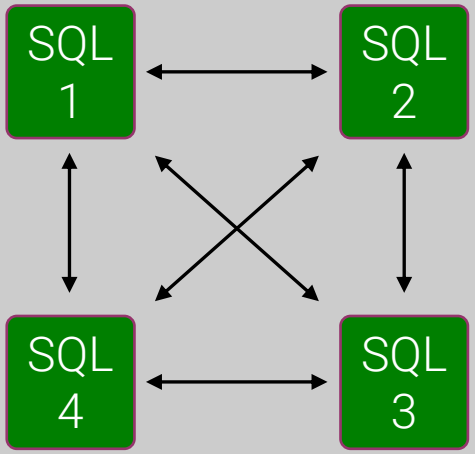
Шаг 3. Закрытие транзакции



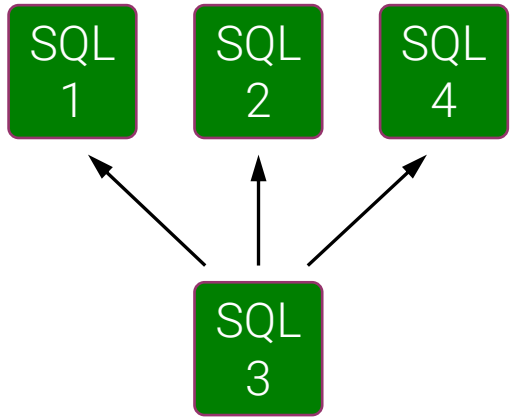
1) Запрос на изменение перенаправляется на центральный сервер.
2) Получаются изменения, инициированные запросом, из репликационных очередей.
3) Изменения перенаправляются на все сервера кластера в том числе на инициатора.
4) Передается контекст управления следующему запросу.

Зеркально-асинхронный(версионный) протокол изменений (каждый с каждым)

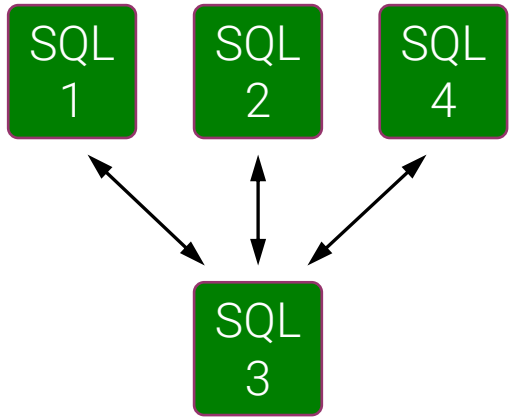
Пример для 4-х серверов



Шаг 1. Отправка пакета на изменение асинхронно на все ноды



Шаг 2. Проверка версий и хэш-сумм изменений на всех серверах



1) Изменение на одном из серверов и следующая за ним мгновенная репликация на все остальные сервера кластера.
2) Контроль версий изменений на серверах. В случае несовпадения откат транзакций (сейчас применяется только для blob полей).
3) Контроль хэш-сумм изменений (состав изменений) в транзакции. В случае несовпадения откат транзакций.

PostgresPro

Почему не репликация

Слишком большие издержки типовых протоколов репликации синхронной транзакцией.

Типовые протоколы распределенной транзакции имеют слишком много дополнительных, в первую очередь временных, сетевых издержек. На один сетевой вызов получается дополнительно до трех вызовов. В подобном виде происходит серьезнейшая деградация простейших атомарных операций.

Не решается конфликт типа Писатель-Писатель

Возникает при одновременном изменении одних и тех же данных в разных транзакциях. По факту прошедшего изменения система «помнит» только абсолютное последнее изменение (или историю). Смысл SQL конструкции для последовательного применения теряется. Подобные конфликты в репликации иногда вовсе не имеют решений.

Оптимизация линейной деградации

Служебные операции в работе кластера, которые могут давать наибольший вклад в задержку

1) Парсинг SQL-запроса → Статус: «НаЧтение», «НаЗапись». Правильный парсинг отнимает ничтожное количество времени.
2) Передача статуса локальному серверу – можно ли менять данные в этой сессии или нет. Потому как в случае ошибки данные рассинхронизируются, что неприемлемо. Это еще один дополнительный сетевой вызов.
3) В случае если операция выполняется не в рамках транзакции то мы должны открыть и закрыть транзакцию – а это еще два сетевых вызова!
4) Если операция на изменение, то
   - Для Версионного варианта изменений выполняется проверка версий и хэш-сумм изменений.
   - Для Централизованного варианта добавляется получение данных с центрального сервера и затем операция реплицирования по всем серверам, сопоставимая по времени с начальной.

ИТОГ – большое количество дополнительных вызовов и как следствие чудовищная деградация простейших атомарных операций

Обеспечение целостности и синхронности данных

Механизмы обеспечения целостности и синхронности распределенных данных на уровне транзакций:

1. Проверка данных транзакций на уровне сверки хэш-сумм.
2. Проверка статуса операций «Чтение-Запись» и откат транзакции в случае ошибки.
3. Распределенная транзакция фиксируется только в случае успешного выполнения зеркальных транзакций на всех нодах, в случае ошибки транзакции откатываются.
4. В случае ошибки при фиксации транзакции – нода отключается из кластера и подключается резервная. Текущая нода ставится в регламент на восстановление данных и подключение к кластеру.

PostgresPro

Механизмы сверки распределенных БД

Таблица для сверки нескольких распределенных БД

Поле	Описание
ИДТабл	Идентификатор таблицы, где фиксируются изменения
ПервичныйКлюч	Первичный ключ записи
ХэшСумма	Хэш-сумма всех колонок записи
ДатаИзменения	Дата последнего изменения

- Таблица заполняется и изменяется триггером. Он пишет ПервичныйКлюч записи и ее хэш-сумму с датой изменения.
- Для сверки двух БД необходимо выгрузить эту таблицу с nolock с одного и другого сервера и сравнить на различие хэш-сумм.

PostgresPro

Механизмы архивирования и восстановления данных в распределенной среде

Для быстрого восстановления данных необходимо для начала восстановить на некоторый момент времени бэкап. Важно чтобы в этот момент времени в БД ничего не писалось. Все инкрементальные данные должны попадать в отдельную репликационную очередь. Именно поэтому должна быть отдельная нода в кластере целью которой будет подготовить все данные для распределенного восстановления. На этой ноде с момента начала создания бэкапа прокси сервер получает указание не писать данные непосредственно в эту базу данных. То есть БД консервируется, все данные накапливаются в инкрементальной, репликационной очереди. Именно из этого бэкапа и имеется возможность восстановить данные на нужный момент времени. Когда станет задача подключить новую ноду к кластеру нужно будет восстанавливать из этого «законсервированного» бэкапа. Затем данные восстанавливаются в многопоточном режиме из репликационной очереди до момента, когда они почти догонят реальное состояние ИТ системы. После чего идет указание прокси серверу поставить на небольшую паузу(5-10 сек.) все транзакции и после этого новая нода включается в подписку на онлайн обновления всех данных в кластере.

Итог. Процедура восстановления в распределенной среде отличается от локальной. Должна быть строго синхронизирована последовательность запуска служб. В приведенном случае для этих целей необходима отдельная нода.

PostgresPro

Разрешение проблемы распределенных дедлоков

Распределенные дедлоки решаются по тем же алгоритмам что и локальные, например с временем ожидания ресурса более чем ВремяX , мы скачиваем на один сервер очередь блокировок и затем с помощью тех же алгоритмов анализируем. Получаем проблемные сессии и откатываем транзакции с ошибкой дедлока , либо убиваем сессии.

Не имеет смысла перебирать и синхронизировать (со всех серверов) все дерево блокировок, это может привести к высоким издержкам производительности и именно по этой причине на устранение дедлока например в MSSQL сервере уходит определенное время. То есть, важно понимать, что какой-то ресурс ожидает другой более времени X. В этом случае потенциально огромное дерево связанных блокировок превращается в небольшое. Например, мы смотрим только те процессы, которые ждут других более 3 секунд. Далее мы начинаем перебирать только связанные. Если в процессе перебора мы входим в замкнутый цикл то это и есть дедлок.

PostgresPro

Создание кластера в приложении с открытым исходным кодом.

Наличие контроля над SQL трафиком

Необходимое и достаточное условие для создания распределенного вычислительного кластера

Реализация

- Создание подсистемы парсинга SQL трафика и анализа метаданных, создание базы знаний.
- Создание подсистемы управления связанными соединениями с нодами кластера а также управления распределенными транзакциями.
- Создание системы разрешения распределенных дедлоков.
- Создание системы сверки и исправления расхождений в распределенных данных.

PostgresPro

Дедлоки в односерверной и распределенной моделях

Deadlock
в односерверной модели
на одной и той же записи

НЕ ВОЗМОЖЕН

даже в теории

Deadlock
в распределенной модели

ВОЗМОЖЕН

Причина – отличие времени доставки пакетов, в следствие чего, есть нарушение порядка выполнения команд на серверах в многопоточной системе.

PostgresPro

Технологии параллельных вычислений как средство, повышающее эффективность использования ПО

Не имеет смысла внедрять систему распределенно вычислительно кластера в неоптимизированную систему которая не эффективно утилизирует мощности одного сервера. Вообще внедрение подобной системы в идеале должно быть стратегическим решением. Разумеется оптимизация в первую очередь эффективных параллельных вычислений является краеугольным принципом для действительно эффективной работы РВК(распределенно вычислительный кластера).

PostgresPro

Спасибо за внимание!

ВЛАДИМИР СЕРДЮК,
генеральный директор SOFTPOINT
softpoint@softpoint.ru