**Лекция 2 - СPT**

**Схемы взаимодействия в информационных системах с горизонтальным масштабированием.**

Межпроцессорное взаимодействие является ключевым аспектом. От этого зависит за сколько будут обрабатываться запросы, сколько пользователей могут одновременно использовать эту систему и на сколько дорого будет содержать и обслуживать систему.

Взаимодействие в информационных системах происходит внутри уровня и между ними.

Клиент-сервер

|  |  |  |
| --- | --- | --- |
| Клиентский слой | Слой серверной логики | Слой БД |
| У1 | Сервер БД 1 | Сервер БД 1 |
| У2 | Сервер БД 2 | Сервер БД 2 |
| У3 | Сервер БД 3 | Сервер БД 3 |

Первый вид взаимодействия происходит между клиентской и серверной частью. Взаимодействие типа многие к одному. В случае горизонтального масштабирования взаимодействуют многие. Реализация данного вида полностью возложена на разработчика ИС.

Следующий вид взаимодействия происходит внутри слоя сервер-приложение. В случае масштабируемой системы на этом слое находится множество серверов и приложений, которые могут выполнять как одинаковые задачи, так и разнотипные. Обмен данными между серверами приложений может происходить двумя способами:

1) Не выходя за рамки слоя. Используется специализированное программное обеспечение для обмена данными. Это могут быть как отдельные системы общения между компонентами и KeyValue хранилища, так и встроенные серверные процессы механизма, которые используют сокеты и другие виды сетевого взаимодействия.

2) Обмен данными с использованием слоя БД. В этом случае данные, которые необходимо передать серверу приложения отправляются в базу данных, а второй сервер приложения производит чтение данных.

Каждый из способов имеет свои плюсы и минусы. Способ использования локальных сред, не выходя за рамки слоя имеет лучшую производительность и экономит ресурсы серверов БД. Однако реализация дополнительного программного обеспечения для серверов и приложений увеличивает сложность разработки, что может привести к снижению отказоустойчивости системы. Ответственность за обеспечение безопасности серверов и баз данных несут разработчики СУБД. Как правило разработчику предоставляются готовые решения, драйверы для различных языков.

Общее время обработки данных состоит из времени, затраченном на клиентском слое, времени затраченном на сервере логики, времени на БД, времени передачи запроса от сервера приложению клиентскому приложению, времени работы алгоритмов бизнес логики, суммарным временем затраченным на обращением любому серверу (зачастую 0), времени затраченным на обработку другого сервера, времени затраченном на передачу запроса от сервера приложения к серверу БД, времени обработки запроса к БД, времени затраченном на передачу запроса БД, времени на передачу сообщения от приложения клиентскому слою.

В свою очередь каждый временной компонент, относящийся к взаимодействию с процессами, состоит из вложенных компонентов. Общее время, затраченное на их взаимодействие, время затраченное протоколом, время затраченное на восстановление сессии, время затраченное на передачу данных.

Оптимизация сетевой инфраструктуры, уменьшение времени на установление сессии, использование StaitLess протоколов, сжатие передаваемых данных, кэширование, использование протоколов с отсутствием проверки целостности пакетов.

Оптимизация сетевой инфраструктуры может заключаться в уменьшении количества промежуточных узлов, использовании новых архитектур и использовании оптимальных протоколов маршрутизации. Уменьшение временных расходов, связанное с восстановлением сессий, достигается путём использования протоколов, которые организуют во время передачи как можно меньшее количество сессий. Существуют протоколы, которые устанавливают сессии под каждый новый запрос. Оптимальные протоколы устанавливают одну сессию, через которую происходит последовательность запросов и ответов.

Использование StaitLess протоколов, то есть протоколов, в сообщении которых отсутствует информация о текущем состояние о принимающей или передающей стороны. В некоторых случаях использование StaitLess протоколов позволяет и вовсе отказаться от сессии, что позволяет мгновенно приступать к передаче данных. Сжатие передаваемых файлов уменьшает размер передаваемых файлов, что снижает нагрузку на инфраструктуру. Кэширование представляет собой промежуточный буфер для данных с возможностью более быстрого доступа. В случае распределенных систем одним из вариантов применения может быть сохранение данных на клиентском устройстве, которые однажды уже были загружены. В прикладном протоколе реализуется механизм, позволяющий проверять, изменились ли данные за то время, которое прошло с момента использования продвинутых современных объектов наиболее часто загружаемых ресурсов позволяет снизить объём трафика, а также сэкономить место на клиентских устройствах.

Синхронизация данных

В распределенных системах невозможно рассчитать точное время, за которое произойдет взаимодействие процесса. В связи с этим были созданы две модели опирающиеся на время: синхронные и асинхронные. Синхронная модель распределенных систем применяет строгие рамки ко времени действий. К ним относятся: 1) локальные часы любого процесса имеют отклонение от идеального времени, строго не превышающее заданный предел; 2) время исполнения каждого действия каждого процесса имеет ограничение сверху и снизу; 3) время обмена между процессами ограниченно верхним и нижним приделами. Нарушение любого из этих правил влечет не правильную работу асинхронной модели или даже к её отказу. Асинхронная модель противоположна: 1) локальные часы могут отклоняться на любую величину от идеального; 2) время исполнения действия конечно; 3) время обмена сообщениями конечно.

В связи с тем, что современные распределительные системы являются сложными и гетерогенными, невозможно предугадать время обмена данными между элементами сети. Современные операционные системы разделяемого времени не дают гарантии заранее определить время на исполнение даже детерминированного алгоритма с постоянной сложностью, поэтому за редким исключением практически все системы РВС являются асинхронными с проблемами асинхронных систем. Основной проблемой асинхронных систем является синхронизация данных.

В результате чего сервер получил ответ не с актуальными данными, тем самым возникло явление рассинхронизации данных. Основным способом борьбы с данным явлением является использование промежуточных узлов, которые берут на себя функции распределения запросов и управления синхронизацией. Репликация данных в режиме MasterSlave все запросы на запись приходят на первичный сервер, который синхронизирует процесс репликации данных. В случае, когда данные необходимо получить без изменений, обращение идёт к вторичным серверам Slave, если необходимо их изменить, то чтение происходит с основного сервера.

**Лекция 3 - СPT**

Первая парадигма совпадает с MasterSlave репликацией MySQL. Вторая, под названием ReplicaSet использует вторичные и первичные серверы. Однако выбор главного сервера происходит автоматически, путем голосования всех серверов. Соответственно в случае выхода главного сервера из строя, процедура синхронизации происходит заново и выбирается первичный сервер. Если обмен данными происходит с использованием одного сервера данных, проблема синхронизации данных также остается актуальной, но только в рамках одного процесса. Соответственно многопоточный процесс параллельно обрабатывает запрос от других процессов, предоставляя им возможности одновременного доступа к одному и тому же ресурсу. Примитивны синхронизации когда необходимо безопасно прочитать ресурсы во время записи одним из процессов. К таким ресурсам относят мьютексы и симофоры по аналоги с ОС.

В трехзвенной модели распределённых систем с горизонтаатльным масштабированием иногда необходимо синхронизировать данные между узлами, находящимися в одном слое. Синхронизация реализуется разработчиками СУБД посредством встроенных механизмов и драйверов БД. Для сервров приложений синхронизация требуется не всегда. В достаточно редких случаях серверная логика позволяет масштабировать его разработчиков в самостоятельные \_\_\_ без потери функциональности. Однако существует ситуация, когда необходим прямой и быстрый доступ к данным. Примером такой системы выступает система, в состав которой входят различные приложения, выполняющие разный функционал. Обрабатывает бизнес логику сервера, запись информации в БД, конвертирует изображения. Двусторонняя связь в данной архитектуре для того, чтобы можно было передавать сообщения на исполнения задачи от сервера логики к серверам обработки информации.

Примером такой информации являются данные об сессиях пользователей в веб-приложениях. Сессия представляет из себя набор данных, состоящий из следующих полей: данные о сессии пользователя. Токен представляет из себя хэш, который используется при авторизации пользователя. Токен имеет свой срок действия. Любые поля, которые характеризуют состояние клиента. Для обеспечения безопасности корректно реализованные приложения производят авторизацию перед каждым действием вне зависимости от характера этого действия, будь то чтение\удаление\записать данных. Операция чтения данных является очень частой процедурой. А авторизация этой операции производится с такой же частотой. Хранение данных о сессиях увеличивает хранение данных как минимум в 2 раза, что в свою очередь ведет за собой увеличение сетевого трафика между серверами приложения и серверами БД, а также увеличением выполнения алгоритмов бизнес логики в связи с увеличением времени ожидания этих данных. Важно, что зачастую реляционные базы данных не эффективны при выполнении частых и простых запросах.

1. Синхронизация через слой БД является простым решением в реализации, но не подходит, когда требуются высокопроизводительные решения, а в качестве БД выступают реляционно.

Домашнее задание: Memcached была написана на Perl, переписана на C. Эту технологию использовали высоконагруженные приложения. Но пришёл Redis. Проанализировать плюсы и минусы, в каких ситуациях использовать тот или иной инструмент?