УДК 004.9

ПРОБЛЕМА МАСШТАБИРУЕМОСТИ РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Нагорных Р.И, студент гр. ПИм-191, I курс
Научный руководитель: Крюкова В.В., к.т.н., доцент
Кузбасский государственный технический университет имени Т.Ф. Горбачева, г. Кемерово.

Все чаще в современном мире применяются повышенные требования к производительности, гибкости и отказоустойчивости информационных систем. Привычного приложения, работающего на одном устройстве уже не хватает для удовлетворения этих требований, поэтому все чаще даже небольшие системы разрабатывают распределенными.

Одной из важнейших характеристик распределенных систем является масштабируемость. Масштабируемость - способность информационной системы адаптироваться к резкому изменению показателей задач и повышению требований (например, увеличение объемов данных, числа пользователей и т.д.) [1]. При построении масштабируемых систем часто приходится сталкиваться с ограничениями вычислительных систем, связанных с централизацией данных, алгоритмов и служб. Все проблемы, возникающие при масштабировании распределенных систем, могут быть решены еще на этапе проектирования.

Цель работы – исследовать подходы к обеспечению масштабируемости распределенной информационной системы, выявить их достоинства, недостатки и проблемы, а также пути решения проблем.

При централизованном подходе масштабируемость системы сильно ограничивается мощностями конкретного сервера. В случае, когда нагрузка на систему возрастает, производится замена комплектующих на более мощные или, иными словами, система подвергается вертикальному масштабированию. Даже самые мощные компоненты имеют предел нагрузки. При его превышении у пользователей возникают проблемы с связанные со стабильностью и доступностью системы, что приводит к убыткам компании или организации.

Для достижения хорошей масштабируемости распределенной системы обычно рекомендуют уходить от централизации и переходить к распределенным архитектурам данных и сервисов [2].

Распределенную архитектуру данных можно подвергнуть вертикальному и горизонтальному масштабированию, что очень хорошо сказывается на денежных затратах, ведь цена за один очень мощный сервер может быть намного выше, чем за несколько серверов послабее. При этом несколько менее производительных машин могут справляться с нагрузкой даже лучше, чем один мощный сервер. Существует два основных способа распределения данных по нескольким узлам – секционирование и репликация.

Репликация – способ хранения копий данных на нескольких машинах, соединённых с помощью компьютерной сети. Она дает возможность горизонтального масштабирования для машин, обслуживающих запросы на чтение, что увеличивает пропускную способность по чтению. Основная проблема репликации заключается в изменении реплицируемых данных. Для решения данной проблемы на текущее время наиболее актуальны следующие алгоритмы репликации изменений: репликация с одним ведущим узлом, с несколькими ведущими узлами, без ведущего узла.

Наиболее распространенным является алгоритм с ведущим узлом. Основным недостатком данного алгоритма является создание ограничении в масштабировании системы. В данной архитектуре операция записи проходит через один ведущий узел, а операции чтения можно выполнять с любого узла. Эта особенность дает возможность привести систему к архитектуре, масштабируемой по чтению. В этом случае создается множество ведомых узлов, которые обрабатывают запросы на чтение, в результате чего нагрузка на ведущий сервер снижается. Но данный подход работоспособен при использовании асинхронной репликации, так как в случае использования синхронной репликации система становится ненадежной – отказ любого из узла приведёт к сбою в работе всей системы. Если же в данном алгоритме использовать асинхронную репликацию, то возможно появление несогласованности базы данных – состояния, при котором данные на ведомом узле будут неактуальны из-за задержек репликации. Также у данного алгоритма есть еще один серьезный недостаток – запись данных в систему производится только одним ведущим узлом. Если по какой-то причине он будет не доступен, то произвести запись данных будет невозможно. Для решения этой проблемы был разработан алгоритм с несколькими ведущими узлами.

Репликацию с несколькими ведущими узами также называют репликацией типа главный-главный или «активный/активный». В данном алгоритме каждый из ведущих узлов одновременно является и ведомым для других узлов. Данный схему целесообразно использовать в случае репликации данных на нескольких ЦОДах и невыгодно в случае одного по экономическим соображениям. Если использовать данную схему на нескольких ЦОДах то в сравнении с репликацией с одним ведущим узлом виден выигрыш в производительности и устойчивости системы. Но недостатком данной схемы является возможность возникновения конфликтов в системе из-за изменения одних и тех же данных в разных ЦОДах. В целом данный алгоритм хранит в себе немало тонкостей и подводных камней. Так, большие сложности могут возникнуть в случае использования автоматически увеличиваемых ключей или триггеров. Поэтому рекомендуется по мере возможностей избегать использования данного алгоритма.

Другим подходом к репликации является использование алгоритма, в котором нет разделения на ведущие и ведомые узлы. Базы данных, использующие данный подход, называют Dynamo-подобными базами данных.

Данный алгоритм обладает хорошей отказоустойчивостью из-за наличия в нем механизмов разрешения конфликтов при чтении и противодействия энтропии. Механизм разрешения конфликтов позволяет клиентам получать актуальные данные даже в случае наличия устаревших данных на отдельных серверах. Достигается это за счет того, что клиент посылает сразу несколько запросов к различным серверам, а клиент выбирает актуальную версию по ключу. Процесс противодействия энтропии заключается в поиске отсутствующих данных в одной реплике и копировании актуальной версии с другой реплики.

Другим способом распределения данных является секционирование. Секционирование – разбиение большой базы данных на небольшие подмножества, называемых секциями, в результате чего разным узлам можно сопоставить различные секции. Фактически каждая секция является отдельной базой данных. Основной целью секционирования данных является масштабируемость. За счет того, что данные будут распределены по отдельным секциям, можно очень просто масштабировать пропускную способность по запросам путем добавления новых узлов. Секционирование нередко используется совместно с репликацией, что повышает отказоустойчивость системы. Основными подходами к секционированию являются хеш-секционирование и секционирование по диапазонам значений ключа.

Секционирование по диапазонам значений ключа – вид секционирования, при котором ключи сортируются, и секция содержит все ключи, начиная с определенного минимума до определенного максимума. Преимуществом такого подхода является удобное извлечение необходимых данных, а недостатком является то, что некоторые паттерны способны привести к появлению горячих точек в системе. Горячая точка- секция с непропорционально высокой нагрузкой.

Хеш-секционирование подход, при котором вычисляется хеш-функция каждого ключа и к каждой секции относится определенное количество ключей. Данный метод нарушает упорядоченность ключей, делая запросы по диапазонам неэффективными, но позволяет более равномерно распределить нагрузку. Вероятность возникновения горячих точек в данном методе меньше, чем при секционировании по диапазонам значений ключа [3].

Достичь хорошей масштабируемости также может помочь разделение системы на отдельные сервисы. При монолитной архитектуре приложения разработчики начинают сталкиваться с проблемами уже на этапе разработки. Основной проблемой является организация командной работы. Другой большой проблемой, относящийся к этапу сопровождения системы, является ее превращение в «грязный шарик». В этой ситуации не один из членов команды не понимает работу системы. Но данную архитектуру проще разрабатывать и тестировать в случае, когда речь идет о небольшом проекте. Кроме того, монолит довольно прост в развертывании и масштабировании до превращения системы в «грязный шарик».

Другой подход к построению системы заключается в ее дроблении на отдельные сервисы. Он становится особенно актуальным, когда требуется масштабируемость системы. Если система будет разбита на несколько компонентов то, это позволит упростить процессы разработки и сопровождения. Разработка системы упроститься за счет появления отдельных зон ответственности. Упроститься и масштабирование системы, так как команда, разрабатывавшая отдельный элемент, знает все его преимущества и недостатки и соответственно сможет лучше его масштабировать. Но и у микросервисной архитектуры есть свои недостатки. При данном подходе придется иметь дело с частичными отказами, что оказывает неблагоприятное влияние на доступность системы. Кроме того, нужно хорошо проработать над взаимодействием между сервисами. Также микросервисную архитектуру сложнее тестировать.

В масштабировании уже разработанной распределённой информационной системы могут помочь облачные среды исполнения приложений. Выделяют следующие типы облачных сред:

- облачные серверы;
- вычислительные слайсы;
- динамические контейнеры;
- микровычисления.

Простейшим способом перехода к масштабируемым вычислениям является использование облачных сервисов. Основным преимуществом облачных серверов является возможность является довольно быстрое развертывание и свертывание системы, что является важным показателем для масштабирования системы. Другими преимуществами данного способа являются:

- дешевый вычислительный цикл;
- минимальные функциональные ограничения;
- непрерывная работа.

К недостаткам относятся:

- оплата за выделенную, но не использованную мощность;
- неэффективное использование физических серверов;
- поддержка сервера со стороны владельца.

Вычислительные слайсы – модель выполнения приложения без предоставления сведений о сервере, на котором оно выполняется. Преимущества данного подхода:

- простое изменение мощности;
- автономность серверов.

Недостатки:

- высокая стоимость в сравнении с облачными серверами;
- оплата за выделенную, но не использованную мощность;
- отсутствие информации о серверах.

Динамические контейнеры – технология, заключающаяся в выделении и перемещении контейнеров между серверами, с целью создания управляемой и масштабируемой среды.

К преимуществам данного подхода можно отнести:

- невысокая цена;
- простое масштабирование;
- непрерывная работа;
- простое развертывание.

Недостатком является необходимость управления контейнерами и покупка программного обеспечения для этих целей.

Микровычисления – это модель, в которой относительно простой фрагмент кода доступен для выполнения. Он запускается, когда необходимо получить ответ на некоторое входное событие, например, запрос пользователя или вызов службы.

Преимущества:

- безграничное масштабирование;
- автономность серверов;
- оплата только за использованную мощность;
- легкость масштабирования.

Недостатки:

- ограниченная функциональность и поддержка языков;
- архитектурная ограниченность – архитектура приложения должна быть заточена под микровычисления;
- самый дорогой вычислительный цикл [4, 5].

Таким образом, в результате проведенного анализа была выявлена основная проблема, ограничивающая масштабируемость системы – централизация. Были рассмотрены основные способы распределения данных, такие как репликация и секционирование. Данные подходы позволяют уйти от централизованной архитектуры данных и этим оказать положительное влияние на масштабируемость. Также было выявлено, что построение системы отдельными сервисами позволяет лучше масштабировать ее за счет появления отдельных зон ответственности. Проведен краткий обзор облачных сред исполнения приложений, позволяющих эксплуатировать и масштабировать разработанную систему.

Список литературы:

1. В. Я. Цветков, А. Н. Алпатов. Проблемы распределенных систем // Международный электронный научный журнал «Перспективы науки и образования» [Электронный ресурс]. – Режим доступа https://pnojournal.files.wordpress.com/2014/07/pdf_140605.pdf , свободный (дата обращения 10.03.2020).

2. Э. Таненбаум. Распределенные системы. Принципы и парадигмы/ Э. Таненбаум, М. ван Стеен. – М.: Питер, 2003. – 877 с.

3. М. Клэпман. Высоконагруженные приложения. Программирование, масштабирование, поддержка. – СПб.: Питер, 2018. – 640 с.

4. Л. Атчисон. Масштабирование приложений. Выращивание сложных систем. – СПб.: Питер, 2018. –256 с.

5. Библиотека программиста proglib [Электронный ресурс]. Режим доступа: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16 , свободный (Дата обращения 23.02.2020).