АЛГОРИТМЫ И МЕТОДЫ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Саркисян Э. Л.
Кафедра информационных технологий автоматизированных систем,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: evelina.sarkisyan.1990@mail.ru

Рассматриваются методы оптимизации веб приложений. Предлагается улучшение существующего подхода для ускорения работы приложения.

Введение

Управление производительностью приложений является мощным инструментом повышения качества работы бизнес-приложений. Поэтому разработчики программного обеспечения уделяют этому вопросу все больше внимания. Оптимизация — это процесс модификации программной системы с целью повышения эффективности ее работы или использования меньшего количества ресурсов.

Архитектурный дизайн системы оказывает особенно сильное влияние на ее производительность. Выбор алгоритма влияет на эффективность больше, чем любой другой элемент дизайна. Более сложные алгоритмы и структуры данных могут хорошо оперировать большим количеством элементов, в то время как простые алгоритмы подходят для небольших объёмов данных [1]. Для оптимизации требуется найти узкое место: критическую часть кода, которая является основным потребителем необходимого ресурса. Утечка ресурсов также может привести к падению скорости выполнения программы.

В работе рассмотрены методы улучшения производительности приложений за счёт внедрения асинхронности и событийно-ориентированного программирования, а также предложен способ оптимизации алгоритма обработки сообщений для сокращения общего времени выполнения.

I. Применение асинхронности в приложении

Синхронные процессы, как и программы, писать и отлаживать намного проще, поэтому такой подход к конструированию процесса очень сильно распространен. Однако синхронные решения приводят к сбою пользовательского интерфейса, плохой масштабируемости и расширяемости. В синхронном коде каждая операция ожидает окончания предыдущей. Все действия выполняются строго последовательно, в том порядке, в котором они записаны в исходном коде программы. Поэтому вся программа может зависнуть, если одна из команд выполняется очень долго. На рисунке 1 изображено сравнение синхронного и асинхронного механизмов.

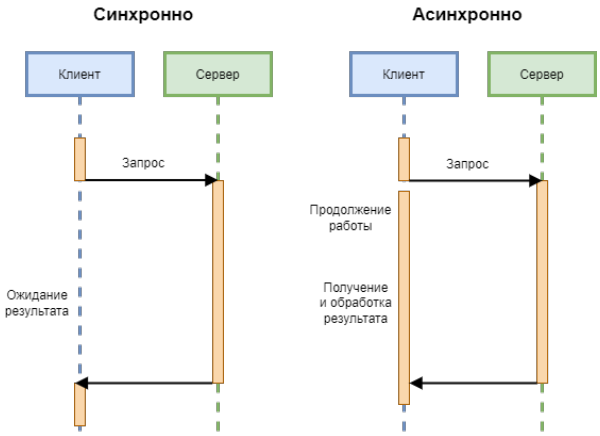

Рис. 1 – Синхронный и асинхронный механизмы клиент-серверного взаимодействия

Асинхронный код делает возможным выполнение процесса в неблокирующем режиме системного вызова, что позволяет потоку программы продолжить обработку. Таким образом, главный «процесс» ставит задачу и передает ее другому независимому «процессу». Использование кода асинхронного программирования позволяет освободить поток выполнения, из которого он был запущен, что приводит к экономии ресурсов, а также предоставляет возможность параллельных вычислений. Асинхронность при передаче данных обеспечивает систему двумя важными свойствами:

- Эластичность - способность масштабироваться горизонтально;
- Устойчивость - способность справиться с отказом и восстановить систему.

Благодаря этим двум характеристикам система становится отзывчивой. Она может адаптироваться к высоким или низким нагрузкам и продолжать обслуживать запросы в условиях высоких нагрузок или отказов. Этот набор принципов имеет первостепенное значение при построении высоко распределенных систем. Необходимо запускать несколько экземпляров сервисов, чтобы сбалансировать нагрузку и справиться с неисправностями без нарушения доступности.

Взаимодействие сообщений позволяет компонентам справляться с отказом локально. Благодаря асинхронному аспекту компоненты не

ожидают ответов, поэтому сбой в одном компоненте не повлияет на другие компоненты [2].

II. Событийно-ориентированное программирование и алгоритм его оптимизации

В отличие от систем «запрос-ответ», в событийно-ориентированной архитектуре отправляющая сторона передаёт «событие», содержащее полезные данные, в брокер сообщений, после чего сообщение рассылается всем сервисам, подписанным на топик.

Программирование, управляемое событиями, — это шаблон архитектурного проектирования для создания программного приложения, в котором компоненты среды выполнения создают события и реагируют на них определенным образом. Подобный механизм гарантирует, что сообщения никогда не будут потеряны. Они будут поставлены в очередь для доставки, как только потребитель будет готов к их получению.

На рисунке 2 представлена схема, состоящая из одного отправителя, брокера сообщений и трёх получателей, один из которых не получает сообщения, так как подписан на другой топик.

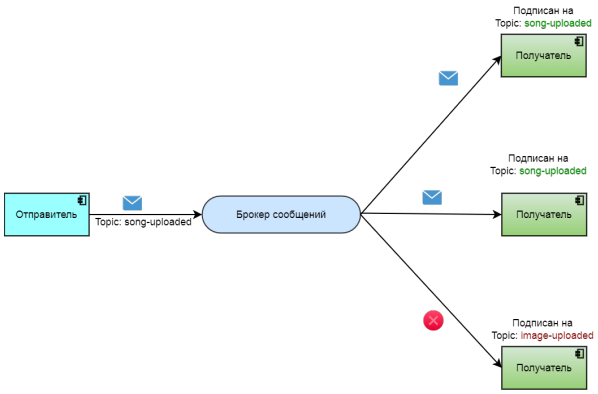

Рис. 2 – Схема событийно-ориентированного программирования

Одним из недостатков использования архитектуры “запрос-ответ” является наличие так называемой единой точки отказа. Если сервис выходит из строя хотя бы на какое-то время, то весь рабочий поток может быть нарушен. Кроме того, при необходимости отправить запрос одновременно нескольким получателям, каждая операция должна выполняться отдельно, что может значительно задерживать основной поток от продолжения работы.

Здесь событийно-ориентированная архитектура наиболее ярко проявляет свое преимущество. Это связано с тем, что слои распределения могут иметь несколько серверов, выделенных для хранения сообщений, и могут горизонтально масштабироваться по мере необходимости. И затем, даже если одна из систем выйдет из строя, другие сервера смогут перехватывать и ретранслировать сообщения [3].

Архитектура, управляемая событиями, сочетает в себе идентификацию шаблонов данных с автоматическими оповещениями и уведомлениями нужных сервисов. Это позволяет предприятиям принимать оперативные решения в режиме реального времени.

С помощью механизма постановки в очередь сообщения могут быть сохранены, в случае отключения сервиса-потребителя, и вновь отправлены, как только потребитель снова начинает прослушивать сообщения. Необработанные сообщения или те сообщения, которые привели к ошибке, помещаются в отдельную очередь "безуспешных сообщений"с целью последующего извлечения и исследования.

Однако такой подход перестает быть эффективным для более сложных систем, где количество сообщений в брокере может стать чрезмерно большим, особенно при временной недоступности обработчика.

Анализируя данную проблему, можно прийти к выводу, что причиной является обработка всего одного сообщения за раз. При невысокой нагрузке на систему такой подход не имеет недостатков, однако для обеспечения надёжности системы требуется оптимизация.

Для достижения данной цели мной была предложена реализация многопоточного алгоритма для потребления и обработки сообщений из брокера. Таким образом, имея всего один экземпляр обработчика, можно достичь более высокой эффективности. Данный оптимизированный алгоритм также способен динамически регулировать количество запущенных потоков в зависимости от нагрузки. Таким образом, ресурсы системы могут быть оптимально использованы.

III. Выводы

Асинхронное программирование является очень мощным инструментом для оптимизации высоконагруженных программ с частым ожиданием системы. Благодаря инновационным инструментам событийно-ориентированные архитектуры становятся более гибкими, универсальными и надежными, что удовлетворяет множество актуальных бизнес-запросов.

В результате работы был разработан алгоритм для эффективного межсервисного взаимодействия. Применение рассмотренных архитектур поможет оптимизировать веб-приложение и в разы сократить время ожидания.

1. Эванс Бенджамин, Гоф Джеймс, Ньюланд Крис. Java, оптимизация программ, практические методы повышения производительности приложений в JVM. - 2019. - с.448
2. Clement Escoffier. Building Reactive Microservices in Java. - 2017. - с.83
3. Ben Stopford. Designing Event-Driven Systems. - 2018. - с.166