Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафеда электронных вычислительных машин

РЕФЕРАТ
по курсу
«Технологии распределенных вычислений и анализа данных»
на тему
«Распределенные вычисления и координация ресурсов в GRID системе микросервисной архитектуры для обработки изображений на основе искусственного интеллекта»

Выполнила: магистрант группы 355841
А.В. Деркач

Проверил: к.т.н., доцент каф. ЭВМ
Д.Ю. Перцев

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном информационном обществе непрерывный поток данных становится неотъемлемой частью нашей повседневной жизни. Изображения играют ключевую роль в этом потоке, представляя собой богатый и информативный источник данных, используемых в различных областях, включая медицину, науку, медиа, маркетинг и многие другие. Однако, объем и сложность этих изображений постоянно возрастает, что создает огромные сложности для их обработки, анализа и управления. В связи с этим, эффективное использование вычислительных ресурсов и передовых технологий становится необходимостью.

Искусственный интеллект, с его способностью к автоматизации анализа и обработки данных, становится важным инструментом в области обработки изображений. Однако, для того чтобы эффективно использовать потенциал искусственного интеллекта, требуются не только высокопроизводительные вычислительные ресурсы, но и современные архитектурные решения, способные обеспечить распределение и координацию этих ресурсов.

Особую важность и интерес представляют распределенные вычисления и GRID-системы в совокупности с микросервисной архитектурой. Они обеспечивают инфраструктуру и механизмы для эффективного управления и использования вычислительными ресурсами, размещенными по всей сети. Микросервисная архитектура, в свою очередь, обеспечивает гибкость и масштабируемость, позволяя создавать сложные приложения из независимых компонентов. Это особенно важно в контексте обработки изображений, где различные задачи требуют разнообразных вычислительных ресурсов и специализированных алгоритмов.

В данном реферате будут рассмотрены принципы и технологии искусственного интеллекта, GRID систем и микросервисной архитектуры, а также их применении в контексте обработки изображений на основе искусственного интеллекта. Мы рассмотрим как эти концепции и технологии взаимодействуют между собой и как их совместное использование может значительно улучшить процессы обработки и анализа изображений, открывая новые возможности и перспективы в различных областях применения.

1 ВВЕДЕНИЕ В ОБРАБОТКУ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

1.1 Описание нейронной сети и машинного зрения

Нейронные сети – это математические модели, вдохновленные структурой и работой человеческого мозга и используемые для обработки информации [1]. Они являются ключевым компонентом машинного обучения и искусственного интеллекта.

Нейронные сети состоят из соединенных взаимосвязанных элементов, называемых нейронами, которые работают параллельно для решения сложных задач. В основе нейронных сетей лежит попытка сымитировать работу нейронов в человеческом мозге. Каждый нейрон в нейронной сети принимает входные данные, обрабатывает их и передает результат следующему нейрону в сети. Соединения между нейронами называются весами. Они играют ключевую роль в определении важности входных данных.

Процесс обучения нейронной сети заключается в подстройке весов таким образом, чтобы сеть могла правильно отвечать на входящие запросы. Для этого используется обучающая выборка, которая содержит пары входных данных и соответствующих эталонных выходных данных. Веса в сети корректируются в процессе обучения так, чтобы минимизировать разницу между предсказанными и эталонными значениями.

Нейронные сети проявляют выдающуюся способность к обучению на основе данных, позволяя выявлять сложные закономерности в информации там, где не справляются традиционные методы программирования.

Методы машинного зрения, в свою очередь, направлены на обработку визуальной информации, такой как изображения и видео. Эти методы включают в себя различные техники, такие как обнаружение объектов, сегментация и распознавание изображений, анализ движения и многие другие [2]. Машинное зрение – это способность компьютеров извлекать информацию и смысл из изображений и видео. С помощью нейронных сетей компьютеры могут различать и распознавать изображения так, как это делают люди.

Сочетание нейронных сетей с методами машинного зрения позволяет создавать сложные системы, такие как распознавание объектов в видеопотоке и их автоматический анализ. Например, нейронные сети могут обучаться на изображениях, а методы машинного зрения могут помочь выделять и анализировать различные объекты на этих изображениях.

Искусственные нейронные сети можно классифицировать по тому, как данные передаются от входного узла к выходному узлу [3]. Ниже приведены несколько примеров.

1. Нейронные сети прямого распространения. Нейронные сети прямого распространения обрабатывают данные в одном направлении, от входного узла к выходному узлу. Каждый узел одного слоя связан с каждым узлом

следующего слоя. Нейронные сети прямого распространения используют процесс обратной связи для улучшения прогнозов с течением времени.

2. Алгоритм обратного распространения. Искусственные нейронные сети постоянно обучаются, используя корректирующие циклы обратной связи для улучшения своей прогностической аналитики. Проще говоря, речь идет о данных, протекающих от входного узла к выходному узлу по множеству различных путей в нейронной сети. Правильным является только один путь, который сопоставляет входной узел с правильным выходным узлом. Чтобы найти этот путь, нейронная сеть использует петлю обратной связи, которая работает следующим образом:

Шаг 1: Каждый узел делает предположение о следующем узле на пути.

Шаг 2: Он проверяет, является ли предположение правильным. Узлы присваивают более высокие значения веса путям, которые приводят к более правильным предположениям, и более низкие значения веса путям узлов, которые приводят к неправильным предположениям.

Шаг 3: Для следующей точки данных узлы делают новый прогноз, используя пути с более высоким весом, а затем повторяют шаг 1.

3. Свёрточные нейронные сети. Скрытые слои в свёрточных нейронных сетях выполняют определенные математические функции (например, суммирование или фильтрацию), называемые свёртками. Они очень полезны для классификации изображений, поскольку могут извлекать из них соответствующие признаки, полезные для распознавания и классификации. Новую форму легче обрабатывать без потери функций, которые имеют решающее значение для правильного предположения. Каждый скрытый слой извлекает и обрабатывает различные характеристики изображения: границы, цвет и глубину.

Обучение нейронной сети – это процесс обучения нейронной сети выполнению задачи [4]. Нейронные сети обучаются путем первичной обработки нескольких больших наборов размеченных или неразмеченных данных. На основе этих примеров сети могут более точно обрабатывать неизвестные входные данные.

При контролируемом обучении специалисты по работе с данными предлагают искусственным нейронным сетям помеченные наборы данных, которые заранее дают правильный ответ. Например, сеть глубокого обучения, обучающаяся распознаванию лиц, обрабатывает сотни тысяч изображений человеческих лиц с различными терминами, связанными с этническим происхождением, страной или эмоциями, описывающими каждое изображение. Нейронная сеть медленно накапливает знания из этих наборов данных, которые заранее дают правильный ответ. После обучения сеть начинает делать предположения об этническом происхождении или эмоциях нового изображения человеческого лица, которое она никогда раньше не обрабатывала.

Глубокое обучение – это разновидность машинного обучения, в котором для обработки данных используются сети глубокого обучения [5].

Традиционные методы машинного обучения требуют участия человека, чтобы программное обеспечение работало должным образом. Специалист по работе с данными вручную определяет набор соответствующих функций, которые должно анализировать программное обеспечение. Это ограничение делает создание и управление программным обеспечением утомительным и трудозатратным процессом.

С другой стороны, при глубоком обучении специалист по работе с данными предоставляет программному обеспечению только необработанные данные. Сеть глубокого обучения извлекает функции самостоятельно и обучается более независимо. Она может анализировать неструктурированные наборы данных (например, текстовые документы), определять приоритеты атрибутов данных и решать более сложные задачи. Например, при обучении программного обеспечения с алгоритмами машинного обучения правильно идентифицировать изображение домашнего животного вам потребуется выполнить следующие шаги:

– Найти и вручную отметить тысячи изображений домашних животных: кошек, собак, лошадей, хомяков, попугаев и т. д.

– Сообщить программному обеспечению с алгоритмами машинного обучения, какие функции необходимо найти, чтобы оно могло идентифицировать изображение методом исключения. Например, оно может подсчитать количество ног, а затем проверить форму глаз, ушей, хвоста, цвет меха и так далее.

– Вручную оценить и изменить помеченные наборы данных, чтобы повысить точность программного обеспечения. Например, если в вашем тренировочном наборе слишком много изображений черных кошек, программное обеспечение правильно определит черную кошку, но не белую.

При глубоком обучении нейронные сети будут обрабатывать все изображения и автоматически определять, что сначала им требуется проанализировать количество ног и форму головы, а уже после посмотреть на хвосты, чтобы правильно идентифицировать животное на изображении.

1.2 Структура свёрточной нейронной сети

Свёрточные нейронные сети (Convolutional Neural Networks, CNN) – особые типы нейронных сетей, которые помогают компьютерам видеть и понимать изображения и видео. Такие сети имеют несколько слоев, называемых свёрточными. Они позволяют CNN изучать сложные особенности и делать более точные предсказания о содержимом визуальных материалов.

Каждый слой такой сети обрабатывает данные и направляет выявленные особенности следующему слою для дальнейшей обработки. В них используются фильтры, которые помогают выделить важные особенности, например края или формы объектов на изображении. Когда к визуальному материалу применяются фильтры, мы получаем свернутое изображение. Затем CNN его анализирует и выявляет важные особенности. Этот процесс

называется извлечением признаков. Помимо свёрточных слоев, CNN включают [6]:

− слои пулинга, которые уменьшают размер изображения, чтобы сеть могла работать быстрее и лучше обобщать данные;

− слои нормализации, которые помогают предотвратить переобучение и улучшить производительность сети;

− полносвязные слои, которые используются для классификации.

Свёрточные нейронные сети работают следующим образом [6]:

− входные данные, такие как изображения или видео, поступают на входной слой;

− свёрточные слои извлекают различные признаки из входных данных (используются фильтры для обнаружения границ, форм, текстур и других характеристик);

− после каждого свёрточного слоя применяется функция активации ReLU, которая добавляет нелинейность и помогает улучшить производительность сети;

− далее следует слой пулинга, который уменьшает размерность карт признаков, выбирая наиболее важные значения из каждой области;

− полносвязные слои принимают выходные данные из слоя пулинга и используют набор весов для классификации или предсказания – они объединяют выделенные признаки и принимают окончательное решение.

Свёрточная нейронная сеть не имеет обратных связей и является многослойной. В большинстве случаев, в её обучении применяется метод обучения с учителем. Данная структура получила название из-за наличия так называемой операции свёртки над парой матриц A (размера $nx \times ny$) и B (размера $mx \times my$), результатом которой является матрица $C = A \times B$ размера $(nx - mx + 1) \times (ny - my + 1)$.

Работа данной модели обычно описывается как переход от определённых особенностей изображения к их абстрактным представлениям, а затем к ещё более абстрактным деталям вплоть до выявления представлений высокого уровня. При этом нейронная сеть изменяет свои настройки и вырабатывает необходимую ротацию абстрактных признаков, отбрасывая маловажные детали и выделяя существенное [7]. Чаще всего в свёрточных нейронных сетях используется техники Padding и Striding.

Padding добавляет к краям пиксели нулевого значения. Таким образом, ядро позволяет ненулевым пикселям оказываться в своем центре, а затем распространяется на нулевые пиксели за пределами края, создавая выходную матрицу того же размера, что и входная.

Striding же пропускает некоторые области, над которыми скользит ядро, в зависимости от заданного шага. Шаг означает, что берутся пролеты через пиксель, то есть по факту каждый пролет является стандартной свёрткой, и чем выше шаг, тем выше сжатие.

Свёрточная нейронная сеть состоит из большого количества слоёв. Сначала идёт слой входного изображения, затем сигнал передаётся через серию свёрточных слоёв. Чередование таких слоёв даёт возможность группировать карты признаков, а на каждом следующем слое карта уменьшается в размере, однако количество каналов увеличивается. На практике это повышает способности к распознаванию сложных критериев признаков. На выходе свёрточных слоёв сети дополнительно ставят слои полносвязной нейронной сети, на вход которому подаётся результат работы свёрточных слоёв (конечные карты признаков) (см. рисунок 1.5).

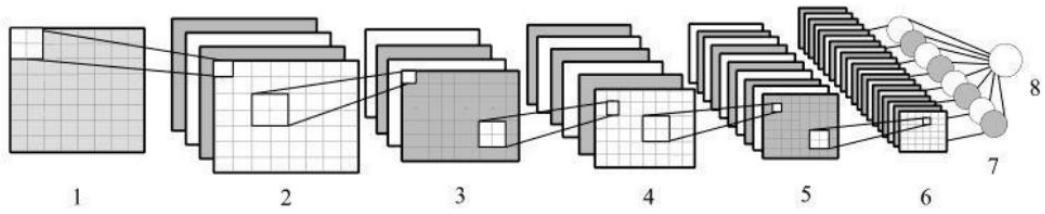

Рисунок 1.1 – Архитектура свёрточной нейронной сети: *1* – вход; *2,4,6* – свёрточные слои; *3,5* – подвыборочные слои; *7* – слои из обычных нейронов; *8* – выход

1.3 Применение нейронных сетей для обработки изображений

Нейросети применяются для обработки изображений с целью улучшения их качества, реставрации, увеличения разрешения и удаления шума [8]. С помощью генеративных нейронных сетей (GAN) можно создавать реалистичные изображения, что находит применение в таких областях, как графический дизайн, искусство и медицинская визуализация.

Свёрточные нейронные сети (CNN) являются основным инструментом для распознавания объектов на изображениях. Они обучаются классифицировать объекты, людей, животных, транспортные средства и другие элементы на фотографиях или видео. Распознавание объектов на изображениях находит широкое применение в автономных транспортных средствах, системах безопасности, медицинской диагностике и других областях.

Нейросети применяются для детектирования и сегментации объектов на изображениях. Детектирование позволяет определить присутствие объектов и их положение, а сегментация – выделить каждый объект в отдельности на изображении. Эти техники широко используются в робототехнике, медицинской диагностике и системах видеонаблюдения.

Нейросети искусственного интеллекта играют ключевую роль в развитии компьютерного зрения. Благодаря этой технологии, компьютеры могут анализировать, понимать и интерпретировать содержание изображений также, как это делает человек. Это имеет множество применений, от автоматического описания изображений до помощи слабовидящим людям.

Преимущества применения нейросетей в обработке изображений и компьютерном зрении:

– Высокая точность: Нейросети обладают высокой точностью в распознавании объектов и обработке изображений, что делает их более эффективными во многих задачах.

– Автоматизация: Нейросети позволяют автоматизировать процессы обработки изображений, что экономит время и ресурсы.

– Адаптивность: Нейросети обучаются на основе больших объемов данных и способны адаптироваться к новым условиям и задачам.

– Расширение возможностей: Использование нейросетей открывает новые возможности в обработке изображений и развитии компьютерного зрения.

Нейронная сеть для распознавания изображений – это, пожалуй, наиболее популярный способ применения нейронной сети. При этом вне зависимости от особенностей решаемых задач, она работает по этапам [9]. В целом создание нейронной сети для распознавания изображений включает в себя шаги, наиболее важные среди которых представленны на рисунке 1.1.

Рисунок 1.1 – Шаги, которые включает в себя создание нейронной сети распознавания изображений

В качестве распознаваемых образов могут выступать самые разные объекты, включая изображения, рукописный или печатный текст, звуки и многое другое. При обучении сети ей предлагаются различные образцы с меткой того, к какому именно типу их можно отнести. В качестве образца применяется вектор значений признаков, а совокупность признаков в этих

условиях должна позволить однозначно определить, с каким классом образов имеет дело нейронная сеть.

Важно при обучении научить сеть определять не только достаточное количество и значения признаков, чтобы выдавать хорошую точность на новых изображениях, но и не переобучиться, то есть, излишне не «подстроиться» под обучающую выборку из изображений. После завершения правильного обучения нейронная сеть должна уметь определять образы (тех же классов), с которыми она не имела дела в процессе обучения. Необходимо учитывать, что исходные данные для нейросети должны быть однозначны и непротиворечивы, чтобы не возникали ситуации, когда нейронная сеть будет выдавать высокие вероятности принадлежности одного объекта к нескольким классам.

Несмотря на значительные преимущества, работа нейронных сетей и методов компьютерного зрения часто сопряжена с рядом трудностей, которые могут усложнить их внедрение и использование [10]. Давайте рассмотрим некоторые из основных проблем:

1. Обучение на большом количестве данных. Нейронные сети нуждаются в большом количестве данных для обучения, и без достаточного количества данных они могут быть неправильно обучены и давать неточные результаты. Решение этой проблемы заключается в использовании техник обучения на малых данных, таких как трансферное обучение и генеративные модели.

2. Необходимость больших вычислительных ресурсов. Обучение нейронных сетей требует больших вычислительных ресурсов, включая высокопроизводительные компьютеры и графические процессоры (GPU). Решением этой проблемы может быть использование облачных сервисов, таких как Amazon Web Services (AWS) и Google Cloud Platform (GCP), которые предоставляют готовые вычислительные ресурсы.

3. Чувствительность к выбросам. Нейронные сети могут быть чувствительны к выбросам в данных, что может привести к неточным результатам. Решение этой проблемы заключается в использовании методов для обработки и чистки данных, таких как сжатие данных и фильтрация выбросов.

4. Неинтерпретируемость. Нейронные сети могут быть трудны для понимания и интерпретации, что может быть проблемой при принятии решений на основе результатов. Решение этой проблемы может быть связано с использованием методов визуализации данных и внедрения обратных связей в модели.

5. Проблемы с обобщением. Нейронные сети могут иметь проблемы с обобщением, то есть они могут быть склонны к переобучению на тренировочных данных и не справляться с новыми данными. Решение этой проблемы заключается в использовании техник, таких как регуляризация, ансамблирование и дропаут.

6. Трудность с обработкой неструктурированных данных. Нейронные сети могут иметь трудности с обработкой неструктурированных данных, таких как изображения, звук и текст. Решение этой проблемы заключается в использовании специальных архитектур нейронных сетей, которые были специально разработаны для обработки неструктурированных данных, таких как свёрточные нейронные сети для обработки изображений и рекуррентные нейронные сети для обработки текстовых данных.

Недостатки интерпретируемости и объяснимости нейронных сетей также являются значимыми проблемами в областях, где необходимо объяснять, как был сделан определенный вывод, например, в медицине и юриспруденции. В таких случаях могут использоваться методы интерпретируемости, которые позволяют объяснить, как работает модель и какие признаки были использованы для сделанного вывода.

Еще одна проблема, связанная с нейронными сетями, заключается в том, что они могут быть подвержены атакам, таким как внедрение шума в данные или изменение входных параметров, что может привести к неправильным выводам. Решение этой проблемы может быть связано с использованием методов защиты от атак, таких как дополнительные слои обнаружения аномалий или обработки входных данных.

Таким образом, несмотря на то, что нейронные сети являются мощным инструментом для многих задач, они также имеют свои ограничения и проблемы, которые могут быть решены с помощью использования специальных методов и техник.

2 GRID СИСТЕМА В КОНТЕКСТЕ ВЫСОКОНАГРУЖЕННЫХ СЕРВИСОВ ДЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ

2.1 Описание GRID системы

Распределенная (GRID) система – это набор компьютерных программ, использующих вычислительные ресурсы нескольких отдельных вычислительных узлов для достижения одной общей цели. Ее также называют распределенными вычислениями или распределенной базой данных. Распределенная система основывается на отдельных узлах, которые обмениваются данными и выполняют синхронизацию в общей сети. Обычно узлы представляют собой отдельные физические аппаратные устройства, но это могут быть и отдельные программные процессы или другие рекурсивные инкапсулированные системы. Распределенные системы направлены на устранение узких мест или единых точек отказа в системе.

Распределенные вычислительные системы обладают следующими характеристиками [11]:

1. Совместное использование ресурсов – в распределенной системе могут совместно использоваться оборудование, программное обеспечение или данные.
2. Параллельная обработка – одну и ту же функцию могут одновременно обрабатывать несколько машин.
3. Масштабируемость – вычислительная мощность и производительность могут масштабироваться по мере необходимости при добавлении дополнительных машин.
4. Обнаружение ошибок – упрощается обнаружение отказов.
5. Прозрачность – узел может обращаться к другим узлам в системе и обмениваться с ними данными.

В централизованной вычислительной системе все вычисления выполняются на одном компьютере и в одном месте. Основное различие между централизованными и распределенными системами заключается в модели взаимодействия между узлами системы. Состояние централизованной системы хранится в центральном узле, к которому индивидуально обращаются клиенты (см. рисунок 3.1). Поскольку все узлы централизованной системы обращаются к центральному узлу, это может привести к перегрузке сети и замедлить ее работу. Централизованная система имеет единую точку отказа, тогда как в распределенной системе такой точки нет.

GRID-системы содержат такие компоненты [12]:

– средства, которые транслируют запросы пользователей в запросы к ресурсам GRID-системы (компьютеров, сетей, дискового пространства, баз данных и тому подобное);

– средства, которые выполняют поиск ресурсов, их подбор и размещение, планирование и координацию вычислительных задач, а также сбор результатов;

– средства безопасности, которые дают возможность руководить аутентификацией и авторизацией пользователей. Они должны поддерживать единственный вход в систему, отображение на механизмы защиты локальных систем, возможность запуска применений с правами пользователя;

– средства разработки применений, которые используют особенности GRID-архитектуры.

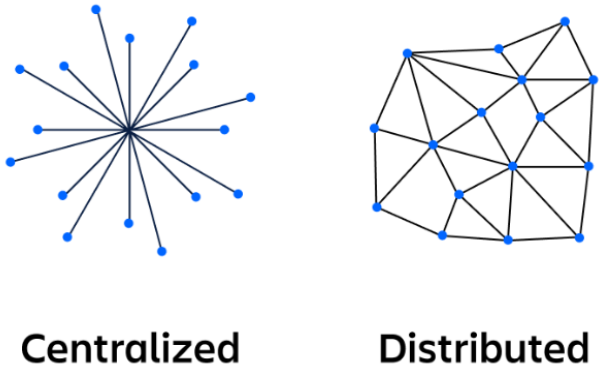

Рисунок 2.1 – Структура централизованной и распределенной систем

Распределенные системы часто помогают повысить надежность и производительность системы. Надежность повышается за счет устранения единых точек отказа и узких мест. Узлы распределенной системы обеспечивают избыточность, поэтому при отказе любого узла его могут заменить другие. Производительность повышается благодаря тому, что узлы можно легко масштабировать по горизонтали и вертикали. В случае большой нагрузки на систему можно добавить дополнительные узлы, которые помогут с ней справиться. Для обработки большой нагрузки можно также повысить производительность отдельных узлов.

Однако обратной стороной этих преимуществ может стать разрастание системы, когда система становится слишком сложной и ее техническое обслуживание затрудняется. По мере усложнения системы у команды могут возникать трудности с эффективным управлением, организацией и совершенствованием системы. В частности, может возникнуть проблема с пониманием того, как связаны между собой различные компоненты и кто является владельцем конкретного программного компонента. Бывает трудно понять, как внести изменения в компоненты, чтобы максимально повысить работоспособность и при этом избежать негативного воздействия на зависимые компоненты и клиентов.

Существует множество типов распределенных систем. Наиболее распространенные из них [13]:

– Клиент-сервер. В клиент-серверной архитектуре ответственность делится на две части. Клиент отвечает за представление (интерфейс пользователя) и поддерживает связь с сервером по сети. Сервер отвечает за

обработку бизнес-логики и управление состоянием. Клиент-серверную архитектуру можно легко превратить в централизованную, если у сервера отсутствует избыточность. По-настоящему распределенная клиент-серверная модель должна иметь несколько серверных узлов для распределения клиентских подключений. Большинство современных клиент-серверных архитектур – это клиенты, которые подключаются к инкапсулированной распределенной системе на сервере.

– Многоуровневая. Многоуровневая архитектура является расширением клиент-серверной архитектуры. Сервер в многоуровневой архитектуре разбивается на более мелкие узлы, которые выполняют дополнительные обязанности внутреннего сервера, такие как обработка данных и управление данными. Эти дополнительные узлы используются для асинхронного выполнения длительных заданий, высвобождая остальные серверные узлы для обработки запросов клиентов и взаимодействия с хранилищем данных.

– Одноранговая. Каждый узел одноранговой распределенной системы содержит полный экземпляр приложения. Отсутствует разделение на узлы представления и узлы обработки данных. Узел содержит уровень представления и уровни обработки данных. Одноранговые узлы могут содержать полные данные о состоянии всей системы.

Преимуществом одноранговой системы является ее огромная избыточность. После инициализации и подключения однорангового узла к сети он находит другие одноранговые узлы, устанавливает с ними связь и синхронизирует свое локальное состояние с состоянием всей системы. Эта особенность означает, что отказ одного узла в одноранговой системе не нарушит работу других узлов. Одноранговая система продолжит работать.

– Сервис-ориентированная архитектура. Сервис-ориентированная архитектура (SOA) является предшественницей микросервисной. Основное различие между SOA и микросервисной архитектурой заключается в области действия узлов: область действия микросервисных узлов относится к уровню функций. В микросервисной архитектуре узел инкапсулирует бизнес-логику для выполнения определенного набора функций, например обработки платежей. Микросервисная архитектура имеет несколько различных узлов бизнес-логики, которые взаимодействуют с независимыми узлами баз данных. Для сравнения, узлы SOA инкапсулируют целое приложение или подразделение компании. Границы сервиса для узлов SOA обычно охватывают всю систему баз данных в узле.

2.2 Связь между GRID системой и микросервисной архитектурой

Микросервисная архитектура – это лишь один из типов распределенных систем. В ней приложение разбивается на отдельные компоненты, или «сервисы». К примеру, микросервисная архитектура может иметь сервисы, соответствующие бизнес-функциям (платежи, пользователи, продукты и др.), и каждый компонент будет обрабатывать бизнес-логику в своей сфере

ответственности. В этом случае в системе будет несколько резервных копий сервисов, и у сервисов не будет единой точки отказа.

Микросервисная архитектура стала более популярной альтернативой SOA благодаря своим преимуществам. Микросервисы проще компоновать, поэтому команды могут многократно использовать функциональные возможности, предоставляемые небольшими сервисными узлами. Микросервисная архитектура более устойчива и допускает динамическое вертикальное и горизонтальное масштабирование.

Сегодня распределенные системы используются многими приложениями. Мобильные и веб-приложения с интенсивным трафиком являются распределенными системами. Пользователи подключаются по принципу «клиент-сервер», где клиентом является веб-браузер или мобильное приложение. В этом случае сервер представляет собой распределенную систему. Современные веб-серверы следуют модели многоуровневой системы. Для делегирования запросов множеству серверных логических узлов, которые взаимодействуют через системы очередей сообщений, используется балансировщик нагрузки.

Популярным инструментом для работы с распределенными системами является платформа Kubernetes [14], которая позволяет создать распределенную систему из набора контейнеров. Контейнеры образуют узлы распределенной системы, а Kubernetes оркеструет сетевое взаимодействие между узлами и выполняет динамическое горизонтальное и вертикальное масштабирование узлов в системе.

Распределенная трассировка – это способ профилирования или мониторинга результата запроса, который выполняется в распределенной системе [15]. Мониторинг распределенной системы может оказаться непростой задачей, поскольку каждый отдельный узел имеет собственный, отдельный поток журналов и показателей. Чтобы получить точное представление о распределенной системе, необходимо свести показатели отдельных узлов в единую картину.

Запросы к распределенным системам обычно затрагивают не весь набор узлов в системе, а лишь его часть, или путь через узлы. Распределенная трассировка выявляет часто используемые пути в распределенной системе и позволяет командам анализировать и контролировать их. Распределенная трассировка ведется в каждом узле системы, поэтому команды могут запрашивать у системы информацию о работоспособности узлов и выполнении запросов.

3 ОСОБЕННОСТИ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

3.1 Описание микросервисной архитектуры

Еще недавно программные приложения создавались преимущественно на базе монолитной архитектуры, где каждое приложение представляет собой самостоятельную единицу. Такой подход приносил пользу многим командам разработчиков до тех пор, пока приложения не становились слишком сложными. Чтобы изменить небольшой участок кода в монолитной системе, приходится заново собирать всю систему, тестировать ее и развертывать новую версию приложения. Затем появились микросервисы. С ними программные системы разбиваются на небольшие элементы, которые можно разрабатывать и развертывать независимо друг от друга. Микросервисная архитектура развивалась при поддержке приверженцев DevOps, которым требовалась быстрая поставка обновлений – новых возможностей, исправлений багов и улучшений безопасности. Кроме того, с такой архитектурой многие компании могли переписать устаревшие приложения с использованием современных языков программирования и обновленного стека технологий.

Микросервисная архитектура (или просто «микросервисы») – это подход к созданию приложения в виде набора независимо развертываемых сервисов, которые являются децентрализованными и разрабатываются независимо друг от друга [16]. Эти сервисы слабо связаны, независимо развертываются и легко обслуживаются. Монолитное приложение создается как единое и неделимое целое, тогда как в микросервисной архитектуре его разбивают на множество независимых модулей, каждый из которых вносит свой вклад в общее дело. Микросервисы неразрывно связаны с DevOps, поскольку лежат в основе методики непрерывной поставки [17], благодаря которой команды могут быстро адаптироваться к требованиям пользователей. Микросервисная архитектура относится к распределенным системам и воплощает паттерны сильногосцепления (High Cohesion) и слабого связывания (Low Coupling).

Микросервис – это веб-сервис, отвечающий за один элемент логики в определенной предметной области. Приложение создают как комбинацию микросервисов, каждый из которых предоставляет функциональные возможности в своей предметной области. Микросервисы взаимодействуют друг с другом посредством различных сетевых запросов через API-интерфейсы, такие как REST, gRPC или Kafka, но не обладают информацией о внутреннем устройстве других сервисов. Такое согласованное взаимодействие между микросервисами называется микросервисной архитектурой. Каждый микросервис может быть написан на разных языках программирования и использовать различные технологии. На рисунке 2.1 представлена структура микросервисной и монолитной архитектуры.

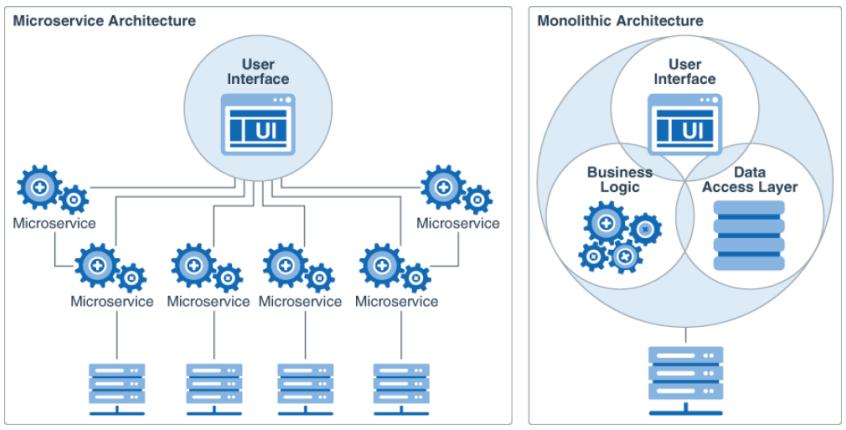

Рисунок 3.1 – Структура микросервисной и монолитной архитектуры

Микросервисная архитектура получила распространение, когда крупным компаниям потребовался более точечный подход к разработке отдельных узлов. Нужно было наращивать отказоустойчивость, но оказалось, что традиционные монолитные IT-системы тяжело масштабировать.

3.2 Ключевые особенности микросервисной архитектуры

К ключевым особенностям микросервисной архитектуры можно отнести [18]:

– Масштабируемость – микросервисы могут быть масштабированы независимо друг от друга. Это позволяет распределять нагрузку и ресурсы по сервисам, которые нуждаются в поддержке прямо сейчас.

– Независимость – каждый сервис полностью автономен и не затрагивает работу соседей. Это означает, что каждый сервис может быть разработан, развернут и обновлен отдельно, без воздействия на остальные компоненты.

– Легковесность – микросервисы используют легковесные протоколы для взаимодействия между собой, такие как REST или gRPC. Это позволяет сервисам быстро обмениваться данными.

– Гибкость – Микросервисы могут быть разработаны с использованием разных технологий и языков программирования. Это дает разработчикам большую гибкость при выборе технологий для конкретных компонентов системы. Это также позволяет эффективнее управлять командой разработки.

– Управление ошибками – у каждого сервиса есть свое собственное управление ошибками и восстановлением после сбоев. Если один сервис не отвечает, это не приводит к полной остановке системы. Например, сайт продолжит принимать покупки, если микросервис, отвечающий за логистику, будет какое-то время недоступен.

– Легкость развертывания – микросервисы могут быть легко развернуты на различных серверах или облачных платформах.

– Распределенная разработка – при построении микросервисной архитектуры разработка приложения может быть распределена между несколькими командами. Каждая команда может работать над отдельным сервисом, что ускоряет разработку.

– Легкая замена – если требуется заменить один сервис, его можно легко заменить, не затрагивая другие сервисы. Это упрощает обновление системы и добавление новых функций.

Несмотря на многочисленные преимущества, микросервисная архитектура также имеет ряд особенностей, которые затрудняют интеграцию такого подхода.

К сложностям работы с микросервисами относят [19]:

– Управление – микросервисная архитектура предполагает работу с большим количеством сервисов, каждый из которых имеет собственную версию и набор зависимостей.

– Комплексность взаимодействия – микросервисы взаимодействуют друг с другом посредством сетевых запросов. Это может привести к проблемам с производительностью и надежностью, так как каждый сетевой запрос может стать точкой отказа.

– Обеспечение целостности данных – при использовании микросервисной архитектуры данные могут храниться и обрабатываться разными сервисами. Обеспечение целостности данных и синхронизация между сервисами может быть сложной задачей. Например, в финтехе или других сферах, завязанных на быстродействие системы, синхронизировать тысячи транзакций – действительно непростая задача.

– Сложность отладки и тестирования – в микросервисной архитектуре каждый сервис может быть разработан, развернут и масштабирован независимо. Это может затруднять процесс отладки и тестирования, так как необходимо изолировать проблему до конкретного сервиса.

– Уязвимости безопасности – поскольку каждый сервис имеет доступ к части данных, уязвимость в одном сервисе может привести к компрометации всей системы.

Необходимо учитывать эти риски при проектировании и разработке микросервисной архитектуры, чтобы минимизировать их влияние на систему.

3.3 Технологии и паттерны микросервисной архитектуры

Для создания качественной микросервисной архитектуры необходимо четко разделить функции в приложении и команде. Так можно достичь слабого связывания (REST-интерфейсы) и сильного сцепления (множество сервисов могут компоноваться вместе, определяя более высокоуровневые сервисы или приложение). Существует несколько распространенных паттернов проектирования при работе с микросервисами [20].

Первый и, пожалуй, наиболее распространенный паттерн проектирования при работе с микросервисами – «агрегатор» (Aggregator).

В простейшей форме агрегатор представляет собой обычную веб-страницу, вызывающую множество сервисов для реализации функционала, требуемого в приложении (см. рисунок 2.2). Поскольку все сервисы (A, B и C) предоставляются при помощи легковесного REST-механизма, веб-страница может извлечь данные и обработать/отобразить их как необходимо. Если требуется какая-либо обработка, например, применить бизнес-логику к данным, полученным от отдельных сервисов, то для этого у может использоваться CDI-компонент, преобразующий данные таким образом, чтобы их можно было вывести на веб-странице.

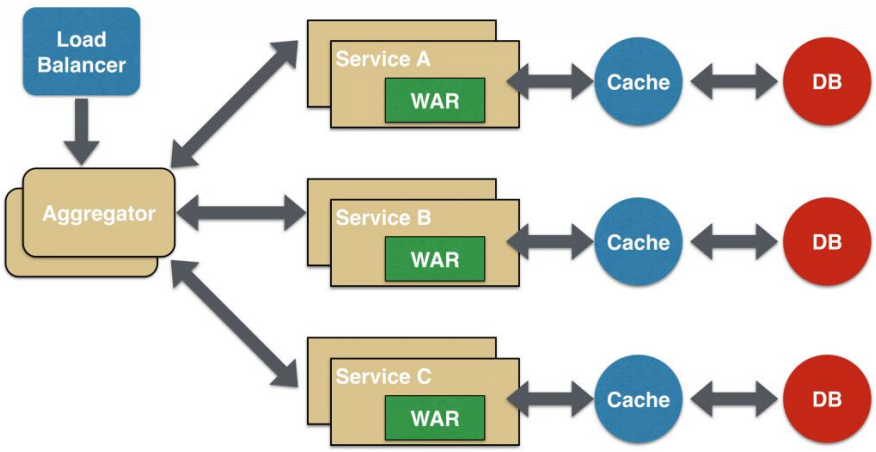

Рисунок 3.2 – Схемотическое представление паттерна «агрегатор» (Aggregator)

Агрегатор может использоваться и в тех случаях, когда не требуется ничего отображать, а нужен лишь более высокоуровневый составной микросервис, который могут потреблять другие сервисы. В данном случае агрегатор просто соберет данные от всех отдельных микросервисов, применит к ним бизнес-логику, а далее опубликует микросервис как конечную точку REST. В таком случае, при необходимости, его смогут потреблять другие нуждающиеся в нем сервисы.

Этот паттерн следует принципу DRY. Если существует множество сервисов, которые должны обращаться к сервисам A, B и C, то рекомендуется абстрагировать эту логику в составной микросервис и агрегировать ее в виде отдельного сервиса. Преимущество абстрагирования на этом уровне заключается в том, что отдельные сервисы, например, A, B и C, могут развиваться независимо, а бизнес-логику будет по-прежнему выполнять составной микросервис. Агрегатор также может независимо масштабироваться как по горизонтали, так и по вертикали.

Паттерн «посредник» (Proxy) при работе с микросервисами – это вариант агрегатора. В таком случае агрегация должна происходить на клиенте, но в зависимости от бизнес-требований при этом может вызываться дополнительный микросервис (см. рисунок 2.3).

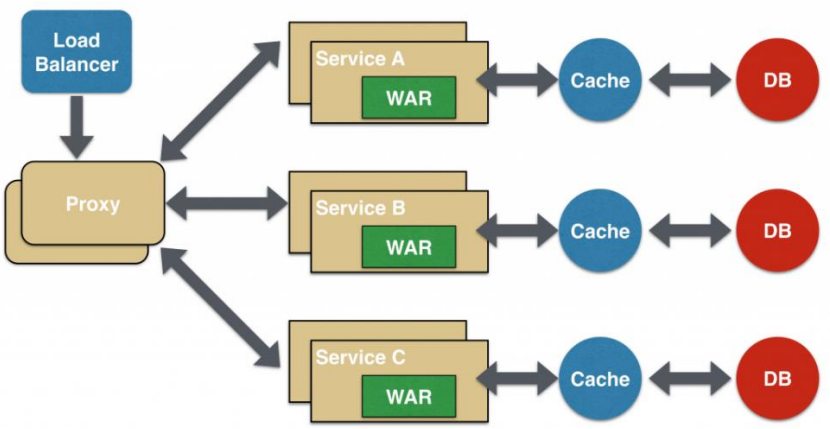

Рисунок 3.3 – Схемотическое представление паттерна «посредник» (Proxy)

Как и агрегатор, посредник может независимо масштабироваться по горизонтали и по вертикали. Это может понадобиться в ситуации, когда каждый отдельный сервис нужно не предоставлять потребителю, а запускать через интерфейс.

Посредник может быть формальным (dumb), в таком случае он просто делегирует запрос одному из сервисов. Он может быть и интеллектуальным (smart), в таком случае данные перед отправкой клиенту подвергаются тем или иным преобразованиям. Например, уровень представления для различных устройств может быть инкапсулирован в интеллектуальный посредник.

Микросервисный паттерн проектирования «цепочка» (Chaine) выдает единый консолидированный ответ на запрос. В данном случае сервис A получает запрос от клиента, связывается с сервисом B, который, в свою очередь, может связаться с сервисом C (см. рисунок 2.4). Все эти сервисы, скорее всего, будут обмениваться синхронными сообщениями «запрос/отклик» по протоколу HTTP.

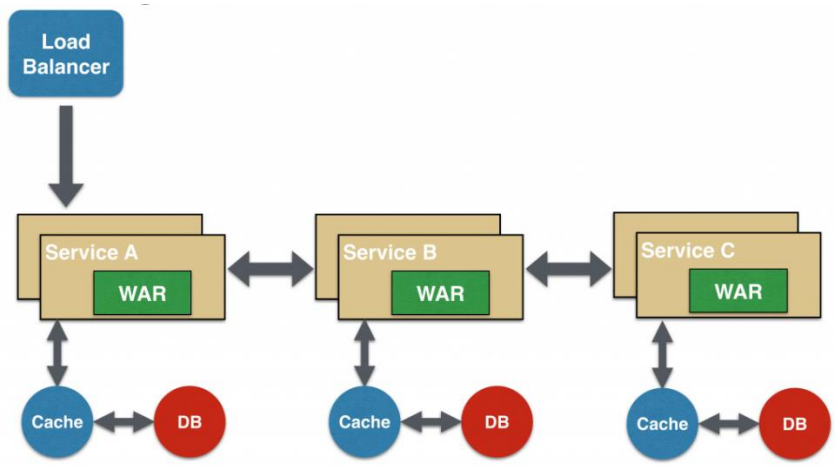

Рисунок 3.4 – Схемотическое представление паттерна «цепочка» (Chaine)

Здесь важнее всего запомнить, что клиент блокируется до тех пор, пока не выполнится вся коммуникационная цепочка запросов и откликов, т.е.

Service ↔ Service B и Service B ↔ Service C. Запрос от Service B к Service C может выглядеть совершенно иначе, нежели от Service A к Service B. Аналогично, отклик от Service B к Service A может принципиально отличаться от отклика Service C к Service B. Это наиболее важно во всех случаях, когда бизнес-ценность нескольких сервисов суммируется.

Здесь также важно понять, что нельзя делать цепочку слишком длинной. Это критично, поскольку цепочка синхронна по своей природе, и чем она длиннее, тем дольше придется ожидать клиенту, особенно если отклик заключается в выводе веб-страницы на экран. Существуют способы обойти такой блокирующий механизм запросов и откликов, и они рассматриваются в следующем паттерне. Цепочка, состоящая из единственного микросервиса, называется «цепочка-одиночка». Впоследствии ее можно расширить.

Микросервисный паттерн проектирования «ветка» (Branch) расширяет паттерн «агрегатор» и обеспечивает одновременную обработку откликов от двух цепочек микросервисов, которые могут быть взаимоисключающими (см. рисунок 2.5). Этот паттерн также может применяться для вызова различных цепочек, либо одной и той же цепочки – в зависимости от потребностей.

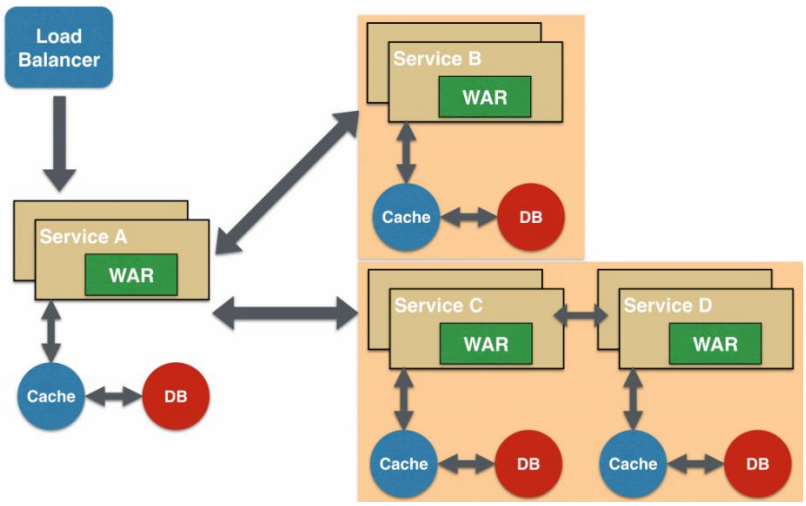

Рисунок 3.5 – Схемотическое представление паттерна «ветка» (Branch)

Сервис A, будь то веб-страница или составной микросервис, может конкурентно вызывать две различные цепочки – и в этом случае будет напоминать агрегатор. В другом случае сервис A может вызывать лишь одну цепочку в зависимости от того, какой запрос получит от клиента.

Такой механизм можно сконфигурировать, реализовав маршрутизацию конечных точек JAX-RS, в таком случае конфигурация должна быть динамической.

Один из принципов проектирования микросервисов – автономность. Это означает, что сервис полностековый и контролирует все компоненты – пользовательский интерфейс, промежуточное ПО, сохраняемость, транзакции. В таком случае сервис может быть многоязычным

и решать каждую задачу при помощи наиболее подходящих инструментов. Например, если при необходимости можно применить хранилище данных NoSQL, то лучше сделать именно так, а не забивать эту информацию в базу данных SQL.

Однако, типичная проблема, особенно при рефакторинге имеющегося монолитного приложения, связана с нормализацией базы данных – так, чтобы у каждого микросервиса был строго определенный объем информации, ни больше, ни меньше. Даже если в монолитном приложении используется только база данных SQL, ее денормализация приводит к дублированию данных, а возможно – и к несогласованности. На переходном этапе в некоторых приложениях бывает очень полезно применить паттерн «разделяемые данные» (Shared Data).

При этом паттерне несколько микросервисов могут работать о цепочке и совместно использовать хранилища кэша и базы данных. Это целесообразно лишь в случае, если между двумя сервисами существует сильная связь. Некоторые могут усматривать в этом антипаттерн, но в некоторых бизнес-ситуациях такой шаблон действительно уместен. Он определенно был бы антипаттерном в приложении, которое изначально создается как микросервисное. Кроме того, его можно рассматривать как промежуточный этап, который нужно преодолеть, пока микросервисы не станут полностью автономными.

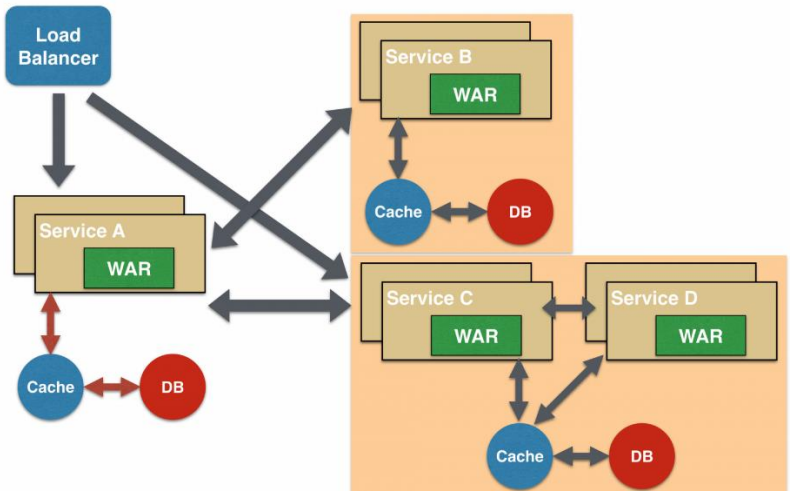

Рисунок 3.6 – Схемотическое представление паттерна «разделяемые данные» (Shared Data)

При всей распространенности и понятности паттерна REST, у него есть важное ограничение, а именно: он синхронный и, следовательно, блокирующий. Обеспечить асинхронность можно, но это делается по-своему в каждом приложении. Поэтому в некоторых микросервисных архитектурах могут использоваться очереди сообщений, а не модель REST запрос/отклик.

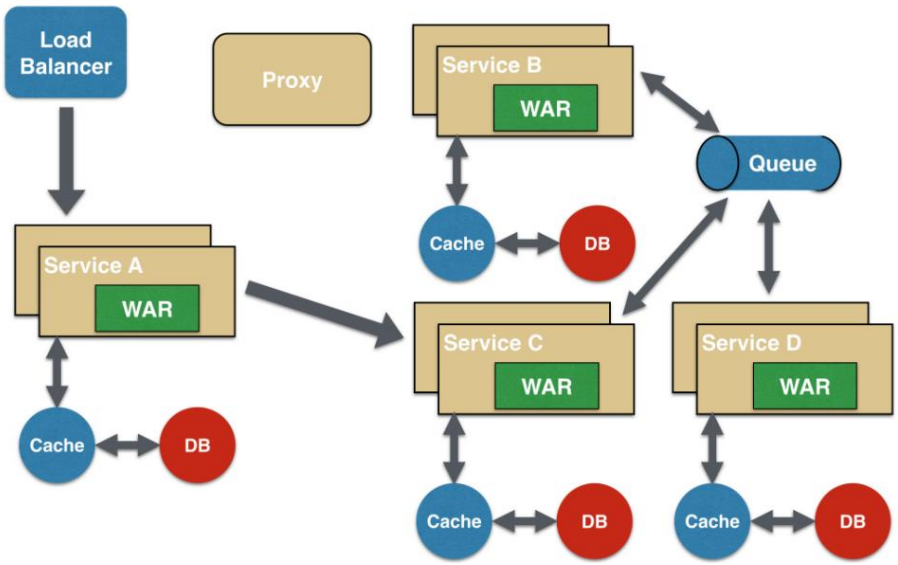

Рисунок 3.7 – Схемотическое представление паттерна «асинхронные сообщения» (Asynchronous Messaging)

В этом паттерне сервис A может синхронно вызывать сервис C, который затем будет асинхронно связываться с сервисами B и B при помощи разделяемой очереди сообщений. Коммуникация Service A → Service C может быть асинхронной, скажем, с использованием веб-сокетов; так достигается желаемая масштабируемость.

Комбинация модели REST запрос/отклик и обмена сообщениями публикатор/подписчик также могут использоваться для достижения поставленных целей.

Современный подход к разработке требует наличия платформы контейнеризации. В большинстве случаев разработчики используют для этих целей Docker. При помощи инструментов Docker они могут отделить приложение от инфраструктуры, то есть одинаково свободно работать с ним как локально, так и в облаке, что очень удобно для разработки. Контейнеризация и развертывание контейнеров – это новая модель распределенной инфраструктуры.

Когда контейнеров становится слишком много, не обойтись без оркестратора – это решение для управления и организации групп контейнеров. В качестве оркестратора чаще всего используют Kubernetes, который имеет хорошую совместимость с Docker. Однако у Docker есть и собственное решение для этих целей: Docker Swarm.

Еще один необходимый инструмент – балансировщик нагрузки, который обеспечивает равномерное распределение сетевого трафика по всем облачным ресурсам. Это заметно повышает отказоустойчивость приложения

ЗАКЛЮЧЕНИЕ

Распределенные системы получили широкое распространение и используются в большинстве современных программных интерфейсов. Приложения для социальных сетей, сервисы потокового видео, сайты электронной коммерции и многие другие продукты работают на базе распределенных систем. Централизованные системы естественным образом преобразуются в распределенные, чтобы получить возможность масштабирования. Микросервисы – это популярная и широко распространенная модель построения распределенных систем.

Важным аспектом, который следует учесть при разработке и внедрении подобных систем, является балансировка нагрузки. Балансировщики нагрузки играют ключевую роль в равномерном распределении запросов и задач между различными узлами системы, обеспечивая оптимальное использование вычислительных ресурсов и минимизацию временных задержек.

Также, важно обратить внимание на аспекты безопасности при разработке и внедрении распределенных систем. Защита данных, аутентификация пользователей и обеспечение целостности информации играют критическую роль в обеспечении устойчивости и доверия к системе. Кроме того, эффективное управление и мониторинг распределенных систем являются необходимыми компонентами для обеспечения их непрерывной работоспособности и быстрой реакции на изменяющиеся условия окружающей среды.

Таким образом, изучение и совместное применение технологий нейронных сетей, GRID систем и микросервисной архитектуры способствует созданию эффективных систем обработки изображений на основе искусственного интеллекта, способных эффективно решать сложные задачи анализа и обработки данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1]. Что такое нейронная сеть? [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/what-is/neural-network/ – Дата доступа: 26.04.2024.

[2]. Как взаимодействие нейронных сетей и методов компьютерного зрения влияет на различные сферы деятельности человека [Электронный ресурс]. – Режим доступа: https://scand.com/ru/company/blog/how-ai-and-computer-vision-techniques-enhance-image-recognition/ – Дата доступа: 26.04.2024.

[3]. 12 основных видов нейросетей [Электронный ресурс]. – Режим доступа: https://blog.ycla.ai/raznovidnosti-nejrosetej/ – Дата доступа: 27.04.2024.

[4]. Алгоритмы обучения нейронных сетей [Электронный ресурс]. – Режим доступа: https://www.etxt.ru/subscribes/algoritmy-obucheniya-neyronnykh-setey/ – Дата доступа: 27.04.2024.

[5]. Глубокое обучение: что это такое? [Электронный ресурс]. – Режим доступа: https://stfalcon.com/ru/blog/post/deep-learning-what-it-is – Дата доступа: 28.04.2024.

[6]. Сверточные нейросети: что это и для чего они нужны? [Электронный ресурс]. – Режим доступа: https://forklog.com/cryptorium/ai/svertochnye-nejroseti-chto-eto-i-dlya-chego-oni-nuzhny – Дата доступа: 01.05.2024.

[7]. Таганов А.И. Нейросетевые системы искусственного интеллекта в задачах обработки изображений – М.: Телеком, 2016. – 148 с.

[8]. Нейросети и изображения: Обработка изображений, распознавание объектов и развитие компьютерного зрения [Электронный ресурс]. – Режим доступа: https://vc.ru/u/2056024-setevye-mysli/766304-neyroseti-i-izobrazheniya-obrabotka-izobrazheniy-raspoznavanie-obektov-i-razvitie-kompyuternogo-zreniya – Дата доступа: 02.05.2024.

[9]. Нейронные сети: распознавание образов и изображений с помощью ИИ [Электронный ресурс]. – Режим доступа: https://center2m.ru/ai-recognition – Дата доступа: 02.05.2024.

[10]. Недостатки нейронных сетей: какие ограничения и проблемы есть у этой технологии и как их можно решить [Электронный ресурс]. – Режим доступа: https://vc.ru/u/22269-aleksandr-shulepov/682524-nedostatki-neyronnyh-setey-kakie-ogranicheniya-i-problemy-est-u-etoy-tehnologii-i-kak-ih-mozhno-reshit – Дата доступа: 03.05.2024.

[11]. Что такое распределенная система? [Электронный ресурс]. – Режим доступа: https://www.atlassian.com/ru/microservices/microservices-architecture/distributed-architecture – Дата доступа: 04.05.2024.

[12]. Grid-системы [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/7152240/page:14/ – Дата доступа: 04.05.2024.

[13]. Что такое распределенные вычисления? [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/what-is/distributed-computing/ – Дата доступа: 04.05.2024.

[14]. Что такое Kubernetes [Электронный ресурс]. – Режим доступа: https://kubernetes.io/ru/docs/concepts/overview/what-is-kubernetes/ – Дата доступа: 06.05.2024.

[15]. Распределенная трассировка в действии: использование OpenTracing в расследовании инцидентов разработки [Электронный ресурс]. – Режим доступа: https://www.comnews.ru/digital-economy/content/222814/2022-10-31/2022-w44/raspredelennaya-trassirovka-deystvii-ispolzovanie-opentracing-rassledovanii-incidentov-razrabotki – Дата доступа: 06.05.2024.

[16]. Microservices: understanding what it is and its benefits [Электронный ресурс]. – Режим доступа: https://www.atlassian.com/ru/microservices – Дата доступа: 10.05.2024.

[17]. Что такое непрерывная доставка? [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/devops/continuous-delivery/ – Дата доступа: 10.05.2024.

[18]. Что такое микросервисная архитектура [Электронный ресурс]. – Режим доступа: https://selectel.ru/blog/what-is-microservice-architecture/ – Дата доступа: 12.05.2024.

[19]. Монолитная vs Микросервисная архитектура [Электронный ресурс]. – Режим доступа: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16 – Дата доступа: 12.05.2024.

[20]. Микросервисные паттерны проектирования [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/piter/articles/275633/ – Дата доступа: 13.05.2024.