Программные системы и вычислительные методы
Правильная ссылка на статью:
Кирилов В.С. Обработка запросов с иерархическим характером разделяемых ресурсов // Программные системы и вычислительные методы. 2025. № 3. DOI: 10.7256/2454-0714.2025.3.72892 EDN: OHCZAE URL: https://nbpublish.com/library_read_article.php?id=72892

Обработка запросов с иерархическим характером разделяемых ресурсов

Кирилов Владимир Святославович

кандидат технических наук

доцент; кафедра Компьютерная безопасность; Северо-Кавказский федеральный университет

355045, Россия, Ставропольский край, г. Ставрополь, ул. Серова, 8, оф. 28

cnhfyysqrjl@gmail.com

Статья из рубрики "Операционные системы "

DOI:

10.7256/2454-0714.2025.3.72892

EDN:

OHCZAE

Дата направления статьи в редакцию:

27-12-2024

Аннотация: Предметом данного исследования является разработка и анализ структуры данных и алгоритма для управления параллельным выполнением сообщений в микросервисной архитектуре без брокера сообщений. В условиях перехода к микросервисной архитектуре и асинхронному обмену сообщениями, особенно при отсутствии централизованного брокера, возникает необходимость в эффективных методах обеспечения порядка обработки сообщений, влияющих на общие ресурсы. Проблема заключается в том, что традиционные методы, такие как сегментирование, не гарантируют соблюдение порядка обработки сообщений при параллельном выполнении и усложняются при необходимости синхронизации доступа к ресурсам. В качестве альтернативы традиционным подходам рассматривается метод с использованием общей очереди и пула потоков. В работе исследуется и предлагается структура данных, которая обеспечивает возможность параллельной обработки сообщений при условии отсутствия конфликтов блокировки, тем самым гарантируя корректный порядок выполнения операций, связанных с общими ресурсами, и избегая взаимных блокировок. Основная цель состоит в создании механизмов управления доступом к ресурсам,

адаптированных для микросервисной архитектуры, без усложнения логики обработки сообщений и позволяющих избежать проблем, связанных с многопоточностью. В работе используется аналитический подход к разработке структуры данных и алгоритма, основанный на формализации задачи синхронизации, а также теоретический анализ алгоритмической сложности и корректности предложенного решения. Научная новизна работы заключается в предложении новой структуры данных, использующей упорядоченные множества и списки ожидания для эффективного управления параллельной обработкой асинхронных сообщений в микросервисных архитектурах, особенно там, где отсутствует брокер сообщений. Предлагаемый алгоритм позволяет динамически определять блокировки, связанные с сообщениями, а также разделять блокирующие и неблокирующие сообщения, что обеспечивает возможность их параллельного выполнения. Предложенная структура данных и алгоритм позволяют изменять детализацию блокируемых ресурсов, не усложняя при этом процедуры обработки сообщений, а также упрощают многопоточное программирование, позволяя рассматривать каждую процедуру обработки сообщения как однопоточную. Алгоритм не имеет проблемы взаимной блокировки ресурсов, что повышает общую отказоустойчивость системы. Выявленные недостатки, связанные с блокировкой ресурсов, предлагается устранить в дальнейших исследованиях.

Ключевые слова:

микросервисы, иерархии, контекст, данные, алгоритмы, сервер, протокол, брокер сообщение, запись, сообщение

Введение

Увеличение информационной нагрузки на корпоративные приложения привело к переходу на распределенные вычисления. Первым решением данной задачи была клиент/сервер архитектура. Однако, дальнейшее развитие систем привело к необходимости декомпозиции задач на подзадачи и увеличению количества серверов. С приходом таких систем как Kibernetis, Docker и DevOps технологий, широкое распространение получила микросервесная архитектура и системы, основанные на передаче сообщений [5][6]. Сообщения играют роль параметров вызова процедур и могут быть переданы используя сетевые протоколы. За время использования данной архитектуры, появилось несколько шаблонов проектирования. В частности, широкое распространение получили микросервисы, передача сообщений в которых построена по архитектуре asynchronous request/response [1][2][3] (рисунок 1).

Особый интерес представляют системы, построенные без брокера сообщений [4] (рисунок 2). Брокер сообщений (это обычно некоторый сервер, такой как Kafka, RabitMQ, ActiveMQ, TIBCO) может быть узким местом системы, вследствие недостаточной производительности. Поэтому, системы, имеющие прямые соединения, потенциально имеют большую пропускную способность и меньшую задержку.

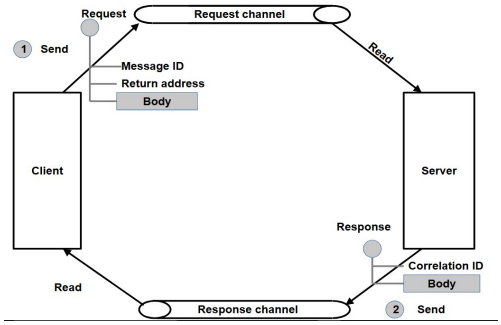

Рис. 1. Asynccgronous request/response передача сообщений.

Микросервисы использующие данный метод передачи данных обладают большой гибкостью и позволяют строить системы с большой пропускной способностью. Однако, на сообщения и способ передачи и обработки сообщений накладываются определенные ограничения. Рассмотрим ситуацию, когда у нас необходимо произвести 3 банковские операции: открыть счет, изменить информацию о счете и отменить открытие счета. Совершенно очевидно, что клиент отправит сообщения именно в таком порядке, как мы это записали. Каналы передачи данных должны передать эти сообщения не меняя их очередности. И обработаны они должны быть именно в этой очередности. При реализации сервера в однопоточном виде эти условия выполняются. Однако, при обработке сообщения бывает необходимо сделать другой вызов для получения дополнительной информации и, в этом случае, сервер выполненный в однопоточном виде будет простаивать. Основным способом решения данной проблемы (простоя сервера), в настоящий момент, является метод известный в англоязычной литературе под названием sharding/шардинг. Шардинг это распределение обработки сообщений на несколько микросервисов основываясь на некотором параметре, например первой букве фамилии). Допустим, что в примере, приведенном выше, производительности сервиса недостаточно для того, чтобы обслуживать клиентов. Тогда, мы делим обслуживание запросов на несколько работающих параллельно экземпляров микросервиса. Каждый экземпляр обрабатывает запросы, которые по какому-либо признаку изолированы. Например, пусть запросы на открытие счета для лиц, чьи первые буквы фамилий находятся в диапазоне А-Н, обрабатывает первый экземпляр сервиса, а остальные - второй [4][7][8][9] (рисунок 3).

Данное решение отличается простотой реализации. Каждый экземпляр сервиса может работать в однопоточном режиме, что значительно снижает затраты на его проектирование. При такой конфигурации, порядок обработки сообщений клиента может не соблюдаться. Например, возможен случай, когда клиент отправит сообщение и оно попадет в сервис 1, а следующее сообщение попадет в сервис 2. В этом случае, порядок выполнения этих операций будет зависеть от многих факторов и возможен случай, когда первое сообщение будет обработано после второго.

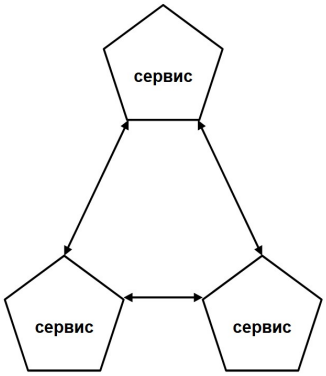

Рис. 2. Микросервесная архитектура без брокера сообщений.

Однако, т.к. как мы оговорили ранее, у нас нет разделяемых ресурсов между сервисом 1 и 2. Для нас важно только, чтобы, для вполне определенного клиента, поток сообщений выполнялся в порядке их отправки клиентом. При этом, не все задачи можно достаточно легко изолировать. Например, случай перевода средств от одного клиента другому может представлять определенные трудности, если клиенты попали в разные экземпляры микросервисов. Вследствие перечисленного, исполнение сервисов в многопоточном виде выглядит перспективным. Самым простым алгоритмом таких сервисов является создания отдельной нити для каждого разделяемого ресурса [10] или coroutines — многопоточный шаблон проектирования позволяющий писать асинхронный код в последовательной синхронной манере [11][12]. С ростом количества разделяемых ресурсов, такой подход становится не актуальным. Также, зачастую требуется групповая блокировка ресурсов, зависящих от некоторого более общего ресурса. Рассмотрим еще один пример. Допустим, что у нас планируется изоляция некоторого сектора аэропорта. В этом случае, мы должны закрыть все двери и не допускать их открытия. Одно из возможных решений - получить из базы данных все блокируемые ресурсы и создать на них блокировки в микросервисе. Однако, с ростом их числа, данный метод начинает требовать большое количество вычислительного времени. Более перспективным представляется метод с использованием разделяемой очереди и пула потоков [13][14][15]. Свободный поток смотрит сообщение в разделяемой очереди, проверяет на наличие блокировок и, если их нет, производит обработку данного сообщения.

В процессе роста информационной системы наблюдается постепенный переход разработки сервисов от однопоточной модели (как самой простой в исполнении) к sharding модели с последующей реализации по одной нити на один разделяемый ресурс. Дальнейшее распараллеливание бизнес логики приводит к созданию массивов средств синхронизации процессов или массивов ID разделяемых ресурсов. Отсутствие стандартизированного подхода приводит к увлечению времени разработки данных систем. Часть распараллеленных процессов влечет за собой рост производительности системы, а часть дала незначительный прирост производительности и не окупает времени на разработку.

Предметом научной работы являются обобщенный алгоритм обработки сообщений микросервисов.

Целью данной статьи является создание алгоритма, который предлагает обобщенный метод блокировки иерархических ресурсов основываясь на закодированной информации об иерархии при обработке поступающих в систему сообщений. Блокировка ресурсов происходит стандартизованным образом и не требует изменений в зависимости от

предметной области. Бизнес логика обработки различных сообщений разрабатывается программистом, без отвлечения ресурсов на проблемы синхронизации, способом близким к разработке однопоточных приложений. В данной работе рассматривается подход использования иерархических моделей описания ресурсов и общего алгоритма построения очередности обработки запросов, основанной на порядке приема сообщений и просмотре входящих сообщений на предмет доступности ресурсов. Подход с использованием иерархической модели блокируемых ресурсов и разделения процесса ожидания блокируемых ресурсов от процесса обработки сообщений является новым подходом решения задач синхронизации процессов. Это позволяет повысить производительность корпоративных приложений и уменьшить трудоемкость их разработки и получить заметный экономический эффект.

Теория

Рассмотрим поток различных сообщений, приходящий в микросервис. Данные сообщения относятся к одной доменной области, однако при этом должны быть

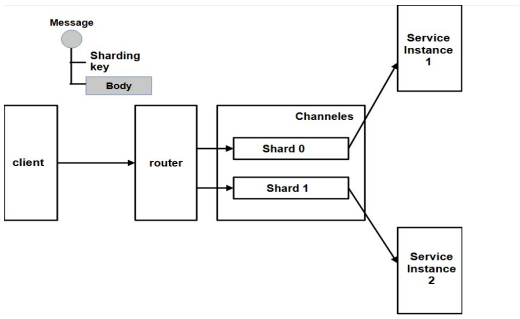

Рис. 3. Шаблон проектирования шардинг.

обработаны различным образом. Возможна ситуация, когда порядок обработки данных сообщений может влиять на результат. Рассмотрим последовательность сообщений, представленных на (рисунок 4). В данной последовательности мы предположили, что сообщения x_1, x_2, x_5 влияют на состояние некоторого разделяемого ресурса X_1, x_3 на X_2, а x_4 на X_3 и X_2, причем X_1 является родительским ресурсом для X_2 и X_3. Сообщения y_1 на Y_1, y_2, y_3 на Y_2, а y_4 на Y_3, причем состояние Y_1 является родительским для Y_2 и Y_3. Сообщение xy изменяет состояния X_1 и Y_1.

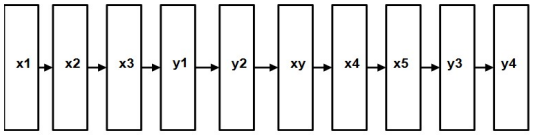

Рис. 4. Последовательность сообщений.

Предполагаемый порядок обработки сообщений представлен на (рисунок 5).

Здесь, сообщения, влияющие на некоторый разделяемый ресурс, выполняются в порядке доставки в микросервис, однако порядок обработки сообщений x и y не гарантируется. В приведенном примере, сообщение x_1 заблокировало выполнение сообщений x_2, x_3 потому как состояние X_1 является родительским для X_2 и X_3. В последовательности сообщений есть сообщения xy, которые могут влиять также на состояния ресурсов X_1, и Y_1. В этом случае, обработка сообщений x_i и y_i должна быть приостановлена, при достижении сообщений xy, и продолжена только после того, как это

сообщение будет обработано.

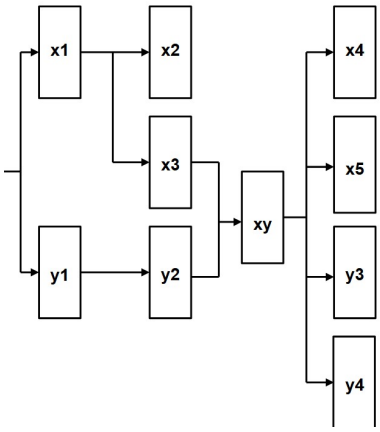

Рис. 5. Обработка сообщений.

Данный алгоритм достаточно трудно реализовать, основываясь на традиционных примитивах синхронизации (мьютексы, семафоры и т.д.), т.к., в общем случае, мы не знаем количество разделяемых ресурсов и, также, время обработки каждого сообщения может быть различным, вследствие того, что для выполнения действий нам возможно понадобятся дополнительные данные и мы вынуждены будем сделать вызов в другой микросервис для их получения. Обработка этих сообщений в одном потоке исполнения также не выглядит перспективным, потому как вызов в другой микросервис достаточно часто занимает много машинного времени и исполнение всех остальных сообщений будет приостановлено.

Рассмотрим структуру данных, представленную на (рисунок 6).

В представленной структуре данных Ordered set (коллекция элементов стандартной библиотеки C++), в котором хранятся указатели на Data Structure. Порядок следования этих указателей определяется наименьшим значением ключа в Set Node в этой структуре. Data structure также содержит Ordered set. Порядок там также определяется ключем в Set Node. Waiting messages представляют собой списки, в которые записи добавляются в порядке их добавления в структуру данных.

Рассмотрим алгоритм добавления сообщения в данную структуру представленный на (рисунок 7).

При получении сообщения, алгоритм сразу же генерирует все ключи, которые блокируют доступ к ресурсам. Данные ключи представляют собой структуры, описывающие иерархию разделяемых ресурсов и позволяющие быстро определять является ли один из ключей потомком или предком другого. Примеры генерации

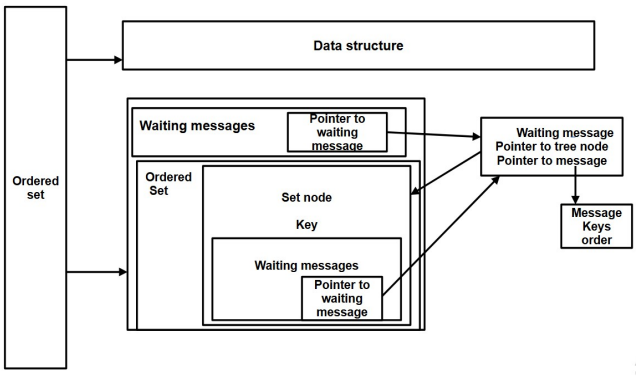

Рис. 6. Структура данных хранения информации о сообщениях.

таких ключей можно найти в следующих источниках [16][19][18]. В данной работе далее рассматриваются ключи, сгенерированные согласно алгоритму [18]. Для каждого ключа проводятся рассмотренные ниже проверки и сообщения попадают в соответствующие списки. В алгоритме мы оперируем указателями на структуры, вследствие этого, алгоритм не производит дублирование данных. Прежде всего, создается запись Waiting Message, в которой мы и храним сообщение и ряд другой информации. Ссылка на эту структуру широко используется в дальнейшем, для определения возможности обработки сообщения. Прежде всего, мы проверяем Ordered set на наличие Data Structure для наименьшего ключа которой, исследуемый ключ является дочерним. Если такая запись находится, значит у нас есть конфликт использования разделяемого ресурса и мы не можем произвести обработку данного сообщения.

В Ordered set может находиться несколько элементов DataStructure, для которых данный ключ является дочерним. Как будет показано далее, нам нужно найти DataStructure с наименьшим ключем, для которого данный ключ является дочерним. Там будет находиться крайняя добавленная запись, которая блокирует разделяемый ресурс. Произведем поиск в элементах Data Structure в Ordered set наличие Set Node с ключем, который совпадает с нашим. Если такая запись имеется, то мы просто добавляем ссылку на наш Waiting Messge в коллекцию Waiting Messages. Все записи в Data Structure Waiting Messages сохраняют исторический порядок, в соответствии с которым, они попали в микросервис. Set Node в коллекции Waiting Messages используются для определения наличия сообщений, которые блокированы данным ключом. Если же у нас Set Node с соответствующим ключем не найден, то мы создаем новую запись, помещаем ее в обрабатываемый элемент Data Structure, при этом, в Waiting Messages будет находится ссылка только на наше сообщение. Элемент Data Structure также обладает списоком Waiting Messages, куда мы добавляем наше сообщение. Помимо приведенной на рисуноке 6 структуры, у нас также имеется структура Locked Keys. Она представляет собой такую же коллекцию Ordered Set с ключами сообщений, которые либо уже обрабатываются, либо готовы к обработке. Для данных сообщений разделяемые ресурсы не блокируются ни одним обрабатываемым или подготовленным к обработке сообщением. Ключи сообщений находящиеся в Locked Keys имеют высший приоритет и не будут заблокированы ни одним новым сообщением. Последним шагом мы проверяем наличие ключей необработанных сообщений в Locked Keys.

Все ключи, которые на данный момент заблокированы, попадают в Waiting Message Keys. Если у нас заблокирован хотя бы один ключ, то для каждого ключа из всего подмножества ключей, для которого не найдено блокировок, создаются элементы Data Structure, в котором находится только один элемент множества Set Node с одним ключем и сообщение помещается в Waiting Message. Если блокировки отсутствуют, то сообщение готово к обработке и все связанные с ним ключи попадают в Locked Keys.

После окончания обработки сообщения, выполняется алгоритм, приведенный на рисунке 8. Для всех ключей, связанных с обработанным сообщением, найдем все элементы DataStructure в коллекции Ordrered Set, ключи которых являются потомками данного ключа. Для найденных структур выполним операцию обработки элемента данных Data Structure. Если ни один элемент Data Structure на предыдущем шаге не был найден, то данную операцию следует провести также и с элементом Data Structure, которая содержит ключ предок для текущего ключа, причем среди всех DataStructure с такими ключами, данная должна содержать наибольший ключ предок.

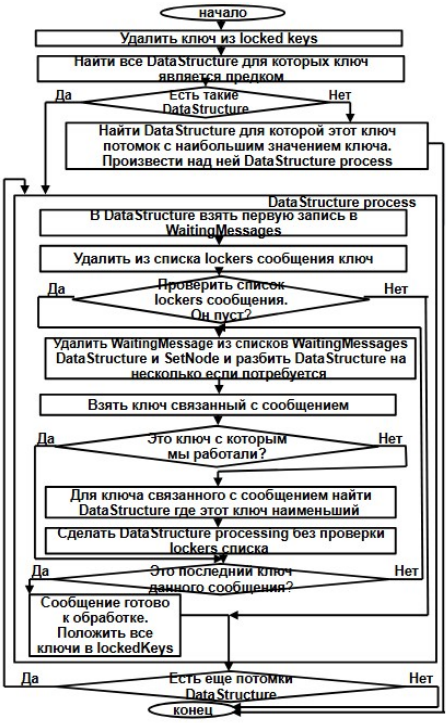

Рис. 7. Алгоритм окончания обработки записи.

Рассмотрим подробнее процесс обработки элемента Data Structure. В нашем случае, первая запись в коллекции Waiting Messages связана с сообщением, которое потенциально может быть обработано при снятии блокировки, связанной с данным ключом. Мы удаляем блокировку, связанную с обрабатываемым ключом, из списка Lockers в элементе Waiting Message. Если список Lockers пуст, то рассматриваемое сообщение более не заблокировано ключами и мы стираем ссылки на элемент Waiting Message из коллекции Waiting Messages для рассматриваемого элемента Data Structure и Set Node. На данном шаге, у нас может возникнуть ситуация, когда элемент Set Node больше не имеет каких-либо сообщений, которые блокировались ключом, связанным с ним. В этом случае, его нужно удалить. Если в эелменте Data Structure больше нет ни одного элемента Set Node, то его тоже нужно удалить. При удалении элемента Set Node возможна ситуация, когда удаляемый объект имеет наименьшее значение ключа для данных элементов Data Structure. В этом случае, возможна ситуация, когда у нас есть несколько ключей потомков, которые не блокируют друг друга. Следовательно, у нас должно быть создано несколько элементов Data Structure, в которые попадут независимые поддеревья первоначального дерева блокированных сообщений. Данное разделение провести достаточно просто. Нам нужно взять наименьший ключ (он находится в начале коллекции Ordered Set) и сгенерировать верхний предел для дочернего поддерева дерева блокированных сообщений. Функционал коллекции Ordered Set позволяет найти положение ключа, который не менее выбранного ключа. Данная операция называется lower_bound. Далее, мы копируем данные из коллекции Waiting Messages каждого элемента Set Node в элементе Data Structure коллекции Waiting Messages и сортируем их в порядке возрастания. Полученные элементы Data Structure добавляем в коллекцию Ordered Set, а первоначальный элемент Data Structure удаляем. С нашим сообщением может быть связано несколько ключей, которые были разблокированы ранее. Их также просто найти, т.к. они будут в элементе Data Structure, ключ которого, предок для текущего ключа. Т.к. мы работаем с указателями, то они будут указывать на сообщение, у которого список блокировок пуст. Этот элемент Waiting Message следует удалить из коллекции. При этом, нам возможно нужно будет удалить

коллекцию Set Node, элемента DataStructure и возможно разделить дерево элементов в DataStructure на несколько поддеревьев, способом, описанным выше. Сообщение, у которого пуст список Locked, готово к обработке и ключи, с ним связанные, следует сохранить в коллекцию Locked Keys.

В алгоритме присутствуют два участка, которые должны выполняться непрерывно. Первая область начинается с просмотра сообщения в очереди и заканчивается перед обработкой сообщения. Вторая начинается после обработки сообщения и заканчивается проверкой блокированных ключей. Таким образом, только обработка сообщения выполняется многопоточно.

Анализ

Произведем оценку сложности по времени работы алгоритма. Здесь и далее рассматриваются ключи, сгенерированные по алгоритму [18].

Рассмотрим процесс добавления записи. В этом случае, сложность алгоритма поиска элемента Data Structure для добавления в него нового сообщения O(log m), где m - количество ключей блокирующих поддеревьев. В худшем случае, это число кратно количеству ожидающих сообщений n. Поиск нужной коллекции Set Node, также имеет вычислительную сложность O(log m). Добавление записи в коллекцию Waiting Messages и Locked Keys имеет сложность O(1). Следовательно, сложность всего алгоритма добавления новой записи O(log m).

При окончании обработки сообщения, для каждого ключа связанного с ним, мы ищем соответствующую элемент Data Structure и в данном случае вычислительная сложность алгоритма O(log m). При разделении элемента Data Structure на несколько, мы вынуждены сортировать коллекцию Waiting Messages и, в худшем случае это имеет сложность O(n*log(n)).

Покажем корректность работы алгоритма.

Все ключи, связанные с сообщением, влияют на блокировку сообщений.

При получении сообщения, для всех ключей проверяется ключи которые являются предками текущего ключа в DataStructure. Если такие элементы находятся, то информация о данном сообщении сохраняется в DataStructure. Если таких ключей не выявлено, но данный ключ является предком для других сообщений то этого ключа создается элемент DataStructure. Таким образом, либо все ключи имеют записи в соответствующих элементах DataStructure, либо сообщение отправляется на обработку.

При добавлении ключа, сохраняется их порядок доступа к разделяемым ресурсам.

При получении нового сообщения проверяются все ключи на наличие блокировки. Алгоритм проверяет наличие элемента DataStructure, ключ которой является предком для проверяемого ключа. Наличие такой структуры говорит о том, что у нас уже есть ожидающие сообщения, которые блокируют ресурс. Данный ресурс необходим для обработки нашего сообщения. Однако, вследствие того, что у нас может быть несколько элементов DataStructure с ключами, которые являются предками к текущему, нам нужно найти наименьший такой объект. Среди всех элементов DataStructure, чьи ключи являются предками к текущему, тот элемент, чей ключ меньше, был добавлен позже. Действительно, мы добавляем новый элемент DataStructure в случае, если не найден ни один объект, для которого ключ является предком нашего ключа. Следовательно, если мы находим несколько элементов DataStructure, для которых наш ключ является

потомком, то это означает, что тот объект с наименьшим ключем был добавлен позже всего. При этом, как не трудно заметить, ключи упорядочены и потомки всегда больше, чем предки. В найденной структуре мы либо добавляем элемент WaitingMassege к коллекции WaitingMasseges структуры SetNode, если у нас есть SetNode с данным ключом, или же создаем новую структуру SetNode. Порядок сообщений, добавляемых в элемент DataStructure, обеспечивается очередью коллекции WaitingMasseges. Таким образом, у нас поддерживается очередность добавления сообщений внутри элементы DataStructure.

Разделение элемента DataStructure сохраняет порядок сообщений.

При завершении обработки сообщения для каждого ключа, связанного с ним, мы находим элемент DataStructure, для которого это сообщение является потомком. Если у нас есть несколько таких объектов, для которых данный ключ является потомком, то мы выбираем тот, у которого ключ наибольший. При разделении DataStructure возникает ситуация, когда следующее сообщение в очереди WaitingMasseges содержит ключ, который наименьший, т.е. это предок для всех остальных ключей в данном объекте. У сообщения это последний ключ, который блокирует данное сообщение от обработки. Ни одного другого сообщения с данным ключом нет. Алгоритм отправляет сообщение на обработку, и должен удалить структуру SetNode т.к. нет больше сообщений, ассоциированных с данным ключом. Следовательно, должен поменяться ключ всего элемента DataStructure. В элементе DataStructure могут находиться структуры SetNode, ключи которых являются потомками удаляемого ключа, но при этом друг другу они могут не быть ни потомками, ни предками. Таким образом, мы получаем несколько подмножеств ключей, для каждого из которых создаем свой элемент DataStructure. Т.к была выбран элемент DataStructure с наибольшим определяющим ключом, то все ключи связанные с его SetNode больше этого ключа. Вследствие этого, ключи новых элементов DataStructure, чей определяющий ключ - это минимальный ключ входящих в неё SetNode, будут больше удаляемого ключа. А, следовательно, порядок элементов DataStructure не поменяется и после обработки сообщения будут протестированы на выполнение более ранних сообщений. Внутри новых элементов DataStructure также будет отсортирован список WaitingMasseges, следовательно более старые сообщения будут первыми в этом списке.

Покажем что алгоритм корректно выбирает сообщения.

После того как алгоритм произвел обработку сообщения, происходит высвобождение ресурсов и выбор следующего сообщения. Последовательно происходит освобождение ресурсов, связанных с ключами. Для каждого ключа находятся все элементы DataStructure, ключи которых являются потомками освобождаемого ключа. Как было сказано ранее, такая ситуация возможна только после разделения элемента DataStructure. Следовательно, у нас есть несколько кандидатов на обработку. Для каждой такого кандидата, мы берем первое сообщение в его очереди WitingMessages и удаляем разблокированный ключ из списка Locked. Далее, мы проводим описанные выше операции и проверяем сообщение на возможность обработки. Наш ключ мог блокировать несколько таких сообщений, которые являются его потомками, но могут быть обработаны независимо друг на друга. Если таких сообщений не найдено, то мы рассматриваем добавленные позже сообщения, которые попали в элементы DataStructure с ключом, который является предком удаляемого. Т.к обработка этого сообщения зависит от нашего ключа, алгоритм берет только одно сообщение, элемент DataStructure которого имеет наибольшее значение и, следовательно, был добавлен раньше других. Как мы видим, хронология обработки сообщений соблюдена и

представленный алгоритм корректно выбирает сообщения.

Рассмотрим возможность взаимной блокировки сообщений с несколькими ключами. Как мы видим, алгоритм имеет 2 непрерывные секции, однако, они имеют доступ к одному и тому же разделяемому ресурсу - списку ключей. Следовательно, это можно реализовать при помощи одного объекта синхронизации. При этом происходит единовременная проверка возможности заблокировать доступ к некоторым ресурсам и невозможна ситуация, когда оба сообщения потребуют доступ к одним и тем же ресурсам и будут одновременно блокировать доступ. Следовательно, взаимная блокировка невозможна.

Заключение

Приведенный алгоритм позволяет изменять детализацию блокируемых ресурсов, при этом не увеличивая сложность написания процедур обработки сообщений. С помощью этого алгоритма создаются иерархические структуры блокируемых ресурсов, которые позволяют выбирать уровень иерархии блокировки ресурса. Предлагаемый подход позволяет программисту декларировать, какие ресурсы необходимы для обработки и не оперировать категориями многоточечного исполнения. Разделение алгоритма блокировки ресурсов и процедуры обработки сообщений позволяет разрабатывать алгоритмы обработки сообщений, исходя из парадигмы однопоточной среды, что уменьшает возможность возникновения ошибок, связанных с многопоточностью.

Алгоритм лишен такой проблемы, как взаимная блокировка ресурсов, что также положительным образом влияет на отказоустойчивость систем.

К недостаткам можно отнести то, что блокировка ресурса происходит перед началом обработки сообщения, а снятие блокировки после его окончания.

Применение данного алгоритма значительно повышает отказоустойчивость микросервисов.

В качестве дальнейшего направления данного научного исследования предполагается проведение сравнительного анализа эффективности построения микросервисов в однопоточном исполнении, без блокировок и с использованием данного алгоритма. Также, планируется разработка алгоритма обработки запросов с иерархической структурой блокируемых ресурсов. В свете проблем, которые наблюдаются в данной области на современном этапе, направление данного исследования является перспективным.

Библиография

1. Asynchronous Request-Reply pattern. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/async-request-reply
2. Microservices: Asynchronous Request Response Pattern. URL: https://medium.com/@pulkitswarup/microservices-asynchronous-request-response-pattern-6d00ab78abb6
3. Request/Response Pattern with Spring AMQP. URL: https://reflectoring.io/amqp-request-response/
4. Richardson C. Microcervices Patterns. – 2019.
5. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. – 20.
6. Fowler M., Rice D., Foemmel M., Hieatt E., Mee R., Stafford R. Patterns of Enterprise Application Architecture. – 2004.

7. Sadalage P., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. – 2012.
8. Wang G., Shi Zhijie J., Nixon M., Han S. SoK: Sharding on Blockchain. // AFT '19: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. – 2019. – С. 41-61. URL: https://dl.acm.org/doi/abs/10.1145/3318041.3355457
9. Introducing Elastic Scale preview for Azure SQL Database. URL: https://azure.microsoft.com/en-us/blog/introducing-elastic-scale-preview-for-azure-sql-database
10. Tasks and Parallelism: The New Wave of Multithreading. // [Электронный ресурс] URL: https://www.codemag.com/article/1211071/Tasks-and-Parallelism-TheNew-Wave-of-Multithreadin
11. Coroutine Is a New Thread. URL: https://medium.com/globant/coroutine-is-a-new-thread-934d9956ce2e
12. Threads vs Coroutines in Kotlin. URL: https://www.baeldung.com/kotlin/threads-coroutines
13. Christudas B. Query by Slice, Parallel Execute, and Join: A Thread Pool Pattern in Java. URL: https://web.archive.org/web/20080207124322/http://today.java.net/pub/a/today/2008/01/31/query-by-slice-parallel-execute-join-thread-pool-pattern.html
14. Programming the Thread Pool in the .NET Framework. URL: https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms973903(v=msdn.10)?redirectedfrom=MSDN
15. Introduction to Thread Pools in Java. URL: https://www.baeldung.com/thread-pool-java-and-guava
16. O'Neil P. et al. ORDPATHs: Insert-friendly XML node labels // Proceedings of the 2004 ACM SIGMOD international conference on Management of data. – 2004. – С. 903-908. URL: http://www.cse.iitb.ac.in/infolab/Data/Courses/CS632/2014/2007/Papers/ordpath.pdf
17. PostgreSQL index-ltree. URL: http://www.sai.msu.su/~megera/postgres/gist
18. Кириллов В.С. Бинарное кодирование иерархических структур // Вестник КРАУНЦ. Физико-математические науки. – 2023. – Т. 43, № 2. – С. 44-54. https://doi.org/10.26117/20796641-2023-43-2-44-54 ISSN 2079-664 EDN: XUMKPG

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Представленная статья на тему «Обработка запросов с иерархическим характером разделяемых ресурсов» соответствует тематике журнала «Программные системы и вычислительные методы» и посвящена вопросу, связанному с увеличением нагрузки на корпоративные приложения, что приводит к переходу на распределенные вычисления. По мнению авторов с приходом Kibernetis, Docker и DevOps технологий, широкое распространение получила микросервесная архитектура и системы, основанные на передаче сообщений. За время использования данной архитектуры, появилось несколько шаблонов проектирования. В частности, широкое распространение получили микросервисы, передача сообщений в которых построена по архитектуре asynchronous request/response.
Авторами рассмотрены системы, построенные без брокера сообщений. Рассмотрены микросервисы, использующие данный метод передачи данных, обладающие большой гибкостью и позволяющие строить системы с большой пропускной способностью. Однако,

на сообщения и способ передачи и обработки сообщений накладываются определенные ограничения. Авторами проанализирована ситуация, когда клиенту необходимо произвести три операции: открыть счет, изменить информацию о счете и отменить открытие счета. Авторам кажется очевидным, что клиент отправит сообщения именно в таком порядке. При реализации сервера в однопоточном виде обработка сообщений происходит в такой последовательности. При обработке сообщения бывает необходимо сделать другой вызов для получения дополнительной информации и тогда сервер, выполненный в однопоточном виде, будет простаивать. По мнению авторов решением данной проблемы является sharding.

Статья достаточно структурирована - в наличии введение, заключение, внутреннее членение основной части. В статье авторами рассмотрены: поток различных сообщений, приходящий в микросервис; структуре данных Ordered set, в котором хранятся указатели на Data Structure, алгоритм добавления сообщения в данную структуру.

Авторы статьи провели аналитический обзор российской и зарубежной актуальной литературы. К недостаткам можно отнести: качество языка не отвечает требованиям научной статьи.

Статья очень средняя по содержанию, смыслу, носит обзорный характер, отсутствуют четко сформулированные выводы и т.п. (с точки зрения ВАКовских требований). Чувствуется «студенческая рука». Также, практическая значимость статьи на среднем уровне.

Из содержания статьи слабо прослеживается научная новизна. Отсутствуют четко сформулированные цель, предмет исследования, также не обоснована актуальность исследования.

Рекомендуется четко обозначить научную новизну исследования, сформулировать предмет, цель исследования, обосновать актуальность. Обратить внимание на содержание и оформление списка литературы в соответствии с ГОСТ. Также будет целесообразным добавить о перспективах дальнейшего исследования.

Рекомендуется более четко обозначить проблему исследования и авторский вклад.

Статья «Обработка запросов с иерархическим характером разделяемых ресурсов» требует доработки по указанным выше замечаниям. После внесения поправок рекомендуется к повторному рассмотрению редакцией рецензируемого научного журнала.

Результаты процедуры повторного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.

Со списком рецензентов издательства можно ознакомиться здесь.

Статья посвящена важной и актуальной проблеме современной компьютерной науки - разработке эффективного алгоритма обработки сообщений в микросервисных архитектурах с учетом иерархического характера разделяемых ресурсов. Автор предлагает инновационное решение, которое позволяет стандартизировать процесс синхронизации доступа к ресурсам в условиях параллельной обработки запросов и высокой нагрузки на систему.

Основной предмет исследования составляет разработка обобщенного алгоритма, который обеспечивает корректную обработку сообщений, учитывая их взаимозависимость через иерархию разделяемых ресурсов. Автор подробно рассматривает проблему соблюдения порядка выполнения операций при распределенной обработке сообщений, что особенно важно для таких критически важных систем, как банковские приложения или системы управления ресурсами.

Методологическая основа исследования базируется на применении иерархической модели блокировки ресурсов, где каждый ресурс описывается специальным ключом, отражающим его положение в иерархии. Автор предлагает оригинальную структуру данных, включающую Ordered Set и Waiting Messages, которая позволяет эффективно управлять очередностью обработки сообщений. Особое внимание уделено разделению процессов блокировки ресурсов и их непосредственной обработки, что существенно упрощает разработку сложных распределенных систем и снижает вероятность ошибок, связанных с многопоточностью.

Актуальность представленного исследования не вызывает сомнений, так как проблема эффективной обработки сообщений в микросервисных архитектурах становится все более значимой с ростом популярности распределенных систем. Автор убедительно демонстрирует, что традиционные подходы к синхронизации, такие как мьютексы или семафоры, оказываются недостаточно эффективными в условиях работы с иерархическими ресурсами и высокой нагрузкой.

Научная новизна работы проявляется в нескольких аспектах. Во-первых, предложен принципиально новый метод блокировки ресурсов, основанный на их иерархическом представлении. Во-вторых, разработан алгоритм, который позволяет разделить процессы синхронизации и обработки сообщений, что значительно упрощает разработку сложных систем. В-третьих, доказана корректность работы алгоритма и отсутствие взаимных блокировок, что существенно повышает надежность и отказоустойчивость систем.

Статья отличается четкой структурой и логичным изложением материала. Введение содержит обоснование актуальности исследования и постановку проблемы. Теоретическая часть подробно описывает предлагаемый алгоритм и используемые структуры данных. Особого внимания заслуживает анализ сложности алгоритма, где автор демонстрирует его эффективность, и доказательство корректности работы, подтверждающее надежность предложенного решения. Графические иллюстрации существенно дополняют текст и делают его более наглядным. Библиография включает актуальные и релевантные источники, отражающие современное состояние исследований в области микросервисов и параллельных вычислений.

Автор приходит к обоснованным выводам о том, что предложенный алгоритм успешно решает поставленную задачу обработки иерархических ресурсов в микросервисных архитектурах. Основные преимущества решения включают стандартизацию процесса блокировки ресурсов, упрощение разработки за счет разделения процессов синхронизации и обработки, а также гарантированное отсутствие взаимных блокировок. Эти характеристики делают алгоритм особенно ценным для практического применения.

Представленная статья, несомненно, вызовет значительный интерес у широкого круга специалистов в области распределенных систем, разработчиков микросервисов и архитекторов программного обеспечения. Практическая значимость исследования особенно очевидна для таких областей, как финансовые системы, корпоративные приложения и другие сферы, где требуются высокая производительность и надежность обработки транзакций.

Учитывая научную ценность, актуальность и высокое качество изложения материала, статью можно рекомендовать к публикации. Представленное исследование является значимым вкладом в область компьютерных наук и открывает перспективные направления для дальнейших разработок в области обработки распределенных запросов. Особого внимания заслуживает предложенный автором план дальнейших исследований, включающий сравнительный анализ эффективности различных подходов к построению микросервисов.