УДК 004.94

Разработка имитационной модели работы микросервисного приложения

И. И. Мариничев, Е. А. Шуватова, С. Ю. Землянская
Донецкий национальный технический университет, г. Донецк
E-mail: ilia2949@mail.ru

Аннотация:
В статье рассмотрены проблемы, которые возникают при проектировании архитектуры микросервисного приложения, сформулирована задача разработки приложения с наиболее эффективной архитектурой. В связи с чрезмерной трудоемкостью проведения натурных экспериментов на разных моделях архитектур для исследования поведения приложения предлагается использовать объектную модель микросервисного приложения и имитационное моделирование с целью получения необходимых параметров работы и проведения анализа эффективности выбранной архитектуры. Выделены основные сущности модели, проанализированы их ключевые особенности и описаны принципы работы. Разработаны алгоритмы взаимодействия компонентов и функционирования имитационной модели.

Введение

В эпоху цифровой трансформации и растущих требований к производительности программного обеспечения, микросервисная архитектура стала популярным подходом к созданию сложных приложений. Разделение монолитных приложений на независимые, легко управляемые микросервисы позволяет улучшить масштабируемость, гибкость и устойчивость систем. Однако, выбор оптимальной комплектации и функционала микросервисов представляет собой значительную задачу, требующую тщательного анализа и моделирования.

В данной статье мы рассматриваем методологию создания имитационной модели работы микросервисного приложения, что облегчит сравнение вариантов разделения и взаимодействия микросервисов.

Постановка задачи исследования

Одна из проблем, с которой сталкиваются разработчики при переходе к микросервисной архитектуре, заключается в необходимости проверки эффективности выбранного распределения функциональности между микросервисами.

Микросервисная архитектура [1][2] — это подход к созданию приложения, подразумевающий отказ от единой, монолитной структуры. Вместо этого, систему разделяют на небольшие независимые сервисы. Каждый выполняет определённую функцию и взаимодействует с другими сервисами через API. Это позволяет сделать систему более гибкой и отказоустойчивой, а также использовать масштабирование системы для распределения нагрузки и увеличения скорости работы. Однако, рост числа пользователей, увеличение самой системы и нагрузки на неё влечёт за собой усложнение взаимодействия между микросервисами и обновления их данных. Это может сказаться на сложности управления системой и увеличении затрат на поддержку её инфраструктуры.

Чтобы минимизировать эти риски, требуется ещё на этапе проектирования грамотно разбить программу на микросервисы [3][4], оценить качество, выявить узкие места системы и с учётом этого оптимизировать архитектуру программы [5]. Одним из вариантов решения этой проблемы является создание имитационной модели программы. В этой статье мы описываем процесс создания такой модели и алгоритм её работы.

Основное содержание и результаты работы

В результате анализа систем, имеющих микросервисную архитектуру, можно сделать вывод, что модель должна содержать следующие сущности: API, заявка, функция, запрос, микросервис, балансировщик нагрузки, таблица, брокер сообщений, сервер.

Рассмотрим эти сущности более детально.

– Заявка – имитация запроса пользователя.

– Функция – имитация работы основных функциональных компонентов клиентской части приложения (например: проверка работ, просмотр лекций/заданий и т.д.).

– Запрос – имитация запроса на обработку определённых данных от клиентского приложения к серверу. Каждая функция может

иметь несколько запросов (например: функция «Проверка работ» может иметь запросы на получение списка студентов, приславших работы, запрос на скачивание работы отдельного студента, запрос на выставление оценки студенту и т.д.).

– Таблица – имитация таблицы из базы данных.

– Микросервис – имитация микросервиса определённого типа. Микросервис может обрабатывать определённый набор типов запросов из функций, в зависимости от таблиц хранящихся в базе данных микросервиса. Для обработки запроса может потребоваться работа как с одной, так и с несколькими таблицами из базы данных.

– API – сущность, представляющая из себя API-слой и имитирующая работу пользователей с функциями.

– Балансировщик нагрузки – сущность, имитирующая работу подпрограммы, распределяющей нагрузку между экземплярами типов микросервисов.

– Брокер сообщений – сущность, представляющая из себя функцию, которая обеспечивает связь между различными типами микросервисов и их экземплярами, своевременное обновление данных в их локальных базах данных.

– Сервер – физический сервер, на котором расположены экземпляры микросервисов.

Рассмотрим более детально требования к сущностям.

Заявка должна содержать в себе следующую информацию:

– тип заявки (тип запроса, который направила функция)

– тип ответа (ответ/ошибка);

– время прихода;

– допустимое время ожидания;

– фактическое время ожидания;

– время обработки;

– id функции создателя;

– тип микросервиса, на который направлена заявка;

– id экземпляра микросервиса, в очередь к которому была записана заявка.

Запрос должен содержать в себе следующую информацию:

– список таблиц, которые потребуется обработать для удовлетворения запроса;

– набор требуемых операций для каждой из таблицы.

Таблица должна содержать в себе следующую информацию:

– набор всех операций, которые могут совершаться с таблицей;

– время обработки каждой из операций, в зависимости от того, на каком сервере находится таблица;

– процент загрузки центрального процессора сервера и объём потребляемой памяти во время обработки каждой из операций, в зависимости от того, на каком сервере находится таблица.

Микросервис должен содержать набор следующих данных и функций:

– список таблиц, содержащихся в базе данных микросервиса;

– список возможных запросов к микросервису;

– приблизительный шанс ошибки/сбоя данных;

– статус активности.

Входные данные:

– Заявка.

Выходные данные:

– Заявка с ответом.

– Заявка с ошибкой.

– Заявка на обновление данных.

Функция должна содержать в себе следующие параметры:

– Список запросов к бэкенду, которые могут потребоваться для работы пользователя с функцией;

– Список типов микросервисов, отвечающих за работу различных запросов в функции;

– Шансы выбора запроса (некоторые запросы могут быть использованы чаще других).

Входные данные:

– Количество заявок, адресованных в функцию (без данных и типа).

Выходные данные:

– Количество заявок с типом запроса для каждого типа микросервисов, содержащихся в функции.

API должен содержать следующие параметры:

– Список функций программы;

– Шанс вызова каждой отдельной функции;

– Список необработанных заявок;

– Балансировщик нагрузки;

– Список обработанных заявок.

Входные данные:

– Количество запросов.

Выходные данные:

– Обработанные заявки/статистика обработанных заявок.

Балансировщик нагрузки должен содержать следующие параметры:

– Очередь заявок для каждого типа микросервиса;

– Очередь с таймером для всех экземпляров по каждому микросервису;

Входные данные:

– Список заявок.

Выходные данные:

– Обработанные заявки.

Брокер сообщений должен содержать следующие параметры:

– Схему связей/зависимостей микросервисов;

– Лог событий.

Входные данные:

– Сообщение от микросервиса с информацией об изменении данных.

Выходные данные:

– Список экземпляров микросервисов со связанными данными.

Сервер должен содержать следующие параметры:

– Данные о нагрузке на процессор и оперативную память каждого развёрнутого контейнера с микросервисом;

– Степень загруженности сервера;

– Список размещённых на нём экземпляров микросервисов.

Входные данные:

– Запрос состояния сервера.

Выходные данные:

– Данные о состоянии сервера.

Составление алгоритма работы модели

Теперь, когда мы определились с требованиями к модели, можно описать принципы её работы и составить алгоритм.

Создание заявок.

Создание заявок должно имитировать запросы пользователей в реальном времени. Для этого необходимо подобрать функцию распределения, учитывая особенности активности пользователей на протяжении периода рабочего дня.

Таким образом, схема создания заявок будет следующей:

В цикле от 0 до N с шагом 1, где 1 – это некий интервал времени, каждую итерацию цикла, согласно выбранному закону распределения, создаётся определённое количество заявок. Время прихода заявки следует распределить между созданными в рамках интервала заявками равномерным способом. Схема создания заявок приведена на рисунке 1.

Имитация работы пользователей с функционалом программы.

После создания заявок и определения времени их прихода, необходимо распределить их назначения. Для этого сымитируем выбор пользователем определённой функции нашей программы. Каждая функция может быть выбрана пользователем с некоторой вероятностью, значение которой было определено эмпирически. Случайную функцию будем выбирать, используя метод колеса рулетки.

За реализацию возможностей каждой из функций отвечает набор запросов, которые сгруппированы по типам микросервисов. Для распределения заявок воспользуемся эмпирически определёнными вероятностями выбора запросов и методом колеса рулетки.

После этого, записываем в заявку тип микросервиса, который обрабатывает данный тип запроса. Алгоритм процесса представлен на рисунке 1.

Рисунок 1 – Определение количества заявок

Распределение заявок между экземплярами микросервисов при помощи балансировщика нагрузки.

Принцип работы балансировщика нагрузки следующий [6]:

В цикле от 0 до N, где N – количество заявок в общей очереди, проходим по списку заявок $\{x_1 \ldots x_n\}$ и распределяем их между экземплярами микросервисов. Для этого выбираем список очередей $\{q_1 \ldots q_k\}$ к экземплярам выбранного типа микросервисов $\{m_1 \ldots m_k\}$ и записываем заявку в одну из них.

Существует несколько основных алгоритмов работы балансировщика нагрузки, такие как Round Robin, Weighted Round Robin, Least Connections и др. Для имитационной модели микросервисного приложения был выбран метод реализации Least Connections, поскольку этот метод сочетает в себе простоту реализации с высокой степенью надёжности: выбирается очередь того из экземпляров микросервисов, который наименее загружен на момент прихода заявки (рисунок 2).

Каждая заявка имеет ограниченное время ожидания. Если это время будет превышено, заявка считается необработанной и возвращается обратно с пометкой «истёкший срок ожидания». Для этого в каждой из очередей экземпляров создаётся отдельная переменная времени освобождения микросервиса $t_{осв}$.

Рисунок 2 – Распределение заявок между экземплярами микрсоервисов

Время освобождения $t_{осв} = t_{осв} + t_{обр}$, где $t_{обр}$ – время обработки очередной заявки экземпляром микросервиса, которое вычисляется внутри микросервиса в установленных путём исследования пределах (детальный разбор работы микросервиса будет рассмотрен ниже).

Время освобождения микросервиса обновляется по мере поступления и обработки заявок.

Имитация обработки заявок и работы микросервиса.

Для имитации работы микросервиса необходимо произвести разбиение программы на микросервисы, а затем определить по имеющимся программным данным предположительное среднее время обработки запроса микросервисом и его зависимость от объёма обрабатываемых данных. Также каждый микросервис имеет определённый эмпирически шанс программного сбоя.

На каждой итерации своей работы балансировщик нагрузки должен проверять микросервисы на активность. Для этого необходимо проверять на активность физические сервера, на которых лежат микросервисы. Если физический сервис недоступен, значит недоступен и микросервис. В таком случае все заявки из его очереди считаются потерянными и возвращаются в систему с соответствующей пометкой.

Алгоритм обработки очередей к микросервисам представлен на рисунке 3.

Рисунок 3 – Процесс обработки заявок

После поступления заявки микросервис получает из неё данные о запросе пользователя, таблицах, которые необходимо обработать, и наборе операций для каждой из таблиц. Временем обработки заявки считается сумма времени выполнения всех операций с таблицами. Среднее время выполнения типовой операции с таблицей в модели рассчитывается для каждой таблицы в

зависимости от того, на каком сервере будет находиться микросервис, и хранится в таблице. Работа микросервиса изображена на рисунке 4.

Рисунок 4 – Имитация работы микросервиса

Брокер сообщений

Брокер сообщений содержит в себе связи между базами данных микросервисов. В нём хранятся все связи между базами данных типов микросервисов и, соответственно, их экземпляров.

Существуют 2 основных типа брокеров сообщений [9][10]: активные и пассивные. Активные («push» модель) брокеры сами отправляют информацию об обновлении в экземпляры микросервисов. В случае с пассивными брокерами («pull» модель), микросервисы-потребители сами запрашивают из него информацию об обновлениях. В рамках модели предлагается использовать именно «push» модель брокера сообщений.

При успешной обработке заявки на изменение данных, микросервис отправляет в брокер сообщений данные об изменённой таблице и конкретных операциях над ней. После этого брокер создаёт заявки на обновление данных для каждого экземпляра микросервиса со связанной таблицей и записывает их в соответствующую очередь. Временем прихода такой заявки считается время освобождения микросервиса. Заявка на обновление обрабатывается микросервисом вне основной очереди, сразу же, как только он освободится от обработки предыдущего запроса.

Заявка сохраняется в очереди на обновление до тех пор, пока сервер не отчитается о её обработке, и не имеет предельного срока ожидания. Алгоритм работы брокера сообщений представлен на рисунке 5.

Рисунок 5 – Имитация работы брокера сообщений

Сервер

Сервер представляет собой модель физического сервера, на котором хранятся экземпляры микросервисов. Параметры модели сервера включают список всех лежащих на нём микросервисов, а также текущий уровень нагрузки на сервер как по центральному процессору, так и по оперативной памяти. Каждый развёрнутый микросервис добавляет постоянную нагрузку на сервер. Нагрузка также увеличивается во время выполнения микросервисом операций.

Если нагрузка на сервер достигает предела его вычислительной мощности, время обработки данных в лежащих на нём микросервисах увеличивается, пока сервер не будет разгружен. Кроме этого, сервер имеет определённый шанс перехода в состояние неисправности. В таком случае все микросервисы на нём считаются недоступными, а все отправленные на них заявки возвращаются с пометкой «ошибка».

Выводы

В статье рассмотрена проблема проектирования эффективной архитектуры микросервисного приложения. Проектирование микросервисной архитектуры информационной системы – довольно сложная задача и от выбранного варианта решения существенно зависит работоспособность создаваемой системы. Таким образом, основной проблемой проектирования становится проблема проверки качества разбиения функционала системы на составляющие и анализа эффективности выбранной архитектуры.

Для решения данной проблемы предложено использование имитационного моделирования. В результате анализа функционирования микросервисного приложения определены основные компоненты имитационной модели и составлен алгоритм работы модели.

Получаемые в результате моделирования параметры для разных конфигураций используются для выбора конфигурации, обеспечивающей наиболее эффективное функционирование моделируемой системы.

Литература

1. Мариничев, И. И. Проблема выбора архитектуры для платформы дистанционного обучения / И. И. Мариничев, Е. А. Шуватова, С. Ю. Землянская // Материалы XIV Международной научно-технической конференции в рамках IX Международного Научного форума Донецкой Народной Республики 24-25 мая 2023 г. – Донецк: ДонНТУ, 2023. - С. 249-253.

2. Ньюмен, С. Создание микросервисов / С. Ньюмен // СПб.: Питер, 2016. — 304 с.: ил. — (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-02011-4 С. 22-30

3. Ньюмен, С. От монолита к микросервисам. Эволюционные шаблоны для трансформации монолитной системы. / С. Ньюмен // От монолита к микросервисам: Пер. с англ. — СПб.: БХВ-Петербург, 2021. — 272 с.: ил. ISBN 978-5-9775-6723-7. С. 97-100

4. Евланов, М. В. Синтез сервис-ориентированной архитектуры информационной системы [Электронный ресурс]. - URL: http://www. rusnauka.com/ 11_EISN_2010/Informatica/64250.doc.htm (дата обращения: 05.05.2024).

5. Долженко, А. И. Анализ качества микросервисов информационной системы на базе нечеткой модели / А. И. Долженко, И. Ю. Шполянская, С. А. Глушенко // Прикладная информатика, 2019. - Т.14, №5(83). – С. 120-128.

6. Ефимов, Г. Адаптивная балансировка нагрузки или как повысить надёжность микросервиса [Электронный ресурс]. // Хабр: сайт. — URL: https://habr.com/ru/companies/ozontech/articles/558926/ (дата обращения: 20.04.2024)

7. Богомаз, М. Популярные брокеры сообщений в микросервисной архитектуре: NATS, Kafka и RabbitMQ // Timeweb Cloud: сайт. [Электронный ресурс]. - URL: https://timeweb.cloud/tutorials/microservices/populyarnye-brokery-soobshchenij/ (дата обращения: 05.05.2024)

8. Шеламов, Д. Асинхронное взаимодействие. Брокеры сообщений. Apache Kafka [Электронный ресурс] // Хабр: сайт. — URL: https://habr.com/ru/companies/vivid_money/articles/534858/ (дата обращения: 05.05.2024)

9. Мареев, Н. А. Обзор брокеров сообщений в качестве инструмента обеспечения асинхронности в микросервисной архитектуре / Н. А. Мареев // Сборник трудов XVII Международной отраслевой научно-технической конференции «Технологии информационного общества» г. Москва, 02–03 марта 2023 г. – Москва, 2023. - С. 231-234.

10. Yipei, Niu. Load Balancing across Microservices / Yipei Niu, Fangming Liu, Zongpeng Li // Conference: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications 16-19 April 2018. DOI: 10.1109/INFOCOM.2018.8486300

Мариничев И.И., Шуватова Е.А., Землянская С.Ю. Разработка имитационной модели работы микросервисного приложения. *В статье рассмотрены проблемы, которые возникают при проектировании архитектуры микросервисного приложения, сформулирована задача разработки приложения с наиболее эффективной архитектурой. В связи с чрезмерной трудоемкостью проведения натурных экспериментов на разных моделях архитектур для исследования поведения приложения предлагается использовать объектную*

модель микросервисного приложения и имитационное моделирование с целью получения необходимых параметров работы и проведения анализа эффективности выбранной архитектуры. Выделены основные сущности модели, проанализированы их ключевые особенности и описаны принципы работы. Разработаны алгоритмы взаимодействия компонентов и функционирования имитационной модели.

Ключевые слова: *Микросервис, имитационная модель, заявка, распределение, функция.*

Marinichev I., Shuvatova E., Zemlyanskaya S. Development of a simulation model for the operation of a micro-service application. *The article considers the problems that arise when designing the architecture of a micro-service application, and the task of developing an application with the most efficient architecture is formulated. Due to the excessive complexity of conducting field experiments on different architectural models, it is proposed to use an object model of a microservice application and simulation modeling to study the behavior of an application in order to obtain the necessary operating parameters and analyze the effectiveness of the chosen architecture. The main entities of the model are highlighted, their key features are analyzed and the principles of operation are described. Algorithms for the interaction of components and the functioning of the simulation model have been developed.*

Keywords: *Microservice, simulation model, application, distribution, function.*

Статья поступила в редакцию 15.05.2024
Рекомендована к публикации профессором Скобцовым Ю. А.