**Troshin Alexander Victorovich,** Povolzhskiy State University of Telecommunications and Informatics, 23, L. Tolstoy Street, Samara, 443010, Russian Federation; Assistant Professor of Networks and Systems of Telecommunication Department, PhD in Technical Science. Tel. +7 846 339-11-26. E-mail: troshin-av@psuti.ru

**References**

1. Narayanan A. et al. Lumos5G: Mapping and predicting commercial MmWave 5G throughput. *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 176–193.
2. 5G. URL: https://ru.wikipedia.org/wiki/5G (accessed: 20.11.2021).
3. What is sub-6GHz, mmWave? Why is 5G needed? URL: https://habr.com/ru/post/524854 (accessed: 20.11.2021). (In Russ.)
4. 5G NR frequency bands. URL: https://en.wikipedia.org/wiki/5G_NR_frequency_bands (accessed: 20.11.2021).
5. Xu Z. et al. Efficient prediction of network traffic for real-time applications. *Journal of Computer Networks and Communications*, 2019, vol. 2019. DOI: https://doi.org/10.1155/2019/4067135
6. Kulkarni A. et al. DeepChannel: Wireless channel quality prediction using deep learning. *IEEE Transactions on Vehicular Technology*, 2020, vol. 69, no. 1, pp. 443–456. DOI: https://doi.org/10.1109/TVT.2019.2949954
7. Chandra R., Goyal S., Gupta R. Evaluation of deep learning models for multi-step ahead time series prediction. *IEEE Access*, 2021, vol. 9, pp. 83105–83123. DOI: https://doi.org/10.1109/ACCESS.2021.3085085
8. Huang C., Chiang C., Li Q. A study of deep learning networks on mobile traffic forecasting. *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–6. DOI: https://doi.org/10.1109/PIMRC.2017.8292737
9. Troshin A.V. Machine learning for traffic prediction in LTE network. *Infokommunikacionnye tehnologii*, 2019, vol. 17, no. 4, pp. 400–407. DOI: https://doi.org/10.18469/ikt.2019.17.4.06 (In Russ.)
10. Brownlee J. Taxonomy of Time Series Forecasting Problems. URL: https://machinelearningmastery.com/taxonomy-of-time-series-forecasting-problems/ (accessed: 20.11.2021).
11. Brownlee J. 4 Strategies for Multi-Step Time Series Forecasting. URL: https://machinelearningmastery.com/multi-step-time-series-forecasting/ (accessed: 20.11.2021).
12. Lumos5G: Mapping and Predicting Commercial MmWave 5G Throughput / A. Narayanan [et al.]. URL: https://lumos5g.umn.edu/ (accessed: 20.11.2021).
13. Troshin A. 5G Throughput Prediction. URL: https://github.com/av-troshin77/5gthroughput (accessed: 20.11.2021).

*Received 20.12.2021*

## ТЕХНОЛОГИИ КОМПЬЮТЕРНЫХ СИСТЕМ И СЕТЕЙ

УДК 004.057.4

### ОСОБЕННОСТИ КОММУНИКАЦИИ МИКРОСЕРВИСОВ ПРИ ИСПОЛЬЗОВАНИИ ШАБЛОНА САГА

*Малюга К.В., Перл И.А.*
*Национальный исследовательский университет ИТМО, Санкт-Петербург, РФ*
*E-mail: konstantin.malyuga@gmail.com*

Сегодняшние облачные решения очень часто строятся на основе микросервисной архитектуры. При таком подходе система реализуется в виде набора слабо связанных сервисов, которые не разделяют общего хранилища и обмениваются данными и командами по сети. Наряду с преимуществами такая структура системы привносит

свои недостатки, в частности – усложнение алгоритмов обеспечения согласованности данных. Известным решением проблемы является архитектурный шаблона сага, который подразумевает обеспечение итоговой согласованности, однако не формализует требования к свойствам отказоустойчивости системы, что может привести к отсутствию согласованности при изменении данных. В работе разбираются архитектурные особенности обеспечения отказоустойчивости в сагах в разрезе протоколов взаимодействия сервисов. Для этого был проанализирован ряд научных работ, посвященных сагам, и определены наиболее распространенные способы коммуникации. Для сформированного ряда протоколов были определены потенциальные уязвимости и предложены решения для сохранения отказоустойчивости в системе. Были показаны способы модификации взаимодействия сервисов с использованием стандартных реализаций протоколов для изменения наличия свойства синхронности взаимодействия микросервисов. Показаны преимущества использования таких технологий, как DLQ и CQRS, для формирования отказоустойчивых саг.

***Ключевые слова:*** *распределенные системы, микросервисы, протоколы коммуникации, сага, отказоустойчивость, согласованность данных*

## Введение

В микросервисной среде доступ к данным ограничен, поэтому только микросервис, реализующий логику определенной области приложения, имеет доступ к хранилищам данных этой области. Работа с данными из других областей системы производится через заранее оговоренные сетевые контракты (API). Такая структура позволяет независимо развивать участки приложения, изменяя способы хранения данных без нарушения сетевых контрактов, и открывает доступ к горизонтальному масштабированию системы для работы с высокой нагрузкой [1].

Однако микросервисная архитектура усложняет процессы, которые обеспечивают согласованность данных сразу на нескольких участках системы.

До прихода микросервисов данные из разных областей сохранялись в одной БД, и изменения в этих областях можно было провести атомарно и изолированно с помощью транзакций базы данных. При микросервисной архитектуре данные распределены сразу по нескольким БД, что не позволяет произвести простую транзакцию.

Известны способы организации распределенных транзакций сразу в нескольких БД, такие как Two-phase commit (2PC) [2]. Но такие подходы не получили широкого распространения при использовании микросервисов в системе ввиду наличия критических недостатков при существенных нагрузках на систему [3].

Альтернативой распределенным транзакциям является организация изменений в ключе итоговой согласованности данных. При таком подходе данные изменяются без изоляции и поэтапно. То есть согласованность обеспечивается не сразу на всех участках системы, а постепенно.

Эта идея была концептуализирована для микросервисных систем в архитектурном шаблоне с названием «*сага*».

Каждая сага – это набор *локальных транзакций* и *компенсаций*. Локальные транзакции в саге – это цепочка запросов на изменение данных в микросервисах. В случае возникновения ошибки применения локальной транзакции в одном из сервисов система должна применить компенсации во всех сервисах, которые уже выполнили локальные транзакции. Применение компенсаций призвано восстановить исходное состояние системы в случае ошибок.

В распределенном программном решении сага может быть реализована одним из двух способов [4].

– *Хореография саг* – при хореографии сага реализуется в каждом сервисе-участнике саги. Выполнив свою локальную транзакцию, сервис отправляет запрос следующему по цепочке сервису. Когда все следующие микросервисы выполнили свою часть изменений, они отправляют *подтверждение* об успешности выполнения предыдущему сервису. Полученное подтверждение позволяет сервису скомпенсировать изменения, если в цепочке вызовов произошла ошибка.

На рисунке 1 продемонстрирована цепочка выполнения локальных транзакций и компенсаций при реализации саги в хореографической форме. Сервисы 1 и 2 успешно выполняют локальные транзакции и передают управление сервису 3. При выполнении локальной транзакции на сервисе 3 происходит ошибка, из-за чего подтверждение с информаций об ошибке возвращается на предыдущие сервисы. Получив подтверждение, сервисы 2 и 1 применяют компенсации для возвращения текущего состояния к моменту до проведения локальных транзакций.

– *Оркестрация саг* – при оркестрации саг в системе реализуется отдельный сервис («*координатор*»), ответственный за управление ходом саг. Это позволяет произвести инкапсуляцию логики, связывающей различные области приложения, предотвращая изменение контролируемых

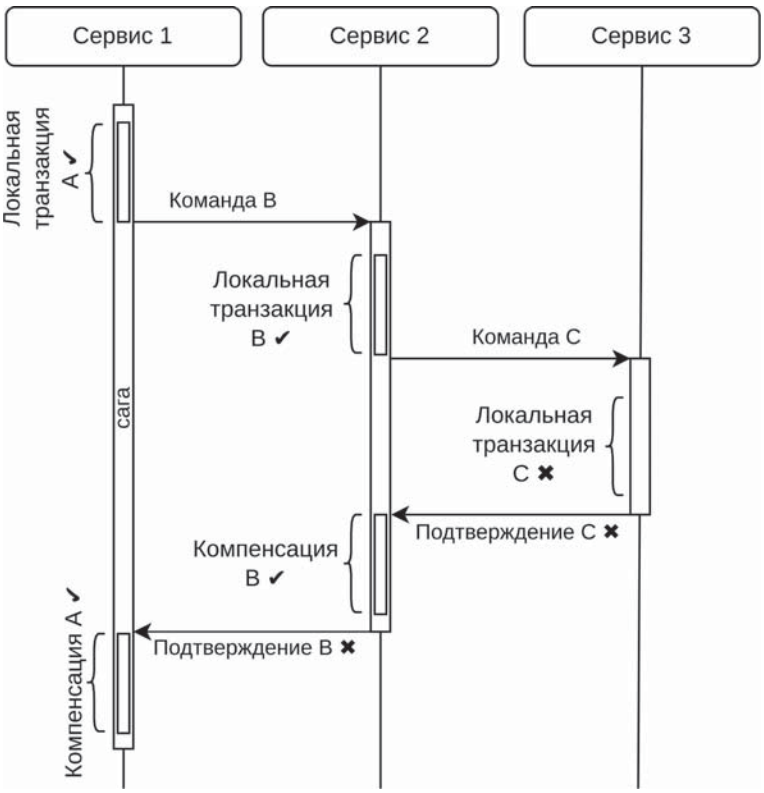


Рисунок 1. Обработка ошибки в локальной транзакции при хореографии саг

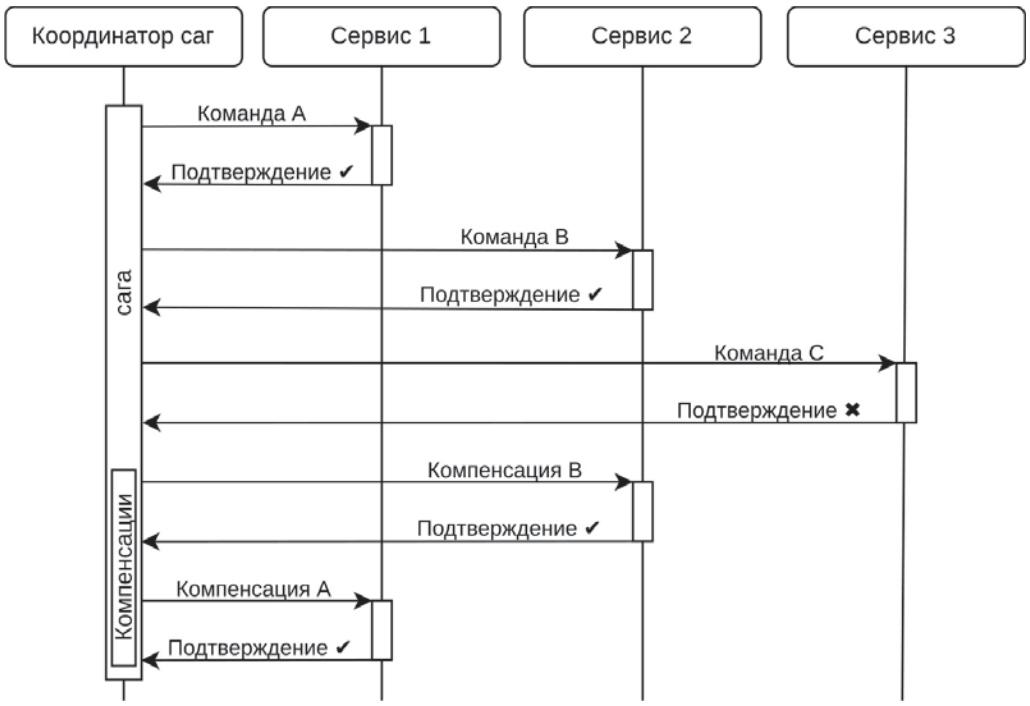


Рисунок 2. Обработка ошибки в локальной транзакции при оркестрации саг

микросервисов в случае изменения связывающей логики.

На рисунке 2 представлена аналогичная ситуация с ошибкой при проведении локальной транзакции саги в сервисе 3. Отличие заключается в том, что только координатор саг отправляет запросы на выполнение локальных транзакций, агрегирует подтверждения и производит запросы на выполнение компенсаций для сервисов, если это необходимо.

Важным аспектом выполнения саг является реакция системы на отказ одного из компонентов саги. Учитывая природу итоговой согласованности данных, когда модификация происходит

не атомарно, отказ одного из участков системы может привести к отсутствию согласованности в системе.

Отказоустойчивая структура саги должна позволить предотвратить ситуации такого рода в любом из подходов реализации саг. Поэтому в данном исследовании мы рассмотрели различные способы по обеспечению отказоустойчивости системы на основе саг, делая упор на канал передачи данных в системе.

## Проведенные исследования

Для рассмотрения аспектов отказоустойчивости в каналах коммуникации саг нами был проанализирован ряд последних научных работ, посвященных использованию саг в микросервисной архитектуре.

В [5] проводится разбор нескольких программных пакетов с разным способом организации коммуникации между сервисами.

- Axon Service – этот программный пакет построен для работы с CQRS (Command and Query Responsibility Segregation) архитектурой.

Axon Service использует систему событий из Spring Boot библиотеки для того, чтобы обеспечить коммуникацию в пределах одного сервиса. Но пользователь программного пакета может расширить его своим способом коммуникации. Это значит, что структура саги и способ организации локальных транзакций и компенсаций полностью обеспечиваются разработчиком, который внедряет пакет в систему.

- Eventuate Service – это программный пакет также создан для обеспечения принципов CQRS.

Протоколы для организации локальных транзакций в нем ограничены использование REST. Описание локальных транзакций и компенсаций также является ответственностью разработчика системы.

- Eventuate Tram Service – данный программный пакет поддерживает автоматическую отправку событий при выполнении изменений в БД. По умолчанию управление сообщениями реализовано на основе Apache Kafka. Отправленное событие приводит к активации следующей локальной транзакции. Eventuate Tram Service выполняет локальные транзакции одну за другой и производит компенсацию в случае возникновения ошибки в одной из них.

- LRA Service – в этом программном пакете организация взаимодействия с микросервисами осуществляется с помощью протокола HTTP. Структура саги определяется специалистом, адаптирующим пакет под систему.

В [6] авторы предложили свой программный пакет для организации саг. В их программной реализации используются 2 протокола для обеспечения коммуникации.

1. HTTP: для организации RESTful коммуникации с полезной нагрузкой в формате JSON для коммуникации сервисов;
2. CArtAgO: основанный на событиях способ коммуникации между сервисом и его управляющим агентом. Агент в SagaMAS – это сервис, который должен быть индивидуально развернут для каждого микросервиса в саге. Именно агенты обеспечивают выполнение саги.

Авторы [7] описывают обновление архитектуры одного из распределенных приложений.

В описанном проекте они предложили использовать брокер сообщений (RabbitMQ) для организации локальных транзакций.

Однако, как показано в статье, руководство проекта приняло решение не реализовывать предложенный протокол взаимодействия сервисов и обойтись коммуникацией в саге через реляционную базу данных, которая уже использовалась в системе.

В исследовании [8] авторы ссылаются на книгу [9]. В ней предлагается использование брокеров сообщения для формирования взаимодействия сервисов в саге. Основной указанной причиной использования именно брокеров сообщений является возможность предотвратить утерю сообщений в случае, когда получатель сообщения (сервис или координатор) временно не может его обработать.

В [10] также предложено использование протоколов, основанных на сообщениях.

В [11] представлен анализ Eventuate Tram Service (который был упомянут ранее) в сравнении с Netflix Conductor.

Последний программный пакет предлагает использовать REST как способ организации локальных транзакций в стандартной конфигурации.

Проведенная оценка публикаций показала, что авторы научных статей обращаются именно к уже популярным протоколам коммуникации в микросервисных средах. Популярность подходов в организации микросервисного взаимодействия среди разработчиков систем можно оценить по результатам опроса (рисунок 3), который провела компания JetBrains в 2021 году [12].

По результатам проведенного нами анализа становится заметно, насколько ответственность за формирование эффективной структуры саги смещена в сторону разработчика системы, что


Рисунок 3. Результат исследования компании JetBrains

увеличивает шанс на появление ошибок, которые могут привести к несогласованности данных.

Для более детального рассмотрения этих ошибок и определения способов их предотвращения мы предлагаем разобрать два аспекта организации взаимодействия саг в микросервисной среде: свойства протоколов взаимодействия сервисов и архитектурный компонент коммуникации в сагах.

### Подтверждение локальных транзакций

Для формирования эффективного взаимодействия сервисов в саге важно определить свойства ключевого элемента – подтверждений локальных транзакций и компенсаций.

Подтверждения в сагах нужны для (1) возможности определить завершение выполнения саги и для (2) передачи вызывающей системе информации о наличии или отсутствии ошибки при выполнении команды, чтобы вызывающая сторона могла среагировать на ошибку, например, компенсацией, если подтверждение относилось к команде.

Подтверждения также позволяют предотвратить компенсационные коллизии. Такого рода коллизии происходят из-за возможности параллельной обработки запросов на одном сервисе или из-за наличия нескольких экземпляров одного сервиса в системе.

Поэтому без системы подтверждений отправка компенсации после отправки запроса на проведение локальной транзакции не будет гарантировать установленный порядок выполнения.

Как показано на рисунке 4, запрос на проведение компенсации может быть применен раньше запроса на проведение локальной транзакции, так как локальная транзакция выполняется параллельно и ее применение может занять больше времени.

Наряду с уже указанными факторами возможность сворачивания реплик микросервисов [13] увеличивает вероятность невыполнения локальных транзакций и компенсаций. Ожидание подтверждений от сервисов устраняет и эту проблему для протоколов, которые не поддерживают автоматическое повторение извлечения данных [14].

### Используемые протоколы

В данной работе рассматриваемые протоколы коммуникации сервисов разделены на клиент-серверные и протоколы на основе очередей сообщений, как показано в таблице.

Клиент-серверные протоколы рассмотрены в контексте синхронного выполнения команд и компенсаций, т. е. выполнение запроса происходит именно в момент коммуникационной сессии. А очереди сообщений ассоциированы с асинхронным выполнением запроса на стороне микросервиса.

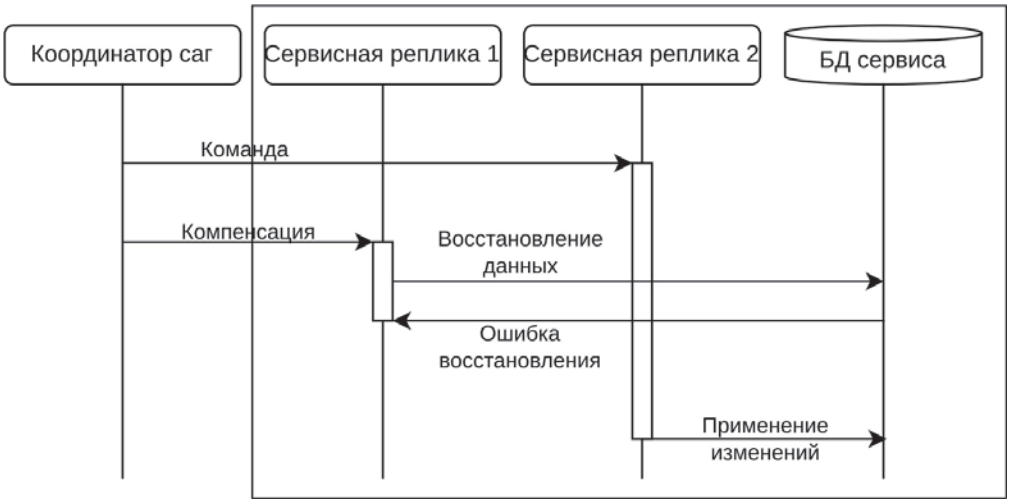


Рисунок 4. Обработка компенсации до обработки команды, несмотря на более раннее получение команды

Таблица. Рассматриваемые протоколы

| Тип | Название | Историчность | Повтор при ошибках |
|---|---|---|---|
| Клиент-серверные | HTTP | – | – |
| | CoAP | – | – |
| Очереди сообщений | Kafka | Сохранение в файл | DLQ |
| | RabbitMQ | – | DLQ |
| | ActiveMQ | – | DLQ |
| | Redis | – | – |

Такое отношение типа протокола к свойству синхронности выполнения не является строгим и поддается корректировке. Его можно изменить при внедрении дополнительной сложности в алгоритм взаимодействия систем в саге.

## Синхронное выполнение локальных транзакций

Стандартное поведение систем с использованием клиент-серверных протоколов, таких как HTTP, предполагает блокировку клиентской части после отправки запроса до момента получения ответа. Известны программные реализации клиент-серверных протоколов, которые позволяют производить другие операции до получения ответа, что может позволить справляться с высокой нагрузкой [15]. Однако получение ответа в клиент-серверных протоколах – это ключевая часть контракта между участниками взаимодействия.

Получение ответа от системы может быть адаптировано для роли подтверждения выполнения локальной транзакции. Но при таком использовании ответа на передний план выходит проблема отказа одной из сторон взаимодействия. Отказ возможен на стороне координатора после отправки запроса или на стороне сервиса-участника саги, после того как тот получил запрос на выполнение локальной транзакции/компенсации. В таком случае серверный ответ уже не удастся извлечь даже после восстановления отказавшего участка.

Реализация свойств идемпотентности позволяет координатору повторить запрос после восстановления и добиться подтверждения без внесения излишнего дублирования изменений.

Как показано на рисунке 5, сервис может отказать уже после получения команды и выполнения изменений, но перед отправкой подтверждения. В случае если реализация выполнения команды представлена в идемпотентном варианте, то повторное получение команды не приводит к повтору изменений данных. Оно приводить лишь к формированию и отправке подтверждения выполнения команды.

Альтернативой можно считать сохранение запроса в сервисе, что позволит обработать его асинхронно. Этот подход сходен с архитектурными решениями на основе CQRS, что позволяет вернуться к обработке запроса в случае ошибки на стороне сервиса (рисунок 6). При таком подходе ответ клиент-серверного протокола не играет существенной роли в коммуникации [16].

## Асинхронное выполнение локальных транзакций

Известными протоколами для организации асинхронного взаимодействия на основе очередей сообщений являются MQTT, AMQP и различные реализации, такие как: Kafka, Rabbit MQ, Apache ActiveMQ, Redis.

При их использовании в сагах сообщения могут уведомлять как о команде и компенсации, так

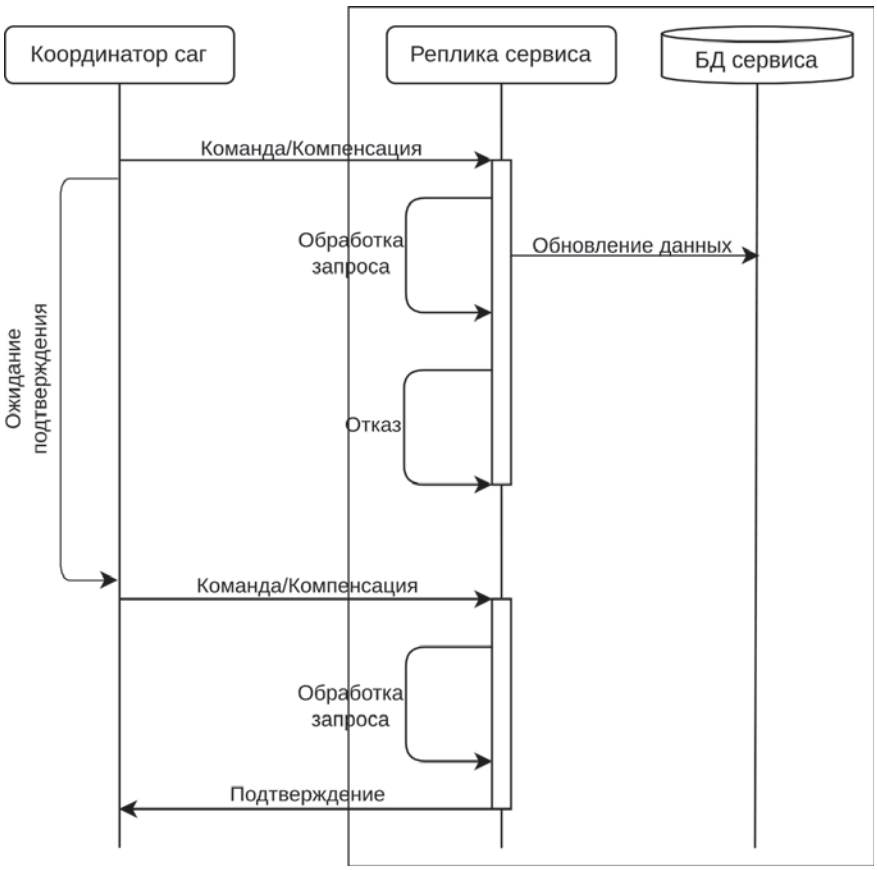


Рисунок 5. Повтор команды при ошибке на стороне сервиса

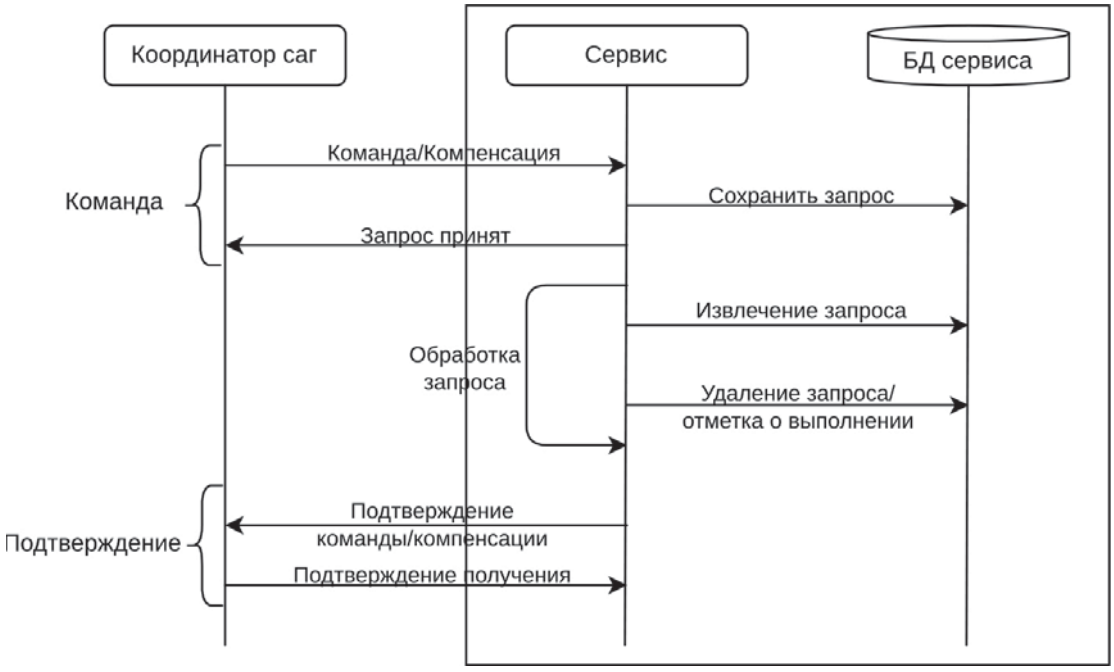


Рисунок 6. Асинхронное применение клиент-серверного протокола

и о подтверждении проведения команды или компенсации.

На фоне клиент-серверных протоколов очереди сообщения выгодно отличаются возможностью повторного извлечения сообщения после восстановления системы в случае отказа. Это позволяет не повторять запрос на стороне, запрашивающей выполнение команды. Такая возможность предоставляется некоторыми очередями на основе технологий, таких как Dead Letter Queue (DLQ).

На рисунке 7 представлен пример повторной обработки сообщений с помощью DLQ в случае

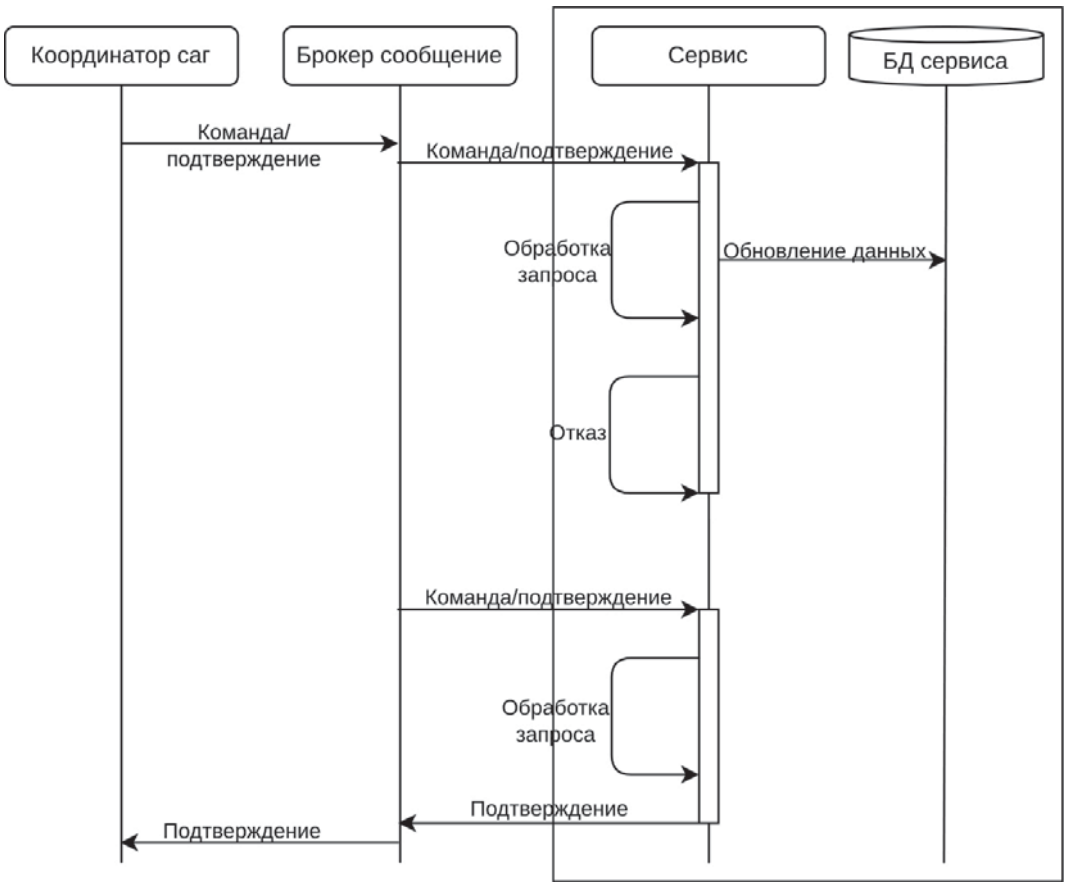


Рисунок 7. Повтор команды при применении брокера сообщений с DLQ

отказа на стороне сервиса. Здесь брокер сообщений позволяет повторно извлечь сообщение, для которого не было сформировано подтверждение. При этом координатор остается изолирован от процесса повторной обработки сообщения.

Некоторые очереди сообщений также предоставляют возможность извлечь историчные данные [16], что позволит восстановить причину возникновения ошибок в таких ситуациях, как конфликт при применении компенсации.

CQRS-архитектура часто совмещается с событийно ориентированной архитектурой [15]. Восстановление прогресса выполнения саги при такой структуре системы возможно и без дополнительных функций, предоставляемых очередями сообщений.

## Дальнейшее исследование

Построение архитектуры системы с учетом возможных ошибок в каналах коммуникации различных участков системы позволит повысить показатели отказоустойчивости при выполнении саг, однако не будет гарантировать согласованность данных. Отсутствие изоляции внесенных изменений при выполнении саги подразумевает возможность параллельного появления несовместимых модификаций, что приведет к невозможности произвести компенсацию. В таком случае данные будут оставлены в несогласованном состоянии до момента ручной обработки ошибки.

По этой причине изучение свойств описанных коллизий при выполнении компенсаций позволит продвинуться в разработке методов формирования архитектуры саг с обеспечением максимальных показателей согласованности данных.

## Заключение

В работе был представлен разбор наиболее используемых способов коммуникации микросервисов в контексте разработки архитектуры систем с использованием саг.

Показаны структурная разница протоколов и влияние этой разницы на отказоустойчивость системы. Для этого способы коммуникации были разделены на синхронные и асинхронные с указанием преимуществ и недостатков для формирования отказоустойчивой архитектуры системы.

Были представлены способы адаптации разных типов протоколов коммуникации в сагах для соответствия требованиям отказоустойчивости в системах. Также были продемонстрированы методы изменения структуры взаимодействия при использовании клиент-серверных протоколов и очередей сообщения для применения протоколов

в асинхронном и синхронном варианте соответственно.

Работа призвана помочь исследователям и инженерам в области распределенных облачных вычислительных решений на основе микросервисов.

### Литература


1. Brown K., Woolf B. Implementation patterns for microservices architecture // Proceedings of the 23rd Conference on Pattern Languages of Programs (PLoP '16). 2016. P. 7-1–35.
2. Two-phase commit optimizations and tradeoffs in the commercial environment / G. Samaras [et al.] // Proceedings of IEEE 9th International Conference on Data Engineering. 1993. P. 520–529.
3. Towards dependable agent systems / J. Nimis [et al.] // Multiagent Engineering. 2006.
4. Rudrabhatla C. Comparison of event choreography and orchestration techniques in microservice architecture // International Journal of Advanced Computer Science and Applications. 2018. P. 9.
5. Štefanko M., Chaloupka O., Rossi B. The saga pattern in a reactive microservices environment // 14th International Conference on Software Technologies. 2019. P. 483–490.
6. SagaMAS: A Software framework for distributed transactions in the microservice architecture / X. Limón [et al.] // 6th International Conference in Software Engineering Research and Innovation (CONISOFT). 2018. P. 50–58.
7. Assessing migration of a 20-year-old system to a micro-service platform using ATAM / P. Cruz Navea [et al.] // IEEE International Conference on Software Architecture Companion (ICSA-C). 2019. P. 174–181.
8. Microservice patterns for the life cycle of industrial edge software / F. Li [et al.] // EuroPLoP '18: Proceedings of the 23rd european conference on pattern languages of programs. 2018. P. 1–11.
9. Richardson C. Microservices patterns. Shelter Island: Manning, 2018. 520 p.
10. Li Z., Liang P., Avgeriou P. Architectural technical debt identification based on architecture decisions and change scenarios // Proceedings 12th Working IEEE/IFIP Conference on Software Architecture, WICSA. 2015. 63 p.
11. Dürr K., Lichtenthäler R., Wirtz G. An evaluation of saga pattern implementation technologies // CEUR Workshop Proceedings. 2021. 74 p.
12. Микросервисы // JetBrains. URL: https://www.jetbrains.com/ru-ru/lp/devecosystem-2021/microservices (дата обращения: 19.10.2021).
13. Deploying microservice based applications with Kubernetes: Experiments and lessons learned / V. Leila [et al.] // IEEE 11th International Conference on Cloud Computing (CLOUD). 2018. P. 970–973.
14. Fault tolerant central saga orchestrator in RESTful architecture / K. Malyuga [et al.] // Proceedings of the XXth Conference of Open Innovations Association FRUCT. 26. 2020. P. 278–283.
15. Knoche H. Improving batch performance when migrating to microservices with chunking and coroutines // Softwaretechnik-Trends. 2019. No. 39 (4). P. 20–22.
16. An empirical characterization of event sourced systems and their schema evolution – Lessons from industry / M. Overeem [et al.] // Journal of Systems and Software. 2021. 178 p.


*Получено 24.10.2021*

**Малюга Константин Владимирович,** аспирант факультета программной инженерии и компьютерной техники (ПИиКТ) Университета ИТМО. 197101, Российская Федерация, г. Санкт-Петербург, Кронверкский пр., 49. Тел. +7 981 796-40-94. E-mail: konstantin.malyuga@gmail.com

**Перл Иван Андреевич,** к.т.н., доцент кафедры информатики и прикладной математики (ИПМ) Университета ИТМО. 197101, Российская Федерация, г. Санкт-Петербург, Кронверкский пр., 49. Тел. +7 950 005-23-44. E-mail: ivan.perl@itmo.ru

## ASPECTS OF MICROSERVICES COMMUNICATION WHEN USING SAGA TEMPLATE

*Maliuga K.V., Perl I.A.*
*ITMO University*
*E-mail: konstantin.malyuga@gmail.com*

Microservices architectural pattern «saga» is a common approach that addresses the problem of eventual consistency in modern back-end cloud solutions. However, sagas approach does not define a fault tolerant way of solving the data consistency problem of distributed systems that can lead to business data inconsistency. In this paper we analysed a number of scientific publications in order to determine the most common ways of sagas definition from the cross-service communication point of view. The defined set of communication protocols was split on client-server protocols and message queues. For them we collected potential pitfalls of data consistency guarantee as well as ways to provide fault tolerance in these environments. It was shown how to adopt these protocols for both synchronous and asynchronous communication approaches in sagas. The applicability of CQRS and DQL in communication within saga was also studied in the paper.

***Keywords:*** *distributed systems, microservices, data consistency, communication protocols, saga, fault tolerance*

**DOI:** 10.18469/ikt.2021.19.4.06

**Maliuga Konstantin Vladimirovich,** ITMO University, 49, bldg. A, Kronverksky Avenue, Saint Petersburg, 197101, Russian Federation; PhD student. Tel. +7 981 796-40-94. E-mail: konstantin.malyuga@gmail.com

**Perl Ivan Andreevich,** ITMO University, 49, bldg. A, Kronverksky Avenue, Saint Petersburg, 197101, Russian Federation; PhD in Computer Science, Associate Professor of Informatics and Applied Mathematics Department. Tel. +7 950 005-23-44. E-mail: ivan.perl@itmo.ru

**References**

1. Brown K., Woolf B. Implementation patterns for microservices architecture. *Proceedings of the 23rd Conference on Pattern Languages of Programs (PLoP '16)*, 2016, pp. 7-1–35.
2. Samaras G. et al. Two-phase commit optimizations and tradeoffs in the commercial environment. *Proceedings of IEEE 9th International Conference on Data Engineering*, 1993, pp. 520–529.
3. Nimis J. et al. Towards dependable agent systems. *Multiagent Engineering*, 2006.
4. Rudrabhatla C. Comparison of event choreography and orchestration techniques in microservice architecture. *International Journal of Advanced Computer Science and Applications*, 2018, p. 9.
5. Štefanko M., Chaloupka O., Rossi B. The saga pattern in a reactive microservices environment. *14th International Conference on Software Technologies*, 2019, pp. 483–490.
6. Limón X. et al. SagaMAS: A Software framework for distributed transactions in the microservice architecture. *6th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 2018, pp. 50–58.
7. Cruz P. et al. Assessing migration of a 20-year-old system to a micro-service platform using ATAM. *IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019, pp. 174–181.
8. Li F. et al. Microservice patterns for the life cycle of industrial edge software. *EuroPLoP '18: Proceedings of the 23rd European Conference on Pattern Languages of Programs*, 2018, pp. 1–11.
9. Richardson C. *Microservices Patterns*. Shelter Island: Manning, 2018, 520 p.
10. Li Z., Liang P., Avgeriou P. Architectural technical debt identification based on architecture decisions and change scenarios. *Proceedings 12th Working IEEE/IFIP Conference on Software Architecture, WICSA*, 2015, 63 p.
11. Dürr K., Lichtenthäler R., Wirtz G. An evaluation of saga pattern implementation technologies. *CEUR Workshop Proceedings*, 2021, 74 p.
12. Microservices. JetBrains. URL: https://www.jetbrains.com/ru-ru/lp/devecosystem-2021/microservices (accessed: 19.10.2021). (In Russ.)

13. Leila V. et al. Deploying microservice based applications with Kubernetes: Experiments and lessons learned. *IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 970–973.
14. Malyuga K. et al. Fault tolerant central saga orchestrator in RESTful architecture. *Proceedings of the XXth Conference of Open Innovations Association FRUCT*. 26, 2020, pp. 278–283.
15. Knoche H. Improving batch performance when migrating to microservices with chunking and coroutines. *Softwaretechnik-Trends*, 2019, no. 39 (4), pp. 20–22.
16. Overeem M. et al. An empirical characterization of event sourced systems and their schema evolution – Lessons from industry. *Journal of Systems and Software*, 2021, 178 p.

*Received 24.10.2021*

УДК 004

# ОЦЕНИВАНИЕ УСЛОВИЙ ЭФФЕКТИВНОГО ФУНКЦИОНИРОВАНИЯ КОМПЬЮТЕРНЫХ УЗЛОВ ПРИ ИЗМЕНЕНИЯХ ВХОДНОГО ТРАФИКА НА ОСНОВЕ СТОХАСТИЧЕСКИХ ПРОЦЕССОВ

*Ткаченко К.С.*
*Севастопольский государственный университет, Севастополь, РФ*
*E-mail: KSTkachenko@sevsu.ru*

Современные цифровые технологии позволяют организовать эффективное управление процессами в информационных контурах промышленных предприятий. При этом для самих компьютерных узлов существуют разнообразные системы контроля и мониторинга. На основе таких систем можно выполнить построение моделей, а затем, исходя из результатов моделирования, установить эффективный режим функционирования на компьютерном узле. Предлагается подход управления компьютерным узлом, основанный на использовании статистического моделирования процессов Ито. Такой подход позволяет производить моделирование компьютерного узла во времени и в зависимости от результатов потребления узлом ресурсов устанавливать на узле наиболее подходящий режим функционирования с учетом решения эксперта.

***Ключевые слова:*** *производственные предприятия, компьютерные узлы, моделирование*

## Введение

Компьютерные технологии способствуют повышению эффективности работы промышленных предприятий [1]. Внедрение компьютерных технологий в производство приводит к уменьшению количества используемой аппаратуры и повышению его надежности, появлению принципиально новых возможностей управления и самодиагностики, увеличению качества производимой продукции. Наиболее часто на современном производстве используются не отдельные компьютеры, а их сети. При таком подходе различные компьютеры могут отвечать за отдельные производственные участки, управление которыми должно происходить в реальном времени. Для управления в реальном времени необходимо формировать дискретные сигналы управления с определенной скоростью. Если скорость формирования будет недопустимо низкой либо высокой, то возможно несрабатывание отдельных технических блоков. Поэтому необходим выбор режимов управления отдельными компьютерами в автоматическом режиме.

## Управление компьютерными узлами

Управление отдельными блоками на промышленных предприятиях производится с использованием специальных математических моделей [2]. По этим моделями можно сформировать планы работ отдельных исполнительных механизмов. Во многих случаях считается, что точных методов формирования планов нет. Поэтому поиск решения задач планирования производится итерационными методами. Разрабатывается программное обеспечение, позволяющее численными методами находить существующие решения задач оптимизации, в свою очередь, на основе этих решений уже может быть произведено эффективное планирование проводимых работ. При эксплуатации указанного программного обеспечения необходимо учитывать существование определенных ситуаций, когда решение может не быть найдено при его фактическом существова-